Proyecto Fin de Carrera de Ingeniería Informática

Implementación de videovigilancia mediante streaming en aeropuertos

Universidad de La Laguna

Escuela Superior de Ingeniería y Tecnología
Sección de Ingeniería Informática

Alumno
Tomás R. González Barroso

Director
Jezabel Miriam Molina Gil
Co-Director
José Iván Santos González

Doña **Jezabel Miriam Molina Gil**, con N.I.F. 78.507.682-B profesora contratada Laboral Interina adscrita al Departamento de Ingeniería Informática y Sistemas de la Universidad de La Laguna, como tutora.

Don **José Iván Santos González**, con N.I.F. 78.637.989-T investigador adscrito al Departamento de Ingeniería Informática y Sistemas de la Universidad de La Laguna, como cotutor.

CERTIFICAN:

Que el presente Proyecto de Fin de Carrera de la Escuela Técnica Superior de Ingeniería Informática titulado **"Implementación de videovigilancia mediante streaming en aeropuertos"**, presentado por D. **Tomás Ricardo González Barroso**, con N.I.F. 43.376.938-L ha sido realizado bajo nuestra dirección en la Escuela Técnica Superior de Ingeniería Informática.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos que haya lugar se firma el presente en San Cristóbal de La Laguna, a 11 de Julio de 2016.

Jezabel Miriam Molina Gil José Iván Santos González

AGRADECIMIENTOS

A Pino Caballero Gil por la oportunidad de realizar este proyecto.

"A mi me es imposible, pero puede que…."

A la tutora Jezabel Miriam Molina Gil

y al cotutor José Iván Santos González

por su inestimable ayuda.

A mi pareja, por su paciencia, ánimo y apoyo incondicional.

A mi hijo por el tiempo prestado.

A mi familia por siempre estar presente.

Y a todos los que han aportado su granito de arena.

Tabla de contenido.

Índice de Figuras.

Índice de Tablas.

1.Introducción.

La seguridad en la actualidad es uno de los factores más importantes de la sociedad. Una falta de seguridad en un país implica una disminución de la actividad económica del mismo, no solo en la disminución del número de turistas que lo visitan, sino también en la falta de atractivo para empresarios que podrían invertir y montar sus empresas en ellos. Por este motivo, es normal que cada vez se invierta más en seguridad con el objetivo no solo de evitar problemas, sino de prevenir la posibilidad de sufrir algún ataque.

Durante estos últimos años se ha experimentado un aumento de ataques terroristas a diferentes países. En el año 2014, el número total de muertes por terrorismo se incrementó en un 80% respecto al año anterior. Éste es el mayor incremento anual en los últimos 15 años. Desde el comienzo del siglo XXI se ha multiplicado por más de nueve el número de muertes por terrorismo, pasando de 3.329 en el año 2.000 a 32.658 en año 2014. [3]. El pánico y la inseguridad que genera este incremento está justificado, no sólo porque se trata un terrorismo metódico y orquestado desde diferentes entes u organizaciones reconocibles e identificables, con una logística poderosa y precisa, avaladas por una determinada ideología y abanderadas en un fin concreto, sino porque también está conformado por acciones individuales, indiscriminadas, no sujetas a un patrón o perfil de conducta, fruto de los desórdenes individuales de una sociedad masificada. Este panorama obliga a tomar unas medidas de seguridad más extremas, más precisas e innovadoras, donde no sólo se cubra el riesgo real e inminente de un ataque terrorista, organizado o espontáneo, sino la implantación de unas medidas disuasorias y constantes que generen en el usuario una sensación de confianza y protección y en el posible terrorista la de imposibilidad de actuar.

El ámbito aeroportuario, un lugar de tránsito continuo y afluencia de usuarios, así como clave en el transporte de mercancías y pasajeros, se ha convertido en objetivo de estas acciones terroristas. En el modelo de seguridad y protección de los aeropuertos los medios asignados a la protección de sus infraestructuras incluyen desde recursos técnicos sofisticados, con equipamiento para la inspección de personas, equipaje de bodega, circuitos cerrados de televisión a lo largo de todo el recinto aeroportuario y sistemas anti-intrusión, hasta sistemas de detección y grabación. Todos estos sistemas, deben renovarse y actualizarse constantemente apoyándose de las últimas innovaciones tecnológicas. Sin embargo, todos estos recursos requieren un desembolso monetario que no todos los aeropuertos pueden permitirse.

Por otro lado, la industria de dispositivos móviles está atravesando un periodo de rápida transformación, mejora y reducción de costos. Ofreciendo unas prestaciones impensables hace unos años y abriendo nuevos campos de aplicación a estos

dispositivos, como por ejemplo la seguridad. Entre las prestaciones a destacar se encuentran su portabilidad, la capacidad de transmisión y la posibilidad de capturar vídeos e imágenes.

Teniendo en cuenta los requisitos de seguridad y gasto económico necesario para proporcionar seguridad a infraestructuras amenazadas continuamente, como son los aeropuertos, se propone este proyecto que tiene como objetivo proporcionar un sistema de videovigilancia que permite reducir costes utilizando para ello la tecnología móvil. El objetivo es proponer un nuevo paradigma de seguridad donde las cámaras de vigilancia, utilizadas hasta ahora para alertar sobre situaciones de emergencia u ofrecer visión remota a los vigilantes, ya no son solo fijas, sino que además, los propios vigilantes por medio de los dispositivos móviles actúan "como cámaras móviles" compartiendo su visión en todo momento. El sistema, que en este trabajo se propone, dispondrá de cámaras móviles a lo largo de todo el recinto aeroportuario, beneficiándose no solo de esta nueva visión móvil sino, de la inteligencia que aporta la actuación humana, capaz de detectar situaciones que una máquina no reconocería.

Por lo tanto, la finalidad de este proyecto, es por un lado, utilizar el potencial que ofrecen los dispositivos móviles para aportar nuevas funcionalidades a los sistemas de videovigilancia aeroportuaria, y por otro lado, aprovechar los avances en las nuevas tecnologías, con el apoyo de plataformas de código abierto, para crear un sistema de videovigilancia que nos aporte seguridad, eficiencia, versatilidad y bajo coste.

2. Estado del Arte.

Los actuales sistemas de videovigilancia aeroportuaria, tienen una arquitectura común, la de los sistemas CCTV (Circuito Cerrado de Televisión) basados en IP. Los componentes principales son:

- **Cámara IP.-** Captura el vídeo y el audio (en caso de incorporar entrada y salida de audio) y puede ser fijas o móviles, estando conectadas por cable o en modo inalámbrico a una red de datos IP, a través de la cual se puede controlar y almacenar la información en NVRs (Network Video Recorder) o servidores de vídeo en red.
  Las cámaras pueden ser fijas, con zoom, móviles o PT (Pan, Tilt) o PTZ (Pan, Tilt, Zoom) como por ejemplo las llamadas domo, debido a la forma de domo invertido que presentan, y las cámaras con posicionador, que pueden ser remotamente movibles. Este movimiento se puede hacer mediante una consola o teclado mediante el cual se pueden manejar las diversas opciones del software instalado en ésta.
- **NVR (Grabador de vídeo en red) / VMS (Sistema de gestión de video).**- Elemento que permite grabar y/o visualizar la imagen procedente de una o múltiples cámaras tanto localmente (dentro de una red de área local) como remotamente (a través de internet). Estos elementos que pueden ser elementos hardware con software embebido o bien elementos puramente software que se ejecuta en un servidor. También aportan otras funcionalidades como la gestión de accesos y permisos de usuarios o la configuración remota de las cámaras, por poner algunos ejemplos.
- **Grabador de vídeo.-** La grabación puede ejecutarse de manera continua o programada automáticamente por horas, activación por movimiento, detección de eventos específicos, etc.
- **Video Server Encoder.**- Permiten conectar cámaras analógicas CCTV a una red digital de videovigilancia basada en el protocolo IP
- **Software de análisis de vídeo.**- Permite análisis automáticos de las imágenes en función de los parámetros previamente definidos por el usuario. Estas capacidades hacen que los usos de las videovigilancia vayan más allá de la seguridad física, pudiendo aplicarse a inteligencia de negocio. Las nuevas versiones de este software permite, por vía de avanzados algoritmos en el análisis de vídeo, definir parámetros de grabación para que las cámaras únicamente capturen imágenes cuando detecten determinados eventos, lo que optimiza la capacidad de almacenamiento y el consumo del ancho de banda. Estos sistemas son capaces de abordar las tareas de grabación y transmisión de más de 64 cámaras, dependiendo de los requerimientos de tasas de bits y resolución, salvando todo el vídeo en una red de almacenamiento o en discos externos. La gestión del parque de cámaras IP instaladas puede realizarse de

manera centralizada desde un único punto y, gracias al protocolo IP, puede hacerse incluso en modo remoto.

Figura 2.1. Arquitectura CCTV

- **Dispositivos de visualización.-** Los dispositivos más extendidos son los tradicionales monitores o pantallas, PCs o video-walls. Sin embargo y dada la versatilidad del protocolo IP, es posible visualizar las imágenes en dispositivos de bolsillo, como teléfonos móviles y tablets.
- **LED infrarrojos.-** Los LEDs infrarrojos son puntos generadores de luz infrarroja. Este tipo de luz es imperceptible para el ojo humano pero no para Cámaras IP que incorporen filtros infrarrojos, dotando así a la cámara de visión nocturna.
- **Carcasas exteriores.-** Son elementos que protegen las cámaras de exterior frente a inclemencias climatológicas y/o acciones de vandalismo.
- **Sensores**.- Dispositivos que contribuyen a ajustar las grabaciones automáticas en función de determinadas condiciones, como cambios de temperatura, sonido o movimiento, entre otros, pudiendo además activar funcionalidades como la iluminación de infrarrojos (IR) cuando así lo requiera el grado de oscuridad en el lugar de grabación.

En la actualidad, existe un número elevado de empresas que disponen de soluciones de videovigilancia para aeropuertos. Algunas ofrecen todos los componentes mencionados, y otras en cambio, se especializan en alguno, como en las cámaras, o en el software de análisis y gestión de video.

Esas empresas multinacionales, ofrecen productos con altas prestaciones, servicios, herramientas y hardware específico, que cubren las necesidades de seguridad.

Estas son algunas de las empresas más destacadas:

- **Bosh Security System. [1].** Una de las soluciones con mayor prestigio internacional. Ofrece una solución integral y específica para aeropuertos. Cubre todos los componentes: cámaras, todo tipo de sensores, grabadores y un potente software de Inteligencia Artificial.

  Es una solución implantada en muchos aeropuertos internacionales, se puede encontrar en Madrid-Barajas.

Figura 2.2. Bosh Security System.

- **Eneo** [2]. Otra empresa alemana con gran reconocimiento internacional. Ofrece también una solución integral y específica de seguridad para los aeropuertos, cámaras (red, térmicas, HD..), videograbadores y software inteligente. Lo podemos encontrar en el aeropuerto de Weeze en Alemania.

- **Mileston** (XProtect Advanced) [9]. Esta empresa, está especializada en soluciones de software inteligente para sistemas de seguridad. Ofrece una plataforma de software que permite aprovechar la infraestructura existente routers, cámaras, sensores, etc. e integrarlos en un sistema único de gestión. Instalado en los Aeropuertos Varna y Burgas, Bulgaria.

Figura 2.3. Milestone XProtect Advanced.

Todas las soluciones que se encuentran en el mercado, se caracterizan principalmente por tener un alto coste, la utilización de software propietario, que impide introducir nuevas funcionalidades sin costo, y por la infravaloración de los dispositivos móviles. Limitando su uso a los smartphones, para la visualización y manejo de cámaras fijas-móviles, y/o a funcionalidades básicas sobre el servidor central.

La solución de videovigilancia propuesta se distingue y mejora esos aspectos.

- Abarata los costes de la implantación utilizando software libre.
- Flexible. La alta modularidad del software libre permite añadir nuevas funcionalidades sin costo.
- Aprovecha el potencial de los dispositivos móviles, como posibles cámaras en movimiento, como intercomunicadores entre vigilantes y con el uso de Realidad Aumentada, siendo capaces de dar una visión del entorno físico del mundo real con información virtual añadida, creando una realidad mixta entre lo real y lo digital.
- Seguridad y Eficiencia. Maximiza la optimización y seguridad de la infraestructura de red, ya que utiliza un nuevo conjunto de protocolos de comunicación que ofrece WebRTC [12]. De esta manera, se mejora por un lado, la eficiencia en la transmisión de audio y video, y por otro, aporta una capa extra de seguridad. Todo el tráfico se transmite cifrado utilizando protocolos estandarizados y reconocidos (SSL/TLS, DTLS, SRTP).

3.Tecnologías.

Para el desarrollo del sistema se han empleado fundamentalmente herramientas y plataformas de código abierto, inspiradas en el lenguaje Java y JavaScript. Con ello se ha tratado de agilizar el proceso de desarrollo utilizando "el mismo lenguaje" en los distintos componentes y herramientas que integran el sistema.

La elección de éstas tecnologías, está basada en términos de, **menor coste** (opensource), mayor **eficiencia** (NodeJs), **usabilidad-rendimiento** (WebRTC, socket.io, express.js), mayor cuota de mercado (Android) y **modularidad** (Kurento).

A continuación se detallan las tecnologías empleadas.

3.1 Android.

Android [13] es un sistema operativo basado en el núcleo Linux. Fue diseñado principalmente para dispositivos móviles con pantalla táctil, como teléfonos inteligentes, tablets; y también para relojes inteligentes, gafas inteligentes, televisores y automóviles. Inicialmente fue desarrollado por Android Inc., empresa que Google respaldó económicamente y más tarde, en 2005, compró. Android fue presentado en 2007 junto la fundación del Open Handset Alliance (un consorcio de compañías de hardware, software y telecomunicaciones) para avanzar en los estándares abiertos de los dispositivos móviles.

En la actualidad existen aproximadamente 1.000.000 de aplicaciones para Android y se estima que 1.500.000 teléfonos móviles se activan diariamente, ya en 2013 se llegó a los 1.000 millones de teléfonos inteligentes Android en el mundo.

La compañía de investigación y análisis de tecnología Gartner reveló en su último informe respecto al estado del mercado de teléfonos móviles en el segundo cuarto de 2015, que los dispositivos Android son los más dominantes, ocupando un 82.2% de la cuota de mercado, seguido por Apple con un 14.6% y con Microsoft cerrando el podio, aunque reduciendo su participación respecto al año pasado, con un 2.5%. [6]

3.2 Express.

Express.js [4] es un framework para Node.js que proporciona algunas características que facilitan en gran medida el desarrollo de sitios web. Dispone de un intérprete de comandos con el que se pueden crear esqueletos de proyectos web

básicos que incorporan elementos tales como soporte para sesiones de usuario o un sistema de renderización de plantillas para las vistas de la aplicación. Incorpora asimismo otros elementos básicos para la gestión del flujo de datos a través del sitio web como por ejemplo: mapeo de rutas URL, objetos de solicitud y respuesta HTTP, autenticación y seguridad básicos, soporte para cookies, etc.

Express.js se emplea como herramienta sobre la que desarrollar y desplegar el servidor de señales.

3.3 JavaScript.

Javascript [16] es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Es orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.

Se utiliza principalmente del lado del cliente, implementado como parte de un navegador web. De ese modo permite mejoras en la interfaz de usuario y páginas web dinámicas. No obstante, existe también una forma de JavaScript del lado del servidor. Su uso en aplicaciones externas a la web, por ejemplo en documentos PDF, aplicaciones de escritorio es también significativo.

JavaScript se diseñó con una sintaxis similar al C, aunque adopta nombres y convenciones del lenguaje de programación Java. Sin embargo Java y JavaScript no están relacionados y tienen semánticas y propósitos diferentes.

JavaScript soporta un formato ligero para el intercambio de datos llamado JSON (JavaScript Object Notation) [17] con el que es posible representar en texto plano objetos del lenguaje mediante pares campo-valor.

Todos los navegadores modernos interpretan el código JavaScript integrado en las páginas web. Para interactuar con una página web se provee al lenguaje JavaScript de una implementación del Document Object Model (DOM).

3.4 Kurento.

Kurento [8] es un servidor multimedia de código abierto y un conjunto de APIs de cliente que facilitan el desarrollo de aplicaciones de vídeo avanzadas para las plataformas de teléfonos inteligentes y web. El desarrollo de Kurento comenzó en 2010 en la Universidad Rey Juan Carlos de Madrid como el resultado de una serie de grandes proyectos de investigación que se ocupan de los sistemas de comunicación multimedia en tiempo real.

3.4.1. Arquitectura y módulos de Kurento.

En este apartado se muestra la arquitectura general del servidor Kurento, y su funcionamiento, así como los módulos actualmente disponibles.

Kurento está basado en el principio de modularidad, tiene como base tres módulos que forman el núcleo, *kms-core* (componentes principales), *kms-elements* (componentes multimedia) y *kms-filters* (componentes adicionales), que ofrecen la funcionalidad básica del servidor, y luego diferentes módulos que se pueden ir "conectando" para ir añadiendo nuevas funcionalidades, entre ellos se pueden citar:

- kms-FaceOverlayFilter. Reconocimiento de rostros.
- kms-CrowDetector. Detección de aglomeraciones.
- kms-PointDetector. Detección de puntos basado en color.
- kms-PlateDetector. Reconocimiento de matrículas de coches.
- kms-RecorderEndPoint. Almacenamiento multimedia.

También al ser una plataforma "abierta", tiene la posibilidad de desarrollar y conectar un módulo propio, así como modificar los existentes.

El esquema general de Kurento se ilustra en la siguiente imagen.

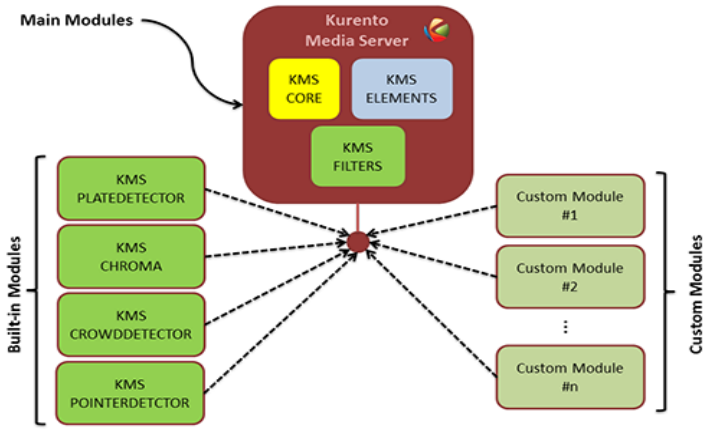

Figura 3.1. Esquema servidor Kurento.

En la siguiente tabla se describen los módulos disponibles .

Módulo	**Descripción**
WebRtcEndpoint	Permite comunicación bidireccional con WebRTC
HttpPostEndpoint	Convierte a formato WebM (HTML5)
RtpEndpoint	Permite comunicación bidireccional con el protocolo RTP.
PlayerEndpoint	Lee secuencias de video desde fichero.

RecorderEndpoint	Graba secuencias de video a fichero.
FaceOverlayFilter	Detección de rostros.
ZBarFilter	Detección códigos de barras y QR.
Composite	Permite crear una matriz con fuentes diferentes
Dispatcher	Hub permite múltiples entradas con múltiples salidas.
DispatcherOneToMany	Hub permite distribuir una entrada a varias salidas.
PointDetectorFilter	Detecta puntos en base a un color.
ChromaFilter	Filtra los colores especificados de la imagen.
CrowdDetectorFilter	Detecta aglomeraciones de personas.
PlateDetectorFilter	Reconoce las matrículas de coches.

Tabla 3.1. Módulos Kurento.

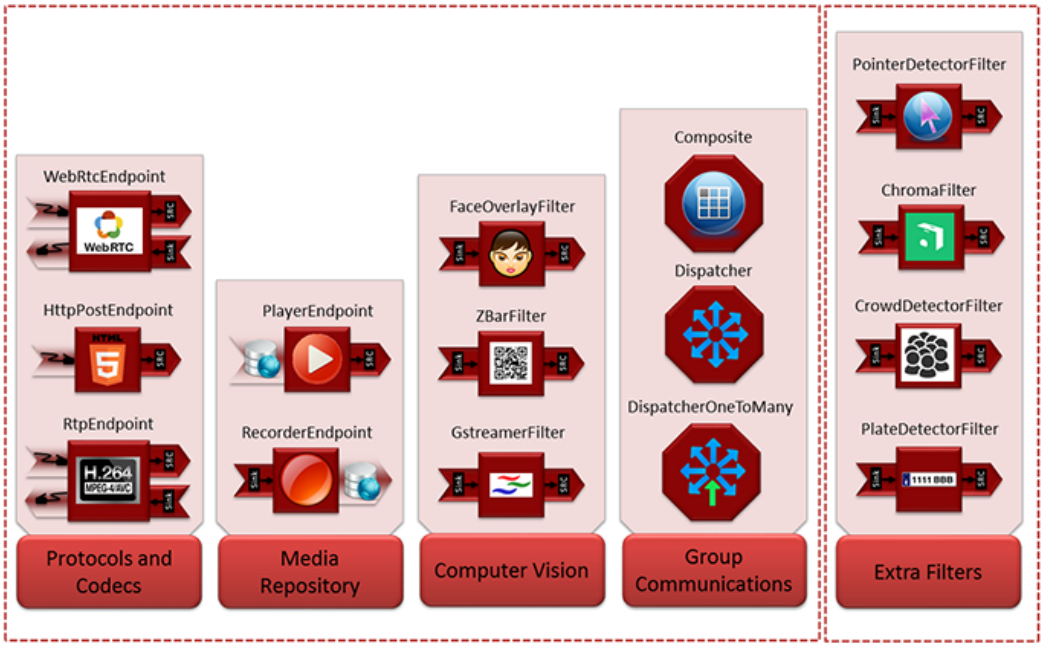

Figura 3.2. Módulos Kurento.

La facilidad de uso, la gran cantidad de módulos disponibles, así como el soporte de WebRTC son los motivos que han llevado a elegir este servidor multimedia para el sistema.

3.5 Node.js

Node.js[10][14] es un entorno en tiempo de ejecución JavaScript, multiplataforma, de código abierto, para la capa del servidor. Emplea un modelo dirigido por eventos no bloqueado por las operaciones de entrada/salida que lo hace ligero y eficiente, perfecto para aplicaciones de uso intensivo de datos en tiempo real que se ejecutan en sistemas distribuidos.

3.6 Socket.io

Socket.io [11] es un framework para la comunicación web en tiempo real. Proporciona una forma sencilla de establecer una comunicación a través de WebSocket. Socket.io dispone librerías tanto para el servidor como para el cliente. Se utilizará para la comunicación inicial entre la App Android y el servidor de señales.

3.7 WebRTC.

Esta tecnología es la piedra angular del proyecto, y sobre la que se ha construido el sistema, por ello, se realizará una descripción más amplia y que abarcará varios sub-apartados.

WebRTC [15] (Web Real-Time Communication) es una API que está siendo elaborada y estandarizada por la World Wide Web Consortium (W3C), APIS web, y el Internet Engineering Task Force (IETF), protocolos y códecs, para permitir comunicación en tiempo real punto a punto, a través del navegador y sin ningún complemento adicional.

En la actualidad, WebRTC se encuentra en un estado avanzado de desarrollo y estandarización, con soporte en Chrome, Firefox y Opera. La mayoría de los fabricantes de infraestructura de red (ALU, Ericsson, Genband, Huawei, NSN, Oracle, etc.), han anunciado el soporte de WebRTC en sus productos. Además, un gran número de desarrolladores están trabajando en pruebas de concepto para varias aplicaciones.

WebRTC no define la parte de señalización, permitiendo a los fabricantes de sistemas VoIP emplear WebRTC independientemente del protocolo de señalización empleado para establecer la llamada. Las opciones más populares para la señalización son: **JSON (*JavaScript Object Notation*) sobre WebSockets** y **SIP (*Session Initiation Protocol*) sobre WebSockets**. De manera que da una mayor flexibilidad para adaptarlo a otros sistemas ya existentes.

En el sistema se utilizará JSON sobre WebSockets.

3.7.1 Arquitectura.

En el siguiente diagrama se mostrará la arquitectura general de las APIs de WebRTC. Los componentes dentro de WebRTC, tal y como se muestra en la Figura 2.5, abarcan los códecs, motores de medios y capa de transporte. Esto facilita el uso de comunicaciones en tiempo real por parte de los desarrolladores de aplicaciones, sin necesidad de ser expertos en estas tecnologías.

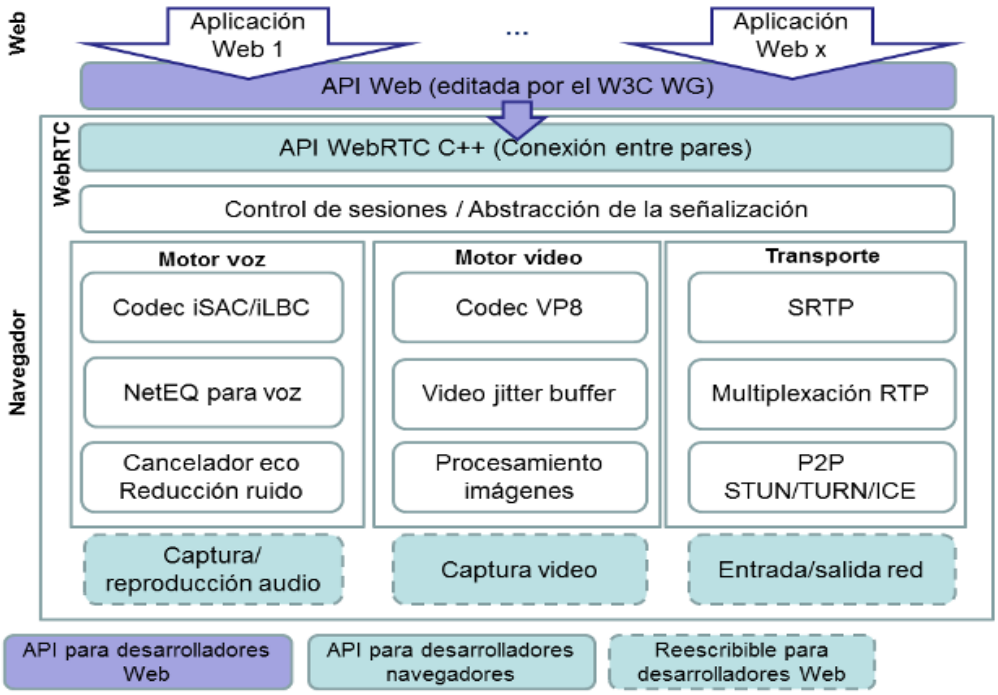

Figura 3.3. Arquitectura WebRTC.

Los principales componentes API de WebRTC son:

- **GetUserMedia**, permite obtener video y audio desde el dispositivo (cámara y/o micrófono). También permite obtener "capturas" de pantalla o incluso compartir la misma.
- **PeerConnection**, establece las conexiones multimedia audio / vídeo (p2p). Se encarga de una gran cantidad de tareas, como el procesamiento de señales, ejecución de códec, seguridad de la transmisión, etc..
- **DataChannels**, permite compartir cualquier tipo de datos a través de peer-to-peer (p2p). Utiliza comunicación bidireccional entre los clientes.

3.7.2 Códecs Audio y Video.

Debido a la naturaleza de las aplicaciones WebRTC, en tiempo real y/o interactivas, los códecs que utiliza están específicamente diseñados para estos propósitos, tienen latencias bajas y un cómputo operacional pequeño. Además, deben de ser de dominio público para encajar con la filosofía del diseño.

Los códecs para audio soportados son los siguientes:

Audio

OPUS. Opus es un compresor de audio con pérdidas de libre utilización desarrollado por la IETF. Este códec tiene un alto grado de ajuste, pudiendo elegir tasas de bits altas o bajas dependiendo de la calidad de servicio que se quiera

ofrecer. Internamente, funciona como un códec de predicción lineal para tasas de bit bajas y como un códec de transformación para tasas de bits altas. Tiene un retraso (Delay) muy bajo (22.5 ms por defecto), cabe decir, que los retrasos en los formatos más populares para música como MP3, Ogg Vorbis y AAC son un poco superiores a los 100 ms, además de que Opus está a la par en términos de calidad del sonido de estos últimos. Se implementa la compresión de audio con pérdida. Opus se puede utilizar tanto para tasas de bits bajas y altas. Tasa de bits compatibles: constante y variable, de 6 kbit/s a 510 kbit/s tasas de muestreo compatibles: de 8 kHz a 48 kHz. En las últimas versiones de Firefox y Chrome este codec se utiliza por defecto para la codificación de secuencias de audio.

iSAC. Internet Speech Audio Codec. Es un códec de banda ancha especialmente diseñado para trabajar con audio en formato voz, desarrollado por Global IP Solutions (GIPS) adquirida por Google en el año 2011, momento en el cual se liberó. Es un códec adecuado para aplicaciones VoIP y audio en streaming. Tasas de bits compatibles: adaptativo y variable: de 10 kbit/s a 52 kbit/s. Tasas de muestreo compatibles: 32 kHz. Bueno para datos de voz, pero no es adecuado para los flujos de audio de alta calidad.

iLBC.Internet Low Bitrate Codec. Es un códec de banda estrecha diseñado para aplicaciones de audio en formato voz. Este códec de audio es muy conocido, fue lanzado en 2004, y se convirtió en parte del proyecto WebRTC en 2011 cuando Google adquirió Global IP Solutions (la empresa que desarrolló iLIBC). Su algoritmo es una versión de la codificación predictiva lineal, con bloques independientes. Velocidades de bits compatibles: tasa de bits fija. 15,2 kbit/s (20ms), o 13,33 kbit/s (30ms), frecuencia de muestreo: 8 kHz/16 bits.

Video

VP8. El códec implementa una tecnología de alta eficiencia de compresión de vídeo. Originalmente, VP8 fue desarrollado por la empresa On2, pero ha sido adquirido por Google en 2010 y ahora el códec es parte del proyecto WebM, y es el único códec de vídeo disponible en WebRTC (por ahora).

VP9. Es un descendiente de VP8. Este códec muestra mejores resultados que VP8, pero necesita mucho más recursos del hardware. Google lo ha introducido en YouTube alegando que reducen un 35% ancho de banda de media por vídeo y que los vídeos comienzan entre un 15-80% más rápido.

H.264. MPEG-4 parte 10. El estándar H.264 es la opción dominante para los streamings de vídeo, pero tiene problemas de patentado que dificulta la implementación de código libre. Es un códec de vídeo de alta compresión, desarrollado conjuntamente por Video CodingExpertsGroup (VCEG) y Moving

Picture ExpertsGroup (MPEG). La intención del proyecto H.264/AVC fue la de crear un estándar capaz de proporcionar una buena calidad de imagen con tasas binarias notablemente inferiores a los estándares previos (MPEG-2, H.263), además de no incrementar la complejidad de su diseño.

H.265. MPEG-H parte 2. El formato H.265 ha sido desarrollado como el HEVC (High Efficiency Video Coding), a veces se denomina HEVC H.265, por la VCEG (Video Coding Experts Group) y la MPEG (Moving Picture Experts Group). Se estableció como el sucesor del H.264 (el códec más extendido) en 2013. El problema es que, como el H.264, los fabricantes de hardware han de pagar la licencia por agregar el soporte y los desarrolladores pagar una cuota.

3.7.3 Esquema de funcionamiento.

A continuación se describirá brevemente el funcionamiento, para ello se pasará a describir los protocolos utilizados por WebRTC junto con unas ilustraciones con muestran su uso.

SDP Session Description Protocol, es un protocolo para describir los parámetros de inicialización de los flujos multimedia. Corresponde a un protocolo actualmente redactado en el RFC 4566 por la IETF, de los anteriores RFC 2327 de abril de 1998 y RFC 3266 de junio de 2002.

SDP está pensado para describir sesiones de comunicación multimedia cubriendo aspectos como anuncio de sesión, invitación a sesión y negociación de parámetros. SDP no se encarga de entregar los contenidos propiamente dichos sino de entablar una negociación entre las entidades que intervienen en la sesión como tipo de contenido, formato, seguridad y todos los demás parámetros asociados. Este conjunto de parámetros se conoce como perfil de sesión. SDP se puede ampliar para soportar nuevos tipos de medios y formatos.

Ejemplo:

v=0 (Versión del protocolo)
o=- 778615731001943688 2 IN IP4 127.0.0.1 (Identificador del origen de la sesión)
s=- (Nombre de la sesión)
t=0 0
a=group:BUNDLE audio video
a=msid-semantic: WMS ARDAMS
m=audio 9 UDP\/TLS\/RTP\/SAVPF 111 103 9 102 0 8 106 105 13 127 126 (Tipo de datos)
c=IN IP4 0.0.0.0 (Información de la conexión)
a=rtcp:9 IN IP4 0.0.0.0 (atributos adicionales)
a=ice-ufrag:ecz00HAoZzFgZAtR (atributos adicionales cifrado)
a=ice-pwd:VncHlU7jrU0KKijJ9RjRVNvL (atributos adicionales cifrado)

a=fingerprint:sha-256 (atributos adicionales cifrado)
50:56:18:56:FF:77:9E:85:C8:06:35:C3:AB:72:9C:C8:98:5A:9D:8D:38:8F:64:85:64:24:26:89:27:D4:A5:96
a=setup:active
a=mid:audio (atributos adicionales)
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level
a=extmap:3 http:\/\/www.webrtc.org\/experiments\/rtp-hdrext\/abs-send-time
a=sendrecv
a=rtcp-mux
a=rtpmap:111 opus\/48000\/2 (atributos adicionales códec audio)

ICE Interactive Connectivity Establishment [7], es una técnica utilizada en NAT para establecer comunicación para VOIP, peer-peer, mensajería instantánea, y otro tipo de contenido multimedia. Típicamente "Ice Candidate" provee información sobre protocolo, la dirección IP y puerto desde donde se intercambiarán los datos.

Ejemplo:

a=candidate:1 1 UDP 2130706431 192.168.1.102 1816 typ host

A continuación se muestra el esquema de utilización de los protocolos SDP/ICE.

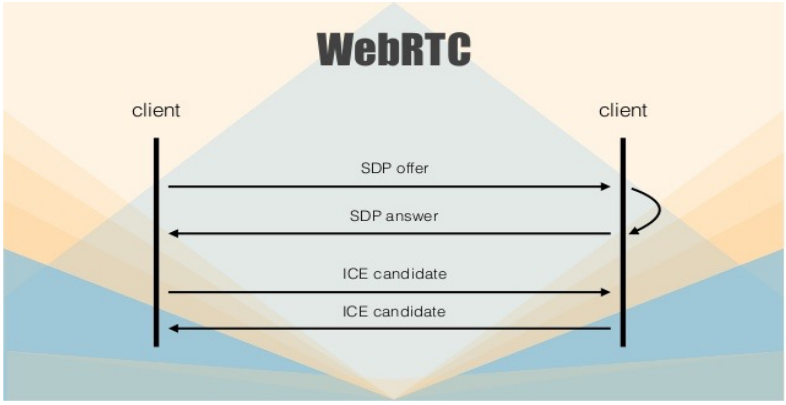

Figura 3.4. Uso de protocolos SDP/ICE en WebRTC.

Inicialmente se produce el intercambio de los parámetros de los flujos multimedia. El cliente A, envía su configuración disponible al otro cliente B (SDP offer), este le responde con sus propias características (SDP answer). Si se produce "acuerdo", se pasa al intercambio de parámetros de conexión de red (ICE candidate) en ambos clientes, a partir de ese momento empezaría el intercambio multimedia entre los clientes.

El siguiente esquema muestra, resumido, el proceso de establecimiento de una conexión WebRTC, así como las llamadas a las APIs.

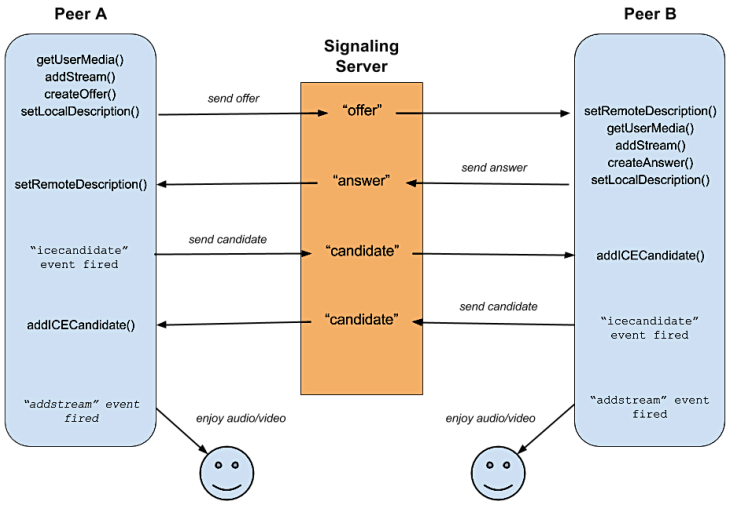

Figura 3.5. Esquema de funcionamiento WebRTC.

3.7.4 Seguridad.

El cifrado es una característica obligatoria de WebRTC, y se aplica en todos sus componentes, y se incluye también en los mecanismos de señalización. De consiguiente, todos los flujos de datos enviados, se cifran, a través de protocolos de encriptación estandarizados y conocidos.

El protocolo de cifrado que se utiliza depende del tipo de canal; los flujos de datos se cifran mediante **DTLS** y flujos multimedia se cifran mediante **SRTP**. Estos protocolos se describen al final de este apartado. WebRTC utiliza SRTP para el cifrado multimedia en lugar de DTLS, ya que es una opción más "ligera".

La utilización de este esquema en las conexiones punto a punto permite detectar cualquier ataque MITM (Man in the Middle).

El protocolo de señalización no se define en WebRTC, en el sistema, para garantizar la seguridad de la comunicación, se utilizará HTTPS y WSS (Secure WebSocket).

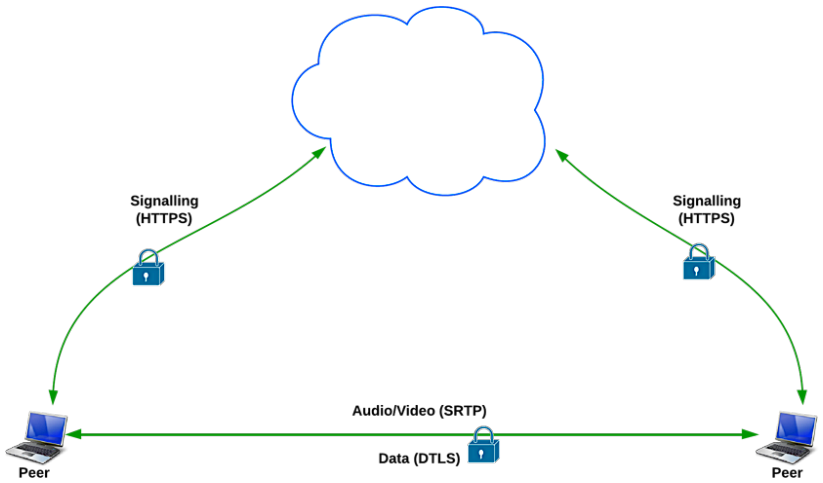

Figura 3.6. Esquema de seguridad WebRTC.

La ilustración muestra los flujos de información así como los protocolos de cifrado. El cifrado de la información durante el establecimiento inicial de las comunicaciones, utilizando HTTPS (SSL/TLS) en el servidor de señales, DTLS para los flujos de datos y SRTP para los multimedia en las comunicaciones punto a punto.

DTLS (Datagram Transport Layer Security) es un protocolo que proporciona privacidad en las comunicaciones para protocolos de datagramas. Este protocolo permite a las aplicaciones cliente/servidor comunicarse de manera que se eviten las escuchas no deseadas (eavesdropping), accesos no permitidos, o modificación de mensajes. El protocolo DTLS está basado en el protocolo TLS y proporciona garantías de seguridad equivalentes. La semántica de los datagramas de los protocolos subyacentes no es modificada al utilizar DTLS. Está definido en el RFC 6347.

SRTP. (Secure Real-time Transport Protocol) define un perfil de RTP (Real-time Transport Protocol), con la intención de proporcionar cifrado, autenticación del mensaje e integridad, y protección contra reenvíos a los datos RTP en aplicaciones unicast y multicast. Fue desarrollado por un pequeño grupo del protocolo IP y expertos criptográficos de Cisco y Ericsson. Fue publicado por primera vez por el IETF en marzo de 2004 como el RFC 3711.

3.7.5 Comparativa RTSP - WebRTC.

Durante la elaboración de este proyecto, junto con el equipo de CryptULL (José Iván Santos, Alexandra Rivero, Jezabel Miriam Molina y Pino Caballero), se realizó un estudio sobre el comportamiento de los tiempos de conexión de los protocolos RTSP y WebRTC utilizando para ello la aplicación desarrollada en Android para este proyecto, y otra desarrollada que utiliza RTSP para las conexiones punto a punto.

El estudio ha confirmado el excelente rendimiento que ofrece WebRTC, frente al protocolo RTSP, prácticamente el utilizado en todos los sistemas de videovigilancia actuales. Con unos tiempos que llegan a reducir puntualmente hasta en un segundo, y en un promedio de medio segundo el tiempo de establecimiento de las conexiones multimedia y también reduciendo el retardo que se produce en el video.

Queda así puesto de manifiesto el acierto en la elección de la tecnología WebRTC.

El estudio completo está incluido en el Anexo E y fue presentado como trabajo en UCAmI 2016 (10th International Conference on Ubiquitous Computing and Ambient Intelligence) que se celebrará desde el 29 de noviembre al 2 de diciembre de 2016.

4.Diseño.

Este capítulo se dedicará a la descripción del sistema desarrollado. En él se especifican el funcionamiento, la arquitectura, las herramientas y las técnicas concretas empleadas en la construcción del sistema de videovigilancia.

4.1 Descripción del sistema

A continuación se muestra el esquema general del sistema:

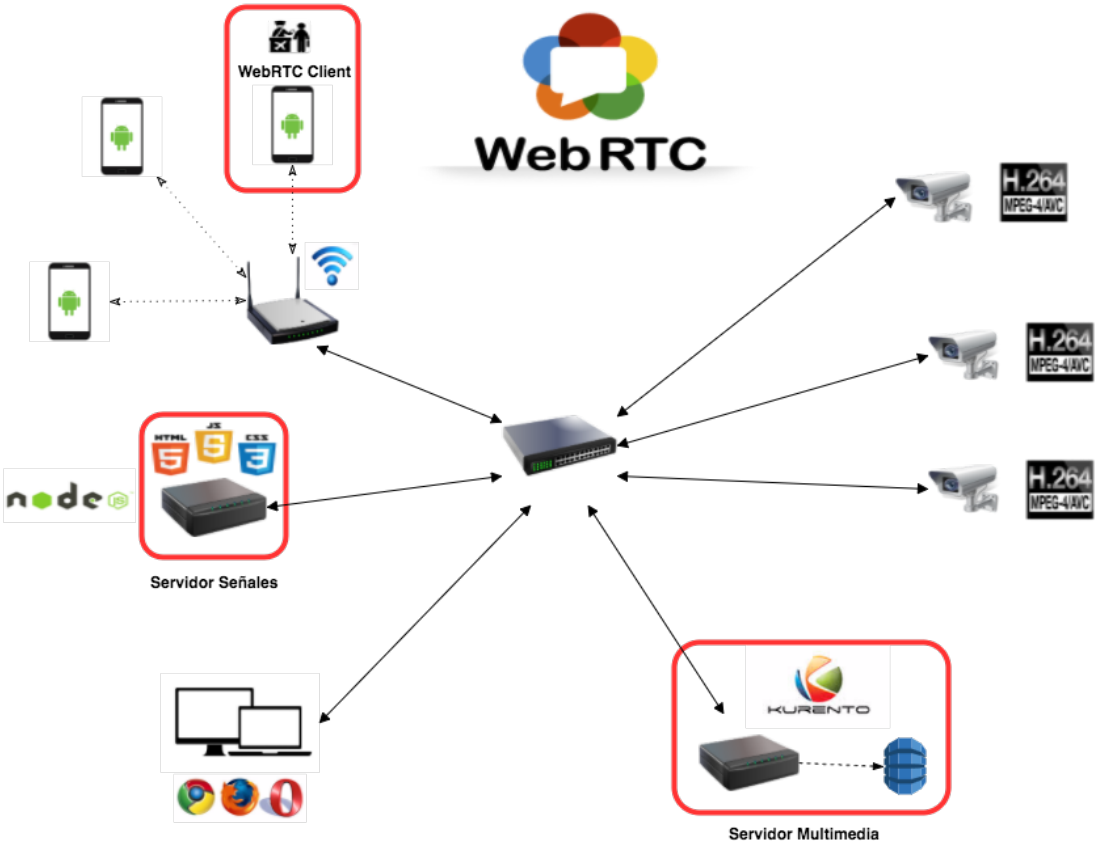

Figura 4.1. Esquema general de proyecto.

En este esquema están destacados, con rectángulos rojos, los componentes principales desarrollados para el proyecto:

- **Aplicación Android.** Se dispondrá de diversos dispositivos móviles en los que se instalará la aplicación desarrollada y que portarán los vigilantes de seguridad. Las funcionalidades son:
  - Visualizar las cámaras fijas del recinto.
  - Compartir con otros vigilantes el video/audio capturado en ese instante.

  - Realizar llamadas de voz a otro/s compañero/s.
  - Grabar el video/audio que capta su dispositivo remotamente, al tiempo que se realiza una detección de rostros sobre el video recibido mediante Realidad Aumentada.
- **Servidor Señales**. Tendrá el papel de central telefónica, se encargará del establecimiento inicial de las llamadas entre los dispositivos, ya sea de los que utilizan la aplicación nativa en Android, como cualquier otro dispositivo: Tablet, PC, Portátil, que utilice Chrome, Firefox, Opera o un navegador compatible con WebRTC. El servidor, adecuará su funcionamiento según el tipo de llamada: grabación de video, videoconferencia, llamada de voz, visualización Cámara-IP, Análisis de Video con Realidad Aumentada.
- **Servidor Multimedia Kurento.** Se encargará de la grabación de video, de la recodificación de las señales de las cámaras a WebRTC y por último, ofrecer una prueba de Realidad Aumentada, detectando y marcando los rostros detectados en los vídeos recibidos.

El sistema se integrará a una infraestructura de red, que se supone ya instalada en el recinto aeroportuario y que cubre de manera inalámbrica todo el recinto. También dispondrá diferentes cámaras fijas IP, situadas en lugares estratégicos y a cuyas imágenes podrán acceder los vigilantes ya sea desde la aplicación móvil, o desde la página web desarrollada para tal fin. En este caso, es necesario disponer de un navegador compatible con WebRTC.

A continuación se describirán detalladamente cada uno de los componentes mencionados.

4.2. Aplicación móvil.

En este apartado, se describirá el funcionamiento de la app **"WebRTC Client"**, así como los detalles de implementación.

Con esta aplicación se pretende que el personal de vigilancia, tenga una herramienta sencilla de utilizar y que realice de manera eficaz las siguientes tareas:

1.- **Conectar con las cámaras fijas del recinto**. Es decir, el vigilante tendrá la posibilidad de seleccionar una o varias cámaras y visualizar lo que ocurre en ese instante en las zonas donde están instaladas.

2.- **Compartir video/audio.** Tendrá la posibilidad de compartir con otro/s compañero/s y/o con la Central las imágenes/audio del dispositivo móvil y así hacerles partícipes de lo que está ocurriendo en ese momento en su ubicación.

3.- **Llamadas de voz.** Es un caso particular del anterior, en el que podrá compartir sólo el audio, y realizar así, llamadas VoIP, utilizando la infraestructura de red y asegurando que van cifradas de extremo a extremo. Este apartado está diferenciado, ya que no compartir el video, reduce significativamente el ancho de banda consumido por la llamada, optimizando así el uso que se realiza de la red.

4.- **Grabación remota y Realidad Aumentada.** Esto permitirá al vigilante almacenar remotamente audio/video de su dispositivo móvil, ante cualquier situación de alarma que así lo requiera. El contenido se almacenará en tiempo real en el servidor multimedia, localizado remotamente, lo que asegura que en caso de avería del terminal, las imágenes y sonido queden almacenados en lugar seguro. Adicionalmente, se ha añadido una funcionalidad extra en este apartado, y es la utilización de Realidad Aumentada. El vigilante no sólo almacenará en el servidor el contenido multimedia que envía el dispositivo, sino que se realizará un análisis de detección facial, marcando todos los rostros que detecta en las imágenes y visualizándose en la pantalla del móvil. Esto podría utilizarse, para la identificación de terroristas o personas "fichadas" en las bases de datos de los Cuerpos de Seguridad del Estado, avisando en el caso de producirse una identificación positiva. Esto último se propone como mejora del proyecto actual.

La implementación que se ha realizado se ha llevado a cabo para dispositivos Android, pero teniendo en consideración una posible adaptación para las gafas y relojes inteligentes basados en la plataforma Android. El alto costo de los dispositivos, así como que muchos de ellos son aún prototipos, ha impedido que pudiera desarrollarse una solución específica para esos dispositivos.

A pesar de ello, la adaptación no supondría un gran esfuerzo, ya que todos los dispositivos comparten la misma plataforma de desarrollo

4.2.1. Implementación.

La aplicación, ha sido desarrollada en el IDE oficial para Android, Android Studio v2.1.1. Puede ser utilizada en plataformas Android 4.3 (Jelly Bean) y superiores, lo que supone un 95% de los dispositivos Android existentes.

Librerías utilizadas:

- **Libjingle**. Librería de Google, que dará soporte en la aplicación a WebRTC. Está escrita en C++, y forma parte del proyecto Chromium, por lo que para utilizarla en Android, hay que compilarla y generar el JNI que permite a Java reutilizar la implementación en C++. Se utiliza la versión precompilada de *pristine*. (Libjingle Peerconnection 11139)
- **Socket.io**. Librería para la utilización de WebSocket, permitirá la comunicación con el servidor de señales.

Permisos necesarios para su correcto funcionamiento:

- **CAMERA**. Hacer fotos / grabar vídeos. Permite acceso al control de la cámara y a la toma de imágenes y vídeos.
- **RECORD_AUDIO**. Grabar audio. Permite acceso grabar sonido desde el micrófono del dispositivo.
- **MODIFY_AUDIO_SETTINGS**. Cambiar ajustes de audio. Permite cambiar ajustes globales de audio, como el volumen.
- **INTERNET.** Acceso a Internet sin límites. Permite establecer conexiones a través de Internet.
- **ACCESS_NETWORK_STATE**. Ver estado de red. Información sobre todas las redes.

4.2.2. Interfaz de usuario.

En este apartado, se describirán cada una de las interfaces desarrolladas para la aplicación así como las diferentes acciones que el usuario podrá realizar.

Pantalla Inicial.
Esta es la pantalla que se inicia cuando se ejecuta la aplicación.

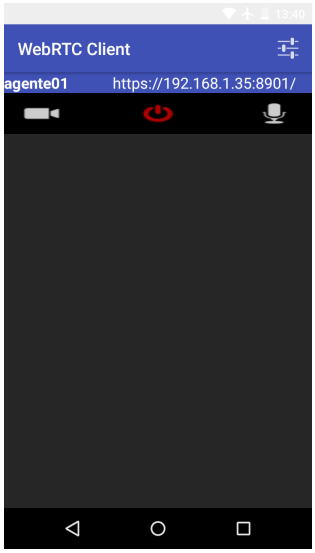

Figura 4.2. Pantalla inicial.

Las acciones que se pueden realizar desde esta pantalla son las siguientes:

Botón	Acción
	Accede al menú de ajustes.
	Activa / Desactiva el video local.
	Conecta / Desconecta el dispositivo del Servidor.
	Activa / Desactiva el audio. Si se desactiva no permitirá realizar llamadas de audio, ocultando el botón asociado.

Tabla 4.1. Acciones pantalla inicial.

Una vez que se conecta al servidor, se deshabilitan los botones de activar/desactivar video y audio. Se previsualizará la cámara del dispositivo en la parte superior derecha en un pequeño rectángulo y también aparecerán tres nuevos botones que nos permiten realizar las acciones que se indican en la tabla 4.2.

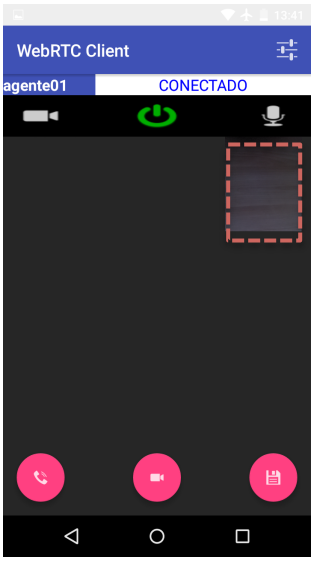

Figura 4.3. Pantalla principal.

Botón	Acción
	Muestra una pantalla para seleccionar el/los destinatario/s y luego establece una llamada sólo de audio.
	Muestra una pantalla para seleccionar el/los destinatario/s y luego establece una llamada video+audio
	Pide confirmación y envía el contenido multimedia al servidor para almacenarlo. Visualizaremos la cámara del dispositivo, realizando una detección de los rostros que aparecen.
	Botón para colgar la llamada. Este botón sustituirá a los anteriores cuando existe alguna conexión en curso.

Tabla 4.2. Acciones pantalla principal.

Si se recibe una llamada, sonará el dispositivo con una melodía y se establecerá la conexión, mostrando la imagen del emisor en el centro de la pantalla, así como su identificador en la barra superior, con fondo blanco y letras azules, bajo el menú ajustes.

Barra de información.

En la parte superior de la aplicación, bajo el nombre de la aplicación, aparece una pequeña barra de información.

En la parte izquierda en fondo azul y letras blancas, se mostrará el identificador del dispositivo.

En la parte derecha en fondo blanco y letras azules, se irán mostrando los diferentes estados durante una conexión (conectado, conectando, completado, desconectado, cerrado)

Una vez establecida correctamente la conexión, mostrará el identificador del destinatario.

Si existen varias conexiones, aparecerá un símbolo **'+'** precediendo al identificador. En este caso, si se pulsa, cambiará de destinatario, pasando a visualizar y escuchar esa conexión.

Figura 4.4. Barra de información.

Pantalla de selección de destinatarios.

Cuando se pulsan los botones de llamada, sea de audio, o de video, se mostrará una pantalla diferente, desde la que se puede seleccionar uno o varios destinatarios, con sólo pulsar sobre los mismos.

Aparecerá un "✔" indicando que está seleccionado.

Una vez seleccionados los destinatarios hay que pulsar el botón "Conectar" para que se establezca las conexión.

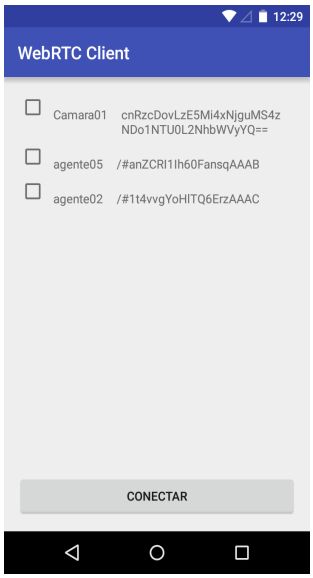

Figura 4.5. Pantalla selección de destinatarios.

Pantalla ajustes.

Esta pantalla aparece cuando se pulsa el botón de ajustes de la aplicación y permite establecer los siguientes parámetros de la aplicación:

- Identificador. Este define el identificador único del dispositivo y con el que se registrará en el servidor.
- La dirección IP del servidor de señales.
- El puerto de conexión con el servidor.
- Activar / Desactivar (SSL/TLS). Por defecto estará activo.
- Cámara. Permite elegir la cámara, sea la frontal o trasera del dispositivo.

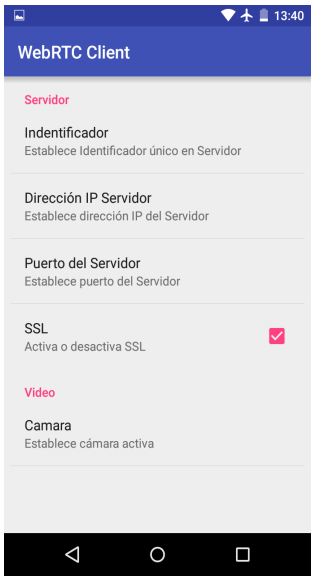

Figura 4.6. Pantalla de ajustes.

4.2.3. Estructura.

En este apartado, se mostrará la estructura general de directorios y ficheros del proyecto en Android y se comentarán los más relevantes.

4.2.3.1. Java.

En esta carpeta se encuentran todas las clases principales Java. A continuación se muestra un cuadro, resumiendo las clases que se han desarrollado en este proyecto y una pequeña descripción de las mismas.

Clase	Descripción
ClientListActivity FeedLiostRowHolder MyReciclerAdapter ServerClient	Se encargan de gestionar y visualizar los clientes conectados. Se utilizan en el momento en el que se pulsa el botón de llamada. El listado se obtiene desde el servidor de señales mediante JSON.
SettingsActivity SettingsFragment	Se encargan de gestionar y visualizar el menú de ajustes de la aplicación.
WebRtcActivity	Es la “actividad” principal, la que se ejecuta cuando

	iniciamos la aplicación.
WebRtcClient PeerConnectionParameters	Estas son los que se encargan de todo el proceso de conexión p2p mediante WebRTC. Se apoya de la librería libjingle.
ApplicationTest ApplicationTest1	Clases de depuración utilizadas para la verificación de la aplicación, como para la obtención de los tiempos de establecimiento de la conexión en las llamadas de WebRTC.

Tabla 4.3. Clases proyecto Android.

4.2.3.2. Recursos.

Los recursos se almacenan en una carpeta denominada "**res**" en la que se encuentran los recursos: layouts o pantallas, imágenes, iconos, sonidos, colores, cadenas, etc. El siguiente cuadro muestra los ficheros más interesantes y una pequeña descripción de los mismos.

Directorio/Fichero	**Descripción**
res/layout	Directorio que contiene las plantillas de las "pantallas" de la aplicación en formato xml
res/layout/activity_client_list.xml	Plantilla para la visualización de los clientes conectados.
res/layout/activity_webrtc.xml	Plantilla de la pantalla principal de la aplicación. Es la que aparece desde iniciamos la app.
res/layout/list_row.xml	Plantilla para el detalle de los datos de los clientes conectados.
menu/menú_webrtc.xml	Configuración del menú de ajustes.
raw/call.ogg	Sonido durante las llamadas entrantes y salientes
raw/server.crt	Certificado del servidor de señales, para realizar las conexiones seguras (SSL/TLS)
values/strings.xml	Cadenas de texto mostradas en la app.

Tabla 4.4. Recursos proyecto Android.

4.2.4. Código Fuente.

En el apéndice B se muestran algunos fragmentos destacables, como son:

- B.1. Build.gradle. Fichero de configuración y dependencias del proyecto Android.
- B.2. Utilización HTTPS. Uso del protocolo https para consultar la lista de clientes conectados.
- B.3. Init conexión y parámetros WebRTC. Inicialización de Socket y valores por defecto de códecs y fps
- B.4. Gestión multiconexión. Activación y visualización sólo del contenido multimedia del cliente seleccionado.

4.2.5. Tests.

Para la realización de pruebas de funcionamiento de interfaz de usuario, así como la obtención de tiempos de establecimiento de conexiones multimedia, necesarios para el estudio de WebRTC, se utiliza el framework Espresso. Este framework, facilita la interacción y depuración de la aplicación, simulando las pulsaciones reales que puede realizar un usuario.

En el ejemplo de código, mostrado en el Apéndice B, Figura B.5, se simula cincuenta conexiones/desconexiones de un vigilante a otro.

4.3. Servidor de señales.

En este apartado, se describe el funcionamiento del servidor de señales encargado del establecimiento inicial de las llamadas entre los dispositivos móviles.

Para que los clientes se registren, se comuniquen entre ellos, intercambien tanto información multimedia (SDP) como su configuración de red (ICE), es necesario hacer uso de un servidor de señalización que se encargue de manejar e intercambiar estos datos, así como los diferentes mensajes de control, tal como se comentó cuando se describió el funcionamiento de WebRTC.

Una vez producido este intercambio, los clientes se conectarán directamente, punto a punto, sin ninguna intervención del servidor.

La forma más sencilla y directa para implementar este intercambio de mensajes es utilizar JSON sobre Sockets y es lo que se ha utilizado, con la ayuda de la librería Socket.io

Mensajes de control.

Para el correcto funcionamiento del sistema, los clientes al solicitar conexión utilizan estos mensajes, que el servidor de señales interpreta para realizar la acción necesaria. Los mensajes de control propios de nuestra aplicación y no del protocolo WebRTC son los siguientes:

Mensaje	**Descripción**
callvideo_clienteA	Si el destinatario es una cámara IP, inicializa el servidor multimedia y realiza las llamadas API necesarias para obtener del servidor un cliente WebRTC con origen la cámara y al que le enviará el resto de comunicaciones, transformando el mensaje al formato propio de Kurento. Si no es cámara, entonces se reenvía el mensaje al destinatario sin modificar. El nombre después del "_" identifica al remitente.
callaudio_clienteA	Como el anterior, sólo que el receptor tiene en cuenta que es una

	llamada de voz y desactiva el video en la emisión multimedia.
Kurento	El destinatario es el servidor multimedia, por lo que inicializa el servidor multimedia, activa los módulos de grabación y detección facial. Crea un cliente WebRTC con origen la detección facial en el servidor multimedia para toda comunicación posterior, transformando el mensaje al formato propio de Kurento.

Tabla 4.5. Mensaje de control.

4.3.1. Implementación.

La aplicación, ha sido desarrollada en JavaScript, bajo un servidor NodeJs ejecutado sobre OsX El Capitan 10.11.4, aunque al ser multiplataforma, podría realizarse sobre otro sistema operativo.

Librerías principales utilizadas:

- **Socket.io**. Librería para la utilización de WebSocket, permitirá la comunicación con los clientes.
- **Express.** Framework para Node.js que permite desplegar el sitio web para utilizar WebSocket
- **Kurento-Client.** Librería para la comunicación con el servidor multimedia Kurento.

4.3.2. Estructura.

A continuación se muestra el árbol con la estructura de los ficheros de la aplicación:

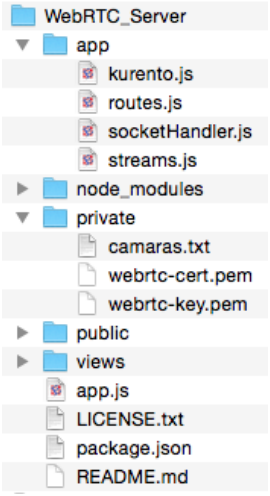

Figura 4.7. Estructura de directorios y ficheros.

La siguiente tabla incluye un breve resumen de los directorios y ficheros más relevantes.

Directorio/Fichero	Descripción
App	Directorio que contiene los fuentes de las “clases” principales

app/kurento.js	Fichero para la Comunicación con servidor multimedia.
app/routes.js	Fichero para la Consola Web de clientes.
app/socketHandler.js	Fichero del Manejador de "Mensajes".
app/streams.js	Fichero para el registro de clientes conectados.
node_modules	Directorio con las dependencias de la aplicación.
Private	Directorio sólo accesible por servidor.
private/cameras.txt	Fichero con las IPs de las cámaras fijas.
private/webrtc-cert.pem private/webrtc-key.pem	Certificados del servidor para las conexiones SSL/TLS
Public	Directorio público con las imágenes, javascripts ,favicons, HTML, etc
Views	Directorio con las vistas de AngularJS.
app.js	Aplicación principal.
package.json	Fichero con las dependencias e información para npm.

Tabla 4.6. Descripción Ficheros y directorios.

4.3.3. Instalación.

Para la instalación del servidor, se utiliza npm, manejador de dependencias de Node.js, que automatiza el proceso.

Desde una shell, dentro del directorio de los fuentes del servidor, se ejecuta:

"npm install"

4.3.4. Ejecución.

Para la ejecución de la aplicación en el servidor, se utilizará una "Shell", dentro del directorio con los ficheros "fuentes" y se escribe la siguiente instrucción:

"node app.js"

Aparecerá la dirección IP y puerto en la que el servidor web está activado, así como las cámaras fijas registradas en el sistema.

Express https server listening on 192.168.1.37:8901

Fichero camaras encontrado, leyendo camaras:

Camara 1 : rtsp://192.168.1.33:5554/camera

4.3.5. Depuración.

Por defecto se muestra por consola las conexiones entre clientes, así como los mensajes entre ellos. Lo que permite ir observando el comportamiento del servidor.

Se registra una nueva cámara fija en el sistema:

Camara 1 : rtsp://192.168.1.33:5554/camera

Se registra un nuevo cliente:

-- /#fh7uHG10HjxXzuz_AAAA joined --
-- /#fh7uHG10HjxXzuz_AAAA is ready to stream --

Se solicita la lista de clientes conectados:

GET /streams.json 200 3.760 ms – 173

El agente05 realiza una llamada a otro dispositivo, se realiza tanto el intercambio de información multimedia (init, offer, answer) y la de red (candidate):

```
message from:/#SJFliEGr8TEEG0i2AAAB to: /#fh7uHG10HjxXzuz_AAAA callvideo_agente05
message from:/#SJFliEGr8TEEG0i2AAAB to: /#fh7uHG10HjxXzuz_AAAA init
message from:/#fh7uHG10HjxXzuz_AAAA to: /#SJFliEGr8TEEG0i2AAAB offer
message from:/#fh7uHG10HjxXzuz_AAAA to: /#SJFliEGr8TEEG0i2AAAB candidate
message from:/#fh7uHG10HjxXzuz_AAAA to: /#SJFliEGr8TEEG0i2AAAB candidate
message from:/#fh7uHG10HjxXzuz_AAAA to: /#SJFliEGr8TEEG0i2AAAB candidate
message from:/#fh7uHG10HjxXzuz_AAAA to: /#SJFliEGr8TEEG0i2AAAB candidate
message from:/#fh7uHG10HjxXzuz_AAAA to: /#SJFliEGr8TEEG0i2AAAB candidate
message from:/#fh7uHG10HjxXzuz_AAAA to: /#SJFliEGr8TEEG0i2AAAB candidate
message from:/#fh7uHG10HjxXzuz_AAAA to: /#SJFliEGr8TEEG0i2AAAB candidate
message from:/#fh7uHG10HjxXzuz_AAAA to: /#SJFliEGr8TEEG0i2AAAB candidate
message from:/#SJFliEGr8TEEG0i2AAAB to: /#fh7uHG10HjxXzuz_AAAA answer
message from:/#SJFliEGr8TEEG0i2AAAB to: /#fh7uHG10HjxXzuz_AAAA candidate
message from:/#SJFliEGr8TEEG0i2AAAB to: /#fh7uHG10HjxXzuz_AAAA candidate
```

4.3.6. Código Fuente.

Las partes mas relevantes, se han incluido en el Apéndice B. Entre ellas se mencionan:

- B.6. Activación del servidor web HTTPS en Express.
- B.7. Lectura y registro de cámaras IP.
- B.8. Activación del servidor multimedia Kurento.
- B.9. Inicialización servicio Kurento y transformación de respuesta JSON.
- B.10. Módulos y dependencias de la aplicación servidor.

4.3.7.Consola Web.

El servidor tiene activa una página web que podemos utilizar con cualquier navegador compatible con WebRTC y que nos mostrará la lista de clientes conectados y cámaras fijas IP, con la posibilidad de establecer conexión con ellos. Esta web, facilitaría la visión simultanea de todas las cámaras fijas IP, y la conexión puntual con los vigilantes desde un ordenador con multipantalla desde una central de seguridad.

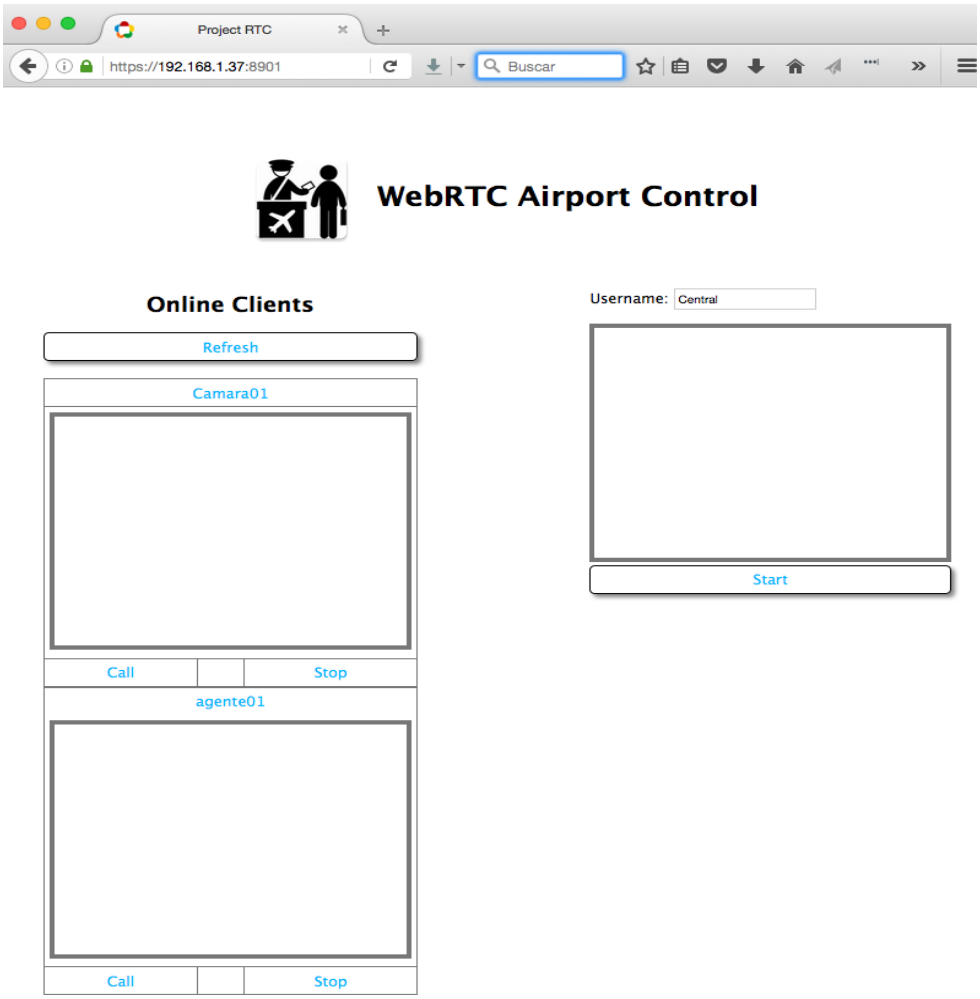

Figura 4.8. Web servidor.

Para establecer conexión con el cliente, se pulsa "**Call**" en dicho cliente, y "**Stop**" para colgar la llamada. Se puede conectar con varios clientes simultáneamente.

El botón "Refresh" actualizará los clientes conectados, cerrando las conexiones abiertas.

4.4. Servidor multimedia.

La última parte del sistema será un servidor multimedia que complementa la parte de comunicación entre dispositivos y ofrece funciones de valor añadido como son:

- Almacenamiento.
- Recodificación.
- Análisis inteligente de imágenes.

Este servidor aunque pudiera implantarse en el mismo servidor que el de señales, se ha mantenido independiente, este diseño aporta más ventajas, tanto en seguridad como en flexibilidad.

El siguiente cuadro resume las ventajas que aporta esta propuesta de diseño.

Ventaja	Descripción
Independencia.	Separar los servidores, permite que en caso de fallo/mantenimiento del servidor de comunicaciones, este siga trabajando de manera independiente.
Seguridad	Permite aplicar restricciones adicionales de acceso al servidor: puertos, MAC, etc.
Seguridad Cámaras IP.	Se puede restringir el acceso a las cámaras fijas IP sólo al servidor multimedia.
Flexibilidad	Permite incorporar cámaras de distintos fabricantes y con protocolos diferentes sin alterar el sistema. El servidor multimedia se encargará de la recodificación WebRTC.
Seguridad en Almacenamiento	Permite almacenar el contenido en ubicaciones con acceso restringido.
Almacenamiento remoto	Almacenar remotamente, asegura que se disponga del video, aunque el dispositivo móvil sufra algún accidente.
Modularidad	Se pueden añadir nuevas funcionalidades en el propio servidor sin afectar a los clientes móviles y de una manera sencilla. Ej. Detección de matrículas.
Abierto	Es OpenSource, lo que permite ajustar y e implementar mejoras.

Tabla 4.7. Ventajas servidor multimedia.

En resumen, las tareas que realizará el servidor Kurento serán:

- Almacenar el contenido multimedia de los vigilantes que lo soliciten en el servidor.
- Conectar a cámaras IP fijas con protocolo RTSP/H.264, realizando una recodificación de la señal.
- Realidad Aumentada, para realizar un análisis del vídeo entregado, realizando una detección de rostros.

4.4.1. Instalación.

La instalación se ha realizado en un servidor Ubuntu 14.04 LTS 64 bits, utilizando para ello una máquina virtual bajo VirtualBox.

```
echo "deb http://ubuntu.kurento.org trusty kms6" | sudo tee /etc/apt/sources.list.d/kurento.list
wget -O - http://ubuntu.kurento.org/kurento.gpg.key | sudo apt-key add -
sudo apt-get update
sudo apt-get install kurento-media-server-6.0
```

4.4.2. Ejecución.

Para inicializar o para los servicios del servidor multimedia, se ejecutan las siguientes instrucciones:

```
sudo service kurento-media-server-6.0 start
sudo service kurento-media-server-6.0 stop
```

4.4.3. Grabación multimedia.

Por defecto los ficheros multimedia que los clientes Android soliciten grabar, se almacenarán en el servidor Kurento.

Tendrán formato "**.webm**", con codec VP8 (video) y OPUS (audio).
se grabarán en la ruta: ***"\tmp\"***
La nomenclatura será **<ID> +< FechaHora>.webm**

Ej: ***"\tmp\agente01_2016-04-27T13:58:30.webm"***

A continuación se muestra el diagrama con la lógica de conexión de los módulos de Kurento utilizados para la grabación.

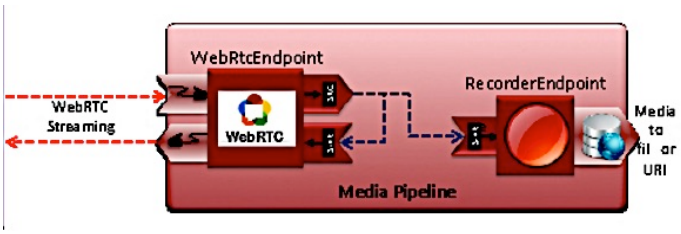

Figura 4.9. Esquema de conexión Grabación.

4.4.4. Conexión cámaras fijas IP.

El sistema permite conectar y visualizar cámaras IP fijas desde la aplicación móvil. Actualmente, estas cámaras para la emisión multimedia, utilizan el protocolo RTSP y códec H.264. Kurento se encargará de "transcodificar" a WebRTC (VP8/Opus), para que la aplicación Android pueda visualizar su contenido.

A continuación se muestra una figura con la lógica de conexión de los módulos utilizados en Kurento.

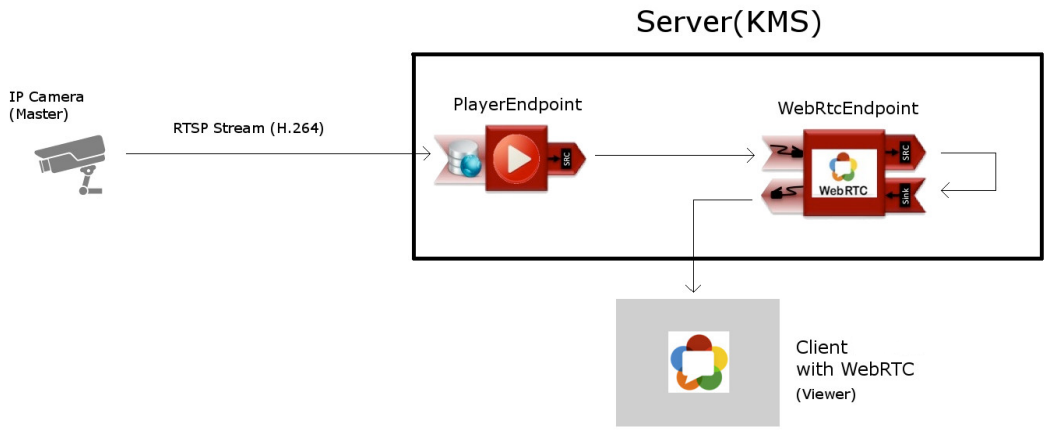

Figura 4.10. Esquema de conexión cámara IP.

4.4.5. Realidad Aumentada. Reconocimiento facial.

En la aplicación móvil, cuando un cliente solicita grabar el contenido multimedia, no sólo se almacena en el servidor, sino que en tiempo real, se procesa el video recibido y es devuelto al cliente modificado, mostrando una pequeña imagen superpuesta en los rostros que detecta.

Para realizar este funcionamiento, se utiliza el módulo **FaceOverlayFilter**.

En el siguiente diagrama muestra la lógica de conexión entre módulos de Kurento.

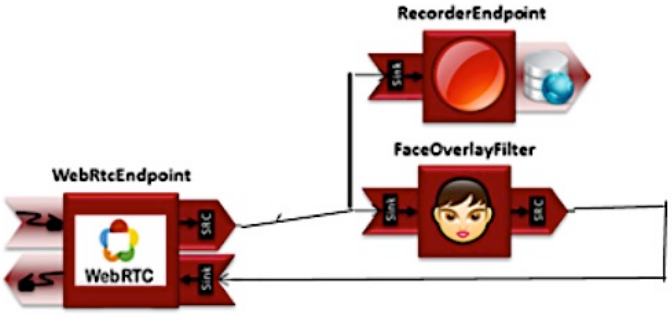

Figura 4.11. Esquema de conexión Realidad Aumentada.

La siguiente ilustración muestra un ejemplo de esta funcionalidad. Desde la aplicación móvil, después de conectar con el servidor, se pulsará el botón de grabar, tras confirmar, se empieza a visualizar por pantalla la propia cámara del dispositivo y se detectan todos los rostros que aparecen en la imagen y se

superpone una imagen predefinida, un icono de peligro, en cada uno de los rostros. En esta versión se limita a reconocer todos los rostros, no realizando distinción ni reconocimiento de personas concretas.

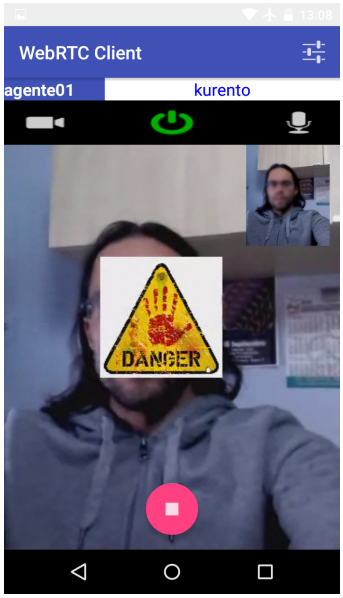

Figura 4.12. Funcionamiento reconocimiento facial.

4.4.6. Comunicación servidor señales y multimedia.

Toda comunicación inicial con el servidor multimedia, se realiza a través del servidor de señales (SIG). Éste se encarga de recibir las peticiones, mensajes, de los clientes (WebRTC_Client App, Navegador) y en el caso de una llamada a un servicio del servidor multimedia, transformará adecuadamente los mensajes, al protocolo propio de Kurento. También se encargará de inicializar correctamente los módulos implicados en el servicio solicitado utilizando para ello su API.

Los pasos que se realizan para la grabación multimedia son:

- Cliente solicita la grabación multimedia.
- Servidor SIG le responde y solicita su información multimedia de intercambio (SDP).
- Cliente envía SDP.
- SIG inicializa la conexión con Kurento (si no estaba ya establecida)
- SIG inicializa el módulo de grabación, el de realidad aumentada y el cliente WebRTC, utilizando la API Kurento.
- SIG Transforma el SDP del cliente en protocolo Kurento y lo envía al servidor multimedia.
- Kurento responde con su SDP.
- SIG transforma SDP y se lo envía al cliente.

- El cliente y Kurento envían información de red para la conexión multimedia. (ICE Candidate)
- SIG intercambia la información recibida.
- Se produce el intercambio multimedia directo Cliente-Kurento.
- Kurento graba el contenido multimedia, el cliente recibe su propio video con la detección de rostros activada.

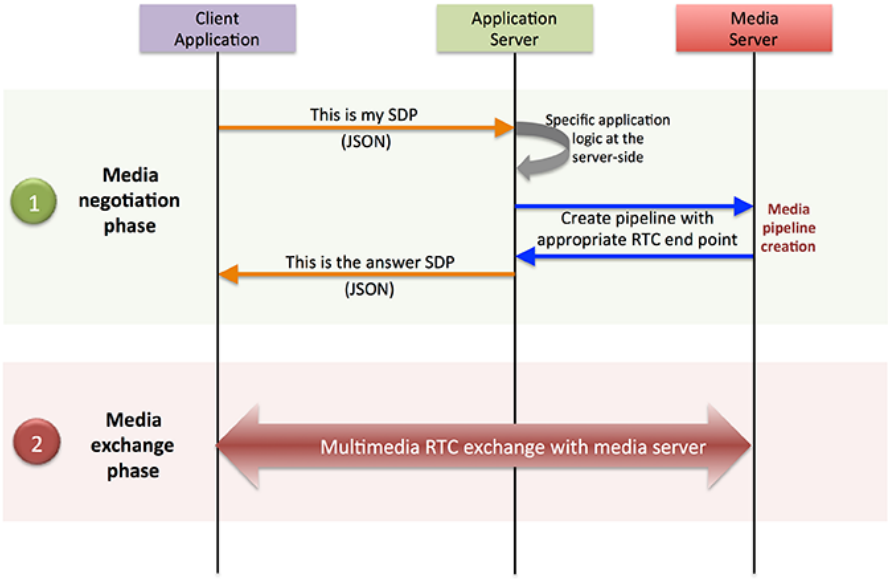

Figura 4.13. Conexión Cliente-Servidor Señales-Kurento

Los pasos que se realizan para la conexión con cámaras IP son:

- Cliente solicita la conexión con cámara IP.
- Servidor SIG inicializa la conexión con Kurento (si no estaba ya establecida).
- SIG inicializa el módulo de reproducción con la cámara IP utilizando protocolo RTSP y el cliente WebRTC.
- Kurento establece conexión con la cámara y responde con su SDP (WebRTC)
- SIG transforma SDP y se lo envía al cliente.
- El cliente y Kurento envían información de red para la conexión multimedia. (ICE Candidate)
- Servidor intercambia la información recibida.
- Se produce el intercambio multimedia directo Cliente-Kurento.
- Cliente recibe contenido de la cámara IP.

5.Conclusiones y trabajos futuros.

5.1. Conclusiones.

En este proyecto se ha desarrollado un sistema de videovigilancia, para el caso particular de los aeropuertos, en el que utilizando tecnologías actuales como WebRTC y el potencial de los dispositivos móviles, se consigue aportar nuevas funcionalidades de las que otros sistemas carecen y manteniendo una excelente relación calidad-precio. La utilización de sistemas de código abierto y tecnologías en auge, nos han garantizado en el proyecto, seguridad, flexibilidad, rendimiento y bajo coste.

Para ello se ha utilizado WebRTC (Web Real-Time Communications), un proyecto mantenido por Google, que permite comunicaciones en tiempo real y que destaca por su alta calidad, y eficiencia . Esta tecnología ofrece multitud de posibilidades y ventajas, tanto en rendimiento como en seguridad.

Además, se ha creado una aplicación móvil que permite la comunicación de los diferentes vigilantes de seguridad y que proporciona las siguientes funcionalidades:

- Visualizar remotamente las cámaras fijas instaladas en el recinto.
- Compartir con otros vigilantes o la central el video y/o audio de su dispositivo.
- Realizar llamadas VoIP a otros vigilantes.
- Almacenar remotamente el video y/o audio de su dispositivo y realizar una prueba de Realidad Aumentada como es la detección de rostros.

Por otra parte, se ha implementado un servidor que se encargará de la gestión de las llamadas entre vigilantes. Y por último, se ha integrado otro servidor que permitirá, almacenar el contenido multimedia, acceder a cámaras fijas IP con protocolos diferentes a WebRTC y realizar un análisis inteligente de los vídeos recibidos, en particular, la detección facial en tiempo real.

Par finalizar es de destacar la incorporación de un módulo de Realidad Aumentada con la detección facial, para así mostrar el potencial que tiene el sistema. Su modularidad, ofrece un alto grado de personalización para un cliente que decidiese añadir un sistema de videovigilancia de estas características, pudiendo incorporar funcionalidades extra que se adapten a sus necesidades reales y a bajo costo.

5.2. Trabajos futuros.

Estas son algunas de las posibles mejoras a realizar sobre el sistema:

- Añadir sistema de localización en interiores. Permitiendo conocer en todo momento la ubicación de los vigilantes y los videos recibidos.
- Adaptar y realizar pruebas sobre gafas y relojes inteligentes.
- Añadir un sistema de alarmas. Cuando las cámaras fijas detecten algún comportamiento anormal, avisen a los vigilantes más cercanos o predefinidos.
- Mejora en el módulo de detección de rostros, añadiendo la posibilidad de captura de imagen, comprobación con una base de datos de las Fuerzas y Cuerpos de Seguridad del Estado (terroristas, delincuentes, personas desaparecidas…) y nos muestre aviso utilizando realidad aumentada.
- Incorporar otros módulos, como la detección de objetos perdidos.

Referencias

[1] [Bosh] Bosh Security Sistems, http://es.boschsecurity.com/es/
[2] [Eneo] Eneo Security, http://eneo-security.com/
[3] [EsGlobal] Indice de Terrorismo, http://www.esglobal.org/indice-de-terrorismo-global-2015/
[4] [Express] Express.js, http://expressjs.com
[5] [Garner] Informe wearable, http://www.gartner.com/newsroom/id/3198018
[6] [Garner] Informe Android, http://www.gartner.com/newsroom/id/3115517
[7] [ICE] ICE, http://io13webrtc.appspot.com/#51
[8] [Kurento] Kurento, https://www.kurento.org/
[9] [Milestone] Milestone, https://www.milestonesys.com/
[10] [Node] Node.js, http://nodejs.org
[11] [Socket.io] Socket.io, http://socket.io/
[12] [WebRTC] WebRTC, https://webrtc.org/, http://io13webrtc.appspot.com/#1
[13] [Wiki-Android] Wikipedia-android, https://es.wikipedia.org/wiki/Android
[14] [Wiki-node] Wikipedia-node,https://es.wikipedia.org/wiki/Node.js
[15] [Wiki-WebRTC] Wikipedia-WebRTC, https://es.wikipedia.org/wiki/WebRTC
[16] [Wiki-javascript] Wikipedia-JavaScript,http://en.wikipedia.org/wiki/JavaScript
[17] [Wiki-json] Wikipedia-JSON, http://es.wikipedia.org/wiki/JSON

Links.

http://www.codeproject.com/Articles/826045/Android-security-Implementation-of-Self-signed-SSL
https://developer.android.com/training/articles/security-ssl.html
http://mami.uclm.es/ucami-2016/cfp.html
https://sites.google.com/site/investigacionwebrtc/introduccion
https://webrtchacks.com/kurento/
http://webrtc-security.github.io/
http://www.vogella.com/tutorials/AndroidTestingEspresso/article.html
http://www.wareable.com/headgear/the-best-smartglasses-google-glass-and-the-rest

Apéndice A. Glosario.

Augmented Reality (AR). La realidad aumentada es el término que se usa para definir una visión a través de un dispositivo tecnológico, directa o indirecta, de un entorno físico del mundo real, cuyos elementos se combinan con elementos virtuales para la creación de una realidad mixta en tiempo real.

DOM (Document Object Model). es esencialmente una interfaz de plataforma que proporciona un conjunto estándar de objetos para representar documentos HTML, XHTML y XML, un modelo estándar sobre cómo pueden combinarse dichos objetos, y una interfaz estándar para acceder a ellos y manipularlos.

DTLS (Datagram Transport Layer Security). Es un protocolo que proporciona privacidad en las comunicaciones para protocolos de datagramas. Este protocolo permite a las aplicaciones cliente/servidor comunicarse de manera que se eviten las escuchas no deseadas (eavesdropping), accesos no permitidos, o modificación de mensajes. El protocolo DTLS está basado en el protocolo TLS y proporciona garantías de seguridad equivalentes. La semántica de los datagramas de los protocolos subyacentes no es modificada al utilizar DTLS.

HTTP (Hypertext Transfer Protocol). Es el protocolo de comunicación que permite las transferencias de información en la World Wide Web (WWW). HTTP fue desarrollado por el World Wide Web Consortium y la Internet Engineering Task Force

HTTPS (Hypertext Transfer Protocol Secure). Es un protocolo de aplicación basado en el protocolo HTTP, destinado a la transferencia segura de datos de Hipertexto, es decir, es la versión segura de HTTP.

ICE (Interactive Connection Establishment). Se puede considerar como una aplicación marco que define cómo emplear los protocolos STUN y TURN para poder atravesar NAT, eligiendo la mejor interconexión posible entre las dos partes que desean comunicarse. ICE define un protocolo de actuación gracias al cual dos dispositivos SIP serán capaces de mantener una sesión multimedia salvando todas las dificultades que el NAT pueda poner de por medio.

IETF (**Internet Engineering Task Force).** Es una organización internacional abierta de normalización, que tiene como objetivos el contribuir a la ingeniería de Internet, actuando en diversas áreas, como transporte, encaminamiento, seguridad. Fue creada en EE. UU. en 1986. El IETF es mundialmente conocido por ser la entidad que regula las propuestas y los estándares de Internet, conocidos como RFC.

JSON (JavaScript Object Notation). Es un formato de texto ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript aunque hoy, debido a su amplia adopción como alternativa a XML, se considera un formato de lenguaje independiente.

NAT Network Address Translation. Es un mecanismo utilizado por routers IP para intercambiar paquetes entre dos redes que asignan mutuamente direcciones incompatibles. Consiste en convertir, en tiempo real, las direcciones utilizadas en los paquetes transportados. Cuando el cliente envía paquetes fuera de la red, NAT traduce la dirección IP interna del cliente a una dirección externa. Para los usuarios externos, todo el tráfico que entra a la red y sale de ella tiene la misma dirección IP o proviene del mismo conjunto de direcciones.

NPM. Es el manejador de paquetes por defecto para Node.js.

P2P (Peer to Peer). Es una red de ordenadores en la que todos o algunos aspectos funcionan sin clientes ni servidores fijos, sino una serie de nodos que se comportan como iguales entre sí. Es decir, actúan simultáneamente como clientes y servidores respecto a los demás nodos de la red. Las redes P2P permiten el intercambio directo de información, en cualquier formato, entre los ordenadores interconectados.

RTP (Real-time Transport Protocol). Es un protocolo de nivel de sesión utilizado para la transmisión de información en tiempo real, como por ejemplo audio y vídeo en una videoconferencia. Está desarrollado por el grupo de trabajo de transporte de Audio y Video del IETF**.**

RTSP (Real Time Streaming Protocol). El protocolo de transmisión en tiempo real, establece y controla uno o muchos flujos sincronizados de datos, ya sean de audio o de video. El RTSP actúa como un mando a distancia mediante la red para servidores multimedia.

SIP (Session Initiation Protocol). Es un protocolo desarrollado por el grupo de trabajo del IETF con la intención de ser el estándar para la iniciación, modificación y finalización de sesiones interactivas de usuario donde intervienen elementos multimedia como el video, voz, mensajería instantánea, juegos en línea y realidad virtual.

Socket. Designa un concepto abstracto por el cual dos programas (posiblemente situados en computadoras distintas) pueden intercambiar cualquier flujo de datos, generalmente de manera fiable y ordenada. Un socket queda definido por un par de direcciones IP local y remota, un protocolo de transporte y un par de números de puerto local y remoto.

SRTP (Secure Real-time Transport Protocol). Define un perfil de RTP (Real-time Transport Protocol), con la intención de proporcionar cifrado, autenticación del mensaje e integridad, y protección contra reenvíos a los datos RTP en aplicaciones unicast y multicast.

STUN (Session Traversal Utilities for NAT). Es un protocolo de red del tipo cliente/servidor que permite a clientes NAT encontrar su dirección IP pública, el tipo de NAT en el que se encuentra y el puerto de Internet asociado con el puerto local a través de NAT. Esta información es usada para configurar una comunicación UDP entre dos hosts que se encuentren tras enrutadores NAT.

TURN (Traversal Using Relay NAT). Es un protocolo de comunicaciones para que un elemento situado tras un NAT o firewall pueda recibir datos del exterior por conexiones TCP o UDP. Es una extensión del protocolo STUN-bis para conseguir atravesar NAT cuando ambos extremos de la comunicación están detrás de NAT simétricos. Con TURN, el tráfico de contenido de la sesión pasará por un servidor que hace de repetidor.

VoIP (voice over IP). Es un conjunto de recursos que hacen posible que la señal de voz viaje a través de Internet empleando el protocolo IP (Protocolo de Internet). Esto significa que se envía la señal de voz en forma digital, en paquetes de datos, en lugar de enviarla en forma analógica a través de circuitos utilizables sólo por telefonía convencional

W3C (World Wide Web Consortium). Es una comunidad internacional que desarrolla estándares que aseguran el crecimiento de la Web a largo plazo.

WebSocket. Es una tecnología que proporciona un canal de comunicación bidireccional y full-duplex sobre un único socket TCP. Está diseñada para ser implementada en navegadores y servidores web, pero puede utilizarse por cualquier aplicación cliente/servidor.

Wearables. Es aquel dispositivo que se lleva sobre, debajo o incluido en la ropa y que está siempre encendido, no necesita encenderse y apagarse. En la actualidad los dispositivos más importantes dentro de este sector según su categoría son relojes inteligentes o smartwatches, pulseras de actividad, gafas inteligentes o smartglasses y ropa inteligente entre otros.

Apéndice B. Código Fuente Destacable.

B.1. Build.gradle. Android.

Fichero de configuración y dependencias del proyecto Android.

```
android {
    compileSdkVersion 23
    buildToolsVersion "23.0.2"

    defaultConfig {
        applicationId "com.bicacaro.android.webrtc_client"
        minSdkVersion 18
        targetSdkVersion 23
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:23.2.0'
    compile 'com.android.support:design:23.2.0'
    //compile 'io.pristine:libjingle:9694@aar'
    compile 'io.pristine:libjingle:11139@aar'
    compile 'com.google.code.gson:gson:2.2.4'
    compile ('io.socket:socket.io-client:0.7.0') {
    // excluding org.json which is provided by Android
    exclude group: 'org.json', module: 'json'
    // TESTING DEPENDENCIES
    androidTestCompile 'com.android.support:support-annotations:23.2.0'
    androidTestCompile 'com.android.support.test:runner:0.4.1'
    androidTestCompile 'com.android.support.test:rules:0.4.1'
    // Optional -- Hamcrest library
    androidTestCompile 'org.hamcrest:hamcrest-library:1.3'
    // Optional -- UI testing with Espresso
    androidTestCompile 'com.android.support.test.espresso:espresso-core:2.2.1'
    // Optional -- UI testing with UI Automator
    androidTestCompile 'com.android.support.test.uiautomator:uiautomator-v18:2.1.1'

    androidTestCompile('com.android.support.test.espresso:espresso-contrib:2.2') {
        // Necessary to avoid version conflicts
        exclude group: 'com.android.support', module: 'appcompat'
        exclude group: 'com.android.support', module: 'support-v4'
        exclude group: 'com.android.support', module: 'support-annotations'
        exclude module: 'recyclerview-v7'
    }
    }
}
```

Figura B.1. Fichero Buil.gradle proyecto Android.

B.2. Utilización HTTPS. Android.

Ejemplo de uso de https para la obtención del listado de clientes conectados del servidor de señales, en la clase ClientListActivity.

```
protected Integer doInBackground(String... params) {
    Integer result = 0;
    HttpsURLConnection urlConnectionSec;
    HttpURLConnection urlConnection;

    try {

        //Utilizamos certificado servidor para SSL
        InputStream ssl = getResources().openRawResource(R.raw.server);

        CertificateFactory certificateFactory = CertificateFactory.getInstance("X.509", "BC");
        X509Certificate cert = (X509Certificate) certificateFactory.generateCertificate(ssl);
        String alias = "alias";  //cert.getSubjectX500Principal().getName();

        KeyStore trustStore = KeyStore.getInstance(KeyStore.getDefaultType());
        trustStore.load(null);
        trustStore.setCertificateEntry(alias, cert);
        KeyManagerFactory kmf = KeyManagerFactory.getInstance("X509");
        kmf.init(trustStore, null);
        KeyManager[] keyManagers = kmf.getKeyManagers();

        TrustManagerFactory tmf = TrustManagerFactory.getInstance("X509");
        tmf.init(trustStore);
        TrustManager[] trustManagers = tmf.getTrustManagers();

        SSLContext sslContext = SSLContext.getInstance("TLS");
        sslContext.init(keyManagers, trustManagers, null);

            /* forming th java.net.URL object */
        URL url = new URL(params[0]);

        if (url.getProtocol().equals("https")) {
            urlConnectionSec = (HttpsURLConnection) url.openConnection();
            urlConnectionSec.setSSLSocketFactory(sslContext.getSocketFactory());
            urlConnectionSec.setRequestMethod("GET");

            int statusCode = urlConnectionSec.getResponseCode();

            /* 200 represents HTTP OK */
            if (statusCode ==  200) {

                BufferedReader r = new BufferedReader(new InputStreamReader(urlConnectionSec.getInputStream()));
                StringBuilder response = new StringBuilder();
                String line;
                while ((line = r.readLine()) != null) {
                    response.append(line);
                }
                parseResult(response.toString());
                result = 1; // Successful
            }
            else{
                result = 0; //"Failed to fetch data!";
            }
```

Figura B.2 Conexión HTTPS en ClientListActivity.

B.3. Init conexión y parámetros WebRTC. Android.

Inicialización de Socket y valores por defecto de códecs y fps (VP9, Opus, 30) en la clase principal WebRtcActivity.

```
private void inicializar()  {

    init_server_info();
    Point displaySize = new Point();
    getWindowManager().getDefaultDisplay().getSize(displaySize);
    PeerConnectionParameters params = new PeerConnectionParameters(
            sharevideo, false, displaySize.x, displaySize.y, VIDEO_FPS, 1, VIDEO_CODEC_VP9, true, 1, AUDIO_CODEC_OPUS, true);
    InputStream ssl = getResources().openRawResource(R.raw.server);
    boolean frcamera = (cameraId == 0);

    String au = "true",vid = "true";
    if (!shareaudio)
        au = "false";
    if (!sharevideo)
        vid = "false";

    try {
        //client = new WebRtcClient(this, mSocketAddress, params, VideoRendererGui.getEGLContext(),ssl,frcamera,vid,au);}
        client = new WebRtcClient(this, mSocketAddress, params, VideoRendererGui.getEglBaseContext(),ssl,frcamera,vid,au);}
    catch (EngineIOException e) {
        e.printStackTrace();
        }

    //activar_botones_condiciones(false);
    mostrar_botones_llamada(true);
    mOcultarGL = false;
    OcultarVistas();

}
```

Figura B.3.Conexión inicial WebRTC.

B.4. Gestión multiconexión. Android.

En multiconexión activamos sólo el cliente que está en pantalla. WebrtcActivity.

```
//Si disponemos de varios videos remotos, solo activamos uno
private void remoteRenderSetVisible(String name) {

    String n;
    Iterator<String> it = remoteMedias.keySet().iterator();
    MediaStream remoteStream;

    while (it.hasNext()) {
        n = it.next();
        remoteStream = remoteMedias.get(n);
        if (remoteStream.videoTracks.size() > 0 )
           remoteStream.videoTracks.get(0).setEnabled(false);//  removeRenderer(new VideoR
        if (remoteStream.audioTracks.size() > 0 )
            remoteStream.audioTracks.get(0).setEnabled(false);//  removeRenderer(new Video
    }
    if (remoteMedias.containsKey(name)) {
        destName = name;
        if (client.getPeersConnected() > 1)
            destName = "+ " + name;
        setStatusText(destName);
        remoteStream = remoteMedias.get(name);
        if (remoteStream.videoTracks.size() > 0) {
            remoteStream.videoTracks.get(0).addRenderer(new VideoRenderer(remoteRender));
            remoteStream.videoTracks.get(0).setEnabled(true);  //);
            VideoRendererGui.update(remoteRender, REMOTE_X, REMOTE_Y,
                    REMOTE_WIDTH, REMOTE_HEIGHT, scalingType, false);
        } else {
            VideoRendererGui.update(remoteRender, REMOTE_X, REMOTE_Y
                    , 0, 0
                    , scalingType, false);
        }
        if (remoteStream.audioTracks.size() > 0) {
            remoteStream.audioTracks.get(0).setEnabled(true);  //);
        }
    }
}
```

Figura B.4 Gestión multiconexión en WebrtcActivity.

B.5. Test con Espresso. Android.

Simulación de cincuenta conexiones/desconexiones de un vigilante a otro utilizando el Framework Espresso.

```
@Test
public void validateMainActivity() {

    //Pulsamos boton conectar
    onView(withId(R.id.connect_button)).perform(click());
    //Verificamos que está conectado
    onView(withId(R.id.btn_call)).check(matches(isDisplayingAtLeast(20)));

    for (int i=0;i<NUM_TEST;i++) {
        //Pulsamos boton conectar
        onView(withId(R.id.btn_call)).perform(click());

        //Marcamos llamada a cliente test
        // Click on the RecyclerView item at position 2
        onView(withId(R.id.recycler_view)).perform(RecyclerViewActions.actionOnItemAtPosition(NUM_CLIENT, click()));
        //Pulsamos boton conectar
        onView(withId(R.id.btn_connect)).perform(click());

        //Esperamos hasta conexión se establezca
        try {
            latch.await(8, TimeUnit.SECONDS);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        //Pulsamos boton colgar
        onView(withId(R.id.btn_stop)).perform(click());

        //Esperamos hasta cierren las conexiones
        try {
            latch.await(7, TimeUnit.SECONDS);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

}
```

Figura B.5. TestCase en Espresso.

B.6. Https en Express. NodeJs.

Activación del servidor web Https utilizando framework Express. Fichero App.js

```
34  var app = express();
35  var http_port = 80
36  var https_port = 8901
37
38  // all environments
39  app.set('port', process.env.PORT || https_port);
40  app.set('views', path.join(__dirname, 'views'));
41  app.set('view engine', 'ejs');
42  app.use(favicon(__dirname + '/public/images/favicon.ico'));
43  app.use(logger('dev'));
44  app.use(bodyParser.json());
45  app.use(bodyParser.urlencoded({ extended: true }));
46  app.use(methodOverride());
47  app.use(express.static(path.join(__dirname, 'public')));
48
49  var options = {
50    key: fs.readFileSync('./private/webrtc-key.pem'),
51    cert: fs.readFileSync('./private/webrtc-cert.pem'),
52  };
53
54  var server = https.createServer(options,app).listen(app.get('port'),function () {
55    console.log('Express https server listening on ' + ip.address() + ':' + https_port);
56  });
```

Figura B6. https en Express.

B.7. Lectura y registro cámaras IP. NodeJs.

Lectura del fichero de texto con las ips de las cámaras fijas y el registro en el sistema como clientes activos. Fichero App.js

```
55  //Leemos fichero de camaras fijas RTP
56  //console.log('Fichero RTP: ' + camerasfile);
57  fs.readFile('./private/camaras.txt', 'utf8', function(err, camerasdata) {
58      if( err ){
59          console.log('Fichero de camaras fijas no encontrado. ' + err);
60      }
61      else{
62          console.log('Fichero camaras encontrado, leyendo camaras: ');
63          var i = 1;
64          camerasdata.toString().split('\n').forEach(function (line) {
65              if (line.length > 3) {
66                  var base64data = new Buffer(line, 'binary').toString('base64');
67                  //var originaldata = new Buffer(base64data, 'base64');
68                  console.log('Camara '+ i + ' : ' + line);
69                  //Añadimos la camara con id base64 de la direccion ip.
70                  streams.addStream(base64data, 'Camara0'+ i);
71                  i++;
72              }
73          });
74      }
75  });
```

Figura B7. Registro cámara fijas IP.

B.8. Activación servidor multimedia. NodeJs.

Registro y activación del servidor Kurento, definiendo también la imagen por defecto que se solapará en el módulo de detección de rostros. Fichero App.js

```
77  //Definimos variables para el servidor kurento uri e imagen superpuesta
78  var ws_uri = 'ws://192.168.1.40:8888/kurento';
79  var kurentoapi = require('./app/kurento.js')(ws_uri,'http://' + ip.address() +'/images/danger.png');
```

Figura B8. Registro servidor multimedia.

B.9. API Kurento y transformación JSON. NodeJs.

Inicialización del servidor, utilizando API Kurento y si no se produce error, transformación de la respuesta JSON de Kurento al formato adecuado. Fichero /app/socketHandler.js.

```
28▼                kurentoapi.start(client.id, io, details,false, function(error, sdpAnswer) {
29▼                    if (error) {
30                         console.log('Error: ' + error);
31                         //Enviamos mensaje STOP
32                         msg ='{"type":"stop","from":"kurento"}';
33                         console.log('Message from: kurento to:' + details.to + ' type: STOP');
34                         otherClient.emit('message', JSON.parse(msg));
35                     }
36▼                    else {
37                        msg ='{"type":"answer","payload":{"type":"answer","sdp":' + JSON.stringify(sdpAnswer) + '},
38                        //onsole.log('Generada respuesta servidor: ' + sdpAnswer);
39                        console.log('Message from: kurento to:' + details.to + ' type: answer');
40                        otherClient.emit('message', JSON.parse(msg));
41                    }
42                });
```

Figura B9. Llamada API Kurento y transformación JSON.

B.10. Módulos y dependencias. NodeJs.

Módulos y dependencias necesarios para nuestra aplicación en el servidor Node. Fichero package.json.

```
{
  "name": "ProjectRTC",
  "version": "0.1.0",
  "author": "Tomas Gonzalez <bicacaro@gmail.com>",
  "description": "a webRTC live server plus Kurento MediaServer",
  "scripts": {
    "start": "forever start app.js",
    "stop": "forever stopall"
  },
  "license": "LGPL-2.1",
  "repository": {
    "type": "git",
    "url": "https://bicacaro@bitbucket.org/bicacaro/webrtc_server.git"
  },
  "dependencies": {
    "ip":">=1.0",
    "body-parser": "^1.12.2",
    "ejs": "1.0",
    "errorhandler": "^1.3.5",
    "express": "^4.12.3",
    "express-session": "^1.11.1",
    "forever": "0.10.8",
    "jade": "^1.9.2",
    "method-override": "^2.3.2",
    "morgan": "^1.5.2",
    "multer": "^0.1.8",
    "serve-favicon": "^2.2.0",
    "socket.io": ">=1.0",
    "kurento-client" : "6.4.0"
  }
}
```

Figura B10. Módulos y dependencias Servidor Señales.

Apéndice C. Instalación servidor multimedia.

Instalar en VirtualBox, Linux Ubuntu 14.04 Desktop 64bits.
A continuación, una captura de pantalla del proceso.

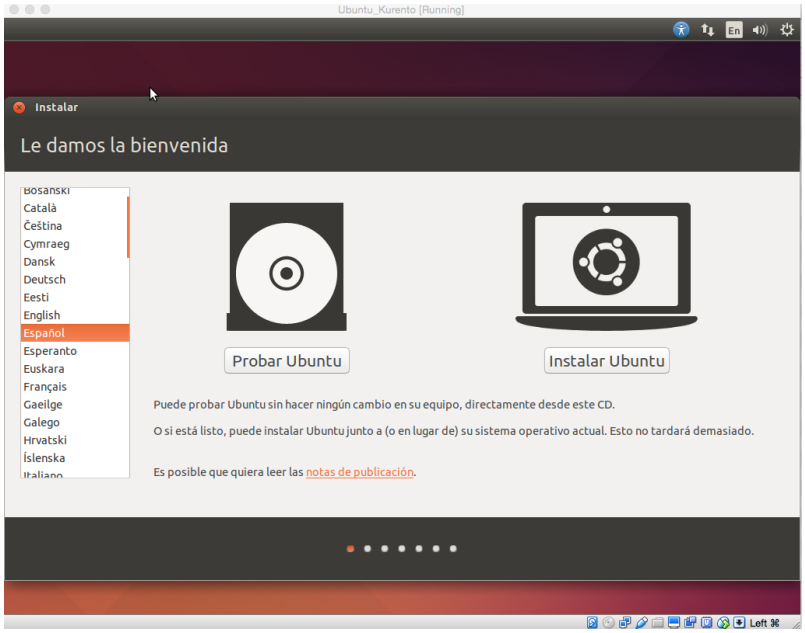

Figura C.1. Instalación Ubuntu 14.04 Desktop.

Configuración de red VirtualBox.

En el router se fija la ip para esa MAC, siempre 192.168.1.40, así se evita que cambie la IP del servidor multimedia. La siguiente imagen ilustra la configuración del red para la máquina virtual.

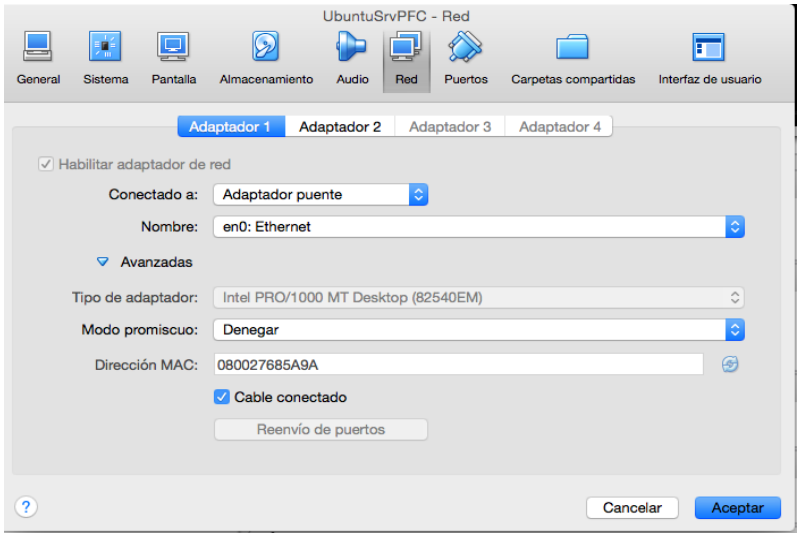

Figura C.2. Configuración de Red.

Carpetas compartidas host/server

A continuación se muestra la configuración de las carpetas compartidas, que facilitará el intercambio de ficheros desde el ordenador Host y el servidor multimedia.

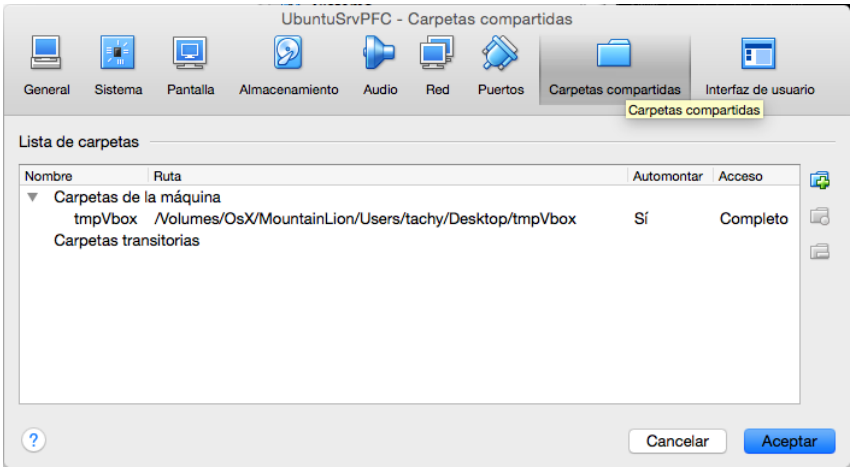

Figura C.3. Instalación Ubuntu 14.04 Desktop.

Instalación de Kurento

```
echo "deb http://ubuntu.kurento.org trusty kms6" | sudo tee
/etc/apt/sources.list.d/kurento.list
wget -O - http://ubuntu.kurento.org/kurento.gpg.key | sudo apt-key add -
sudo apt-get update
sudo apt-get install kurento-media-server-6.0
```

Iniciar/Parar servicio:

```
sudo service kurento-media-server-6.0 start
sudo service kurento-media-server-6.0 stop
```

Apéndice D. Control de versiones.

El proyecto, tanto la parte de la aplicación móvil, como del servidor de señales, utilizó Bitbucket para el control de versiones.

Proyecto Android.

https://bicacaro@bitbucket.org/bicacaro/webrtc_android.git

A continuación se muestra el histórico de actualizaciones del proyecto.

Tomás R. González Barroso / WebRTC_Android

Commits

All branches

Find commits

Author	Commit	Message	Date	Builds
Tomás R. Gonz..	959beb7	Enviamos señal(leve) para cerrar conexión en destino. En multillamadas, sólo activamos el audio del cliente en pa...	2016-05-20	
Tomás R. Gonz..	9df4cd8	Añadimos TestCase para controlar tiempos de conexión.	2016-05-17	
Tomás R. Gonz..	5ae5f21	Actualizaciones Android Studio.	2016-05-16	
Tomás R. Gonz..	0a1b5df	Permitimos multiconexión con cámaras fijas.	2016-05-16	
Tomás R. Gonz..	f7cbb8f	Mejoras en la multiconexión, permitimos cambiar y visualizar todas las conexiones remotas.	2016-05-13	
Tomás R. Gonz..	645ae35	Añadimos multiconferencia.	2016-05-13	
Tomás R. Gonz..	5c1925c	Permitimos mensajes para cerrar conexión(leave).	2016-05-06	
Tomás R. Gonz..	f306821	Añadimos recursos raw.	2016-04-27	
Tomás R. Gonz..	09093d6	Ajustes para el almacenamiento en servidor Kurento y corrección de errores. Actualizamos libjingle a 11139.	2016-04-20	
Tomás R. Gonz..	062a3e0	Corrección de errores en las rellamadas y en llamadas sólo de audio.	2016-04-11	
Tomás R. Gonz..	aba72a8	Mejora de interfaz, mostrando status e ID de la llamada. Mostramos botón para "colgar". Permitimos llamadas de ...	2016-04-07	
Tomás R. Gonz..	764da35	Mejora de interfaz, agregados botones audio/video. Añadida en preferencias selección de cámara.	2016-04-04	
Tomás R. Gonz..	bf0956d	Permitimos conexiones seguras SSL.	2016-03-31	
Tomás R. Gonz..	9d33d7f	Añadido Activity mostrando lista de clientes disponibles permitiendo llamarlos.	2016-03-30	
Tomás R. Gonz..	ab2bda0	Añadido menú ajustes Id y añadido icono para llamadas	2016-03-29	
Tomás R. Gonz..	7a6b07f	Añadido menú ajustes de servidor y cambiado icono app.	2016-03-28	
Tomás R. Gonz..	6f9cde6	Eliminado Fragment.	2016-03-15	
Tomás R. Gonz..	c5abe87	Inicial	2016-03-14	

Figura D.1. Histórico de actualizaciones Android.

Para realizar las actualizaciones, se utilizó el propio Android Studio que se integra perfectamente con las diferentes plataformas de control de versiones, en particular se utilizó el apartado "GitHub" como se muestra en la siguiente imagen.

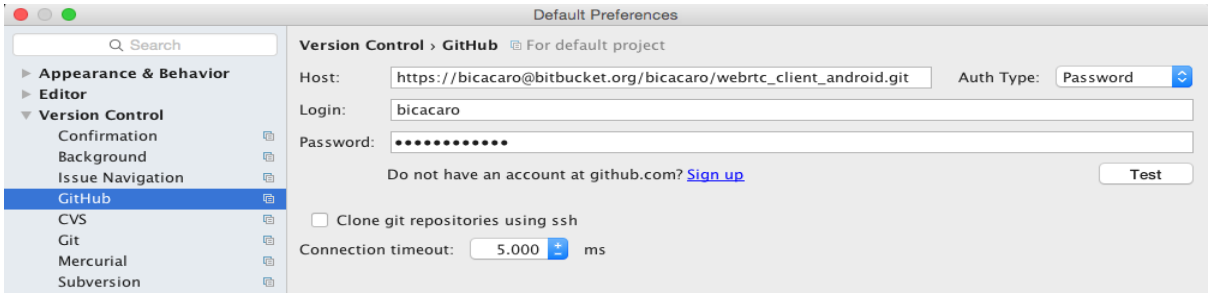

Figura D.2. Configuración Bitbucket Android Studio.

Proyecto NodeJs (Signaling Server)

https://bicacaro@bitbucket.org/bicacaro/webrtc_server.git

A continuación se muestra el histórico de actualizaciones del proyecto.

Tomás R. González Barroso / WebRTC_Server

Commits

All branches — Find commits

Author	Commit	Message	Date	Builds
Tomás R. Gonz..	6b5e1a8	Liberamos recursos en Kurento al cerrar la conexión.	2016-05-20	
Tomás R. Gonz..	806c6a7	Corrección y mejoras en la conexión a cámaras fijas desde navegadores web.	2016-05-09	
Tomás R. Gonz..	637c592	Mejorada interfaz de usuario para navegador. Reorganizadas carpetas.	2016-05-06	
Tomás R. Gonz..	e056bd4	Permitimos conexiones a camaras IP fijas (hhtp/rtsp), utilizando la configuracion desde el fichero camaras.txt.	2016-05-05	
Tomás R. Gonz..	760cdd9	README.md edited online with Bitbucket	2016-04-20	
Tomás R. Gonz..	26b5636	Permitimos grabación en Kurento llamadas video/audio, sólo video, sólo audio.	2016-04-20	
Tomás R. Gonz..	ce5cfd0	Añadida la opción de almacenar a fichero en el servidor multimedia.	2016-04-19	
Tomás R. Gonz..	a7d838b	Implementada detección de rostros utilizando modulo de Kurento (FaceOverlay).	2016-04-19	
Tomás R. Gonz..	8ca2b30	Ya tenemos una conexión bidireccional entre los clientes y servidor multimedia Kurento.	2016-04-19	
Tomás R. Gonz..	edb0944	Añadida comunicación servidor multimedia KURENTO. Pendiente enviar a KMS OnIceCandidate.	2016-04-19	
Tomás R. Gonz..	81258bc	Corrección de dependencias.	2016-04-14	
Tomás R. Gonz..	9e2fceb	Servidor Ice, funcionando con clientes Android y navegadores Chrome, Firefox.	2016-04-14	

Figura D.2. Histórico de actualizaciones Server.

Para realizar las actualizaciones, se ha utilizado la herramienta SourceTree.

Figura D.3. SourceTree.

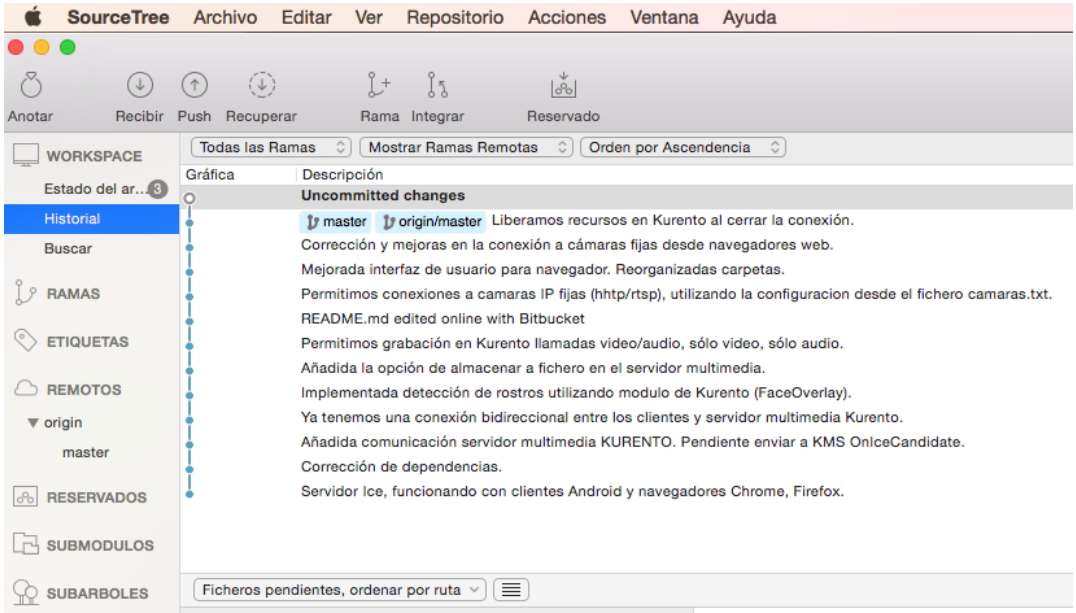

Figura D.4. Histórico SourceTree.

Apéndice E. Conference paper.

El estudio realizado de la comparación de los tiempos de conexión entre los protocolos RTSP y WebRTC, utilizando la parte de la aplicación móvil de este proyecto, fue presentado como trabajo en UCAmI 2016 (10th International Conference on Ubiquitous Computing and Ambient Intelligence) que se celebrará desde el 29 de noviembre al 2 de diciembre de 2016.

A continuación se muestra el documento presentado con el Título **"Real-Time Streaming: A Comparative Study Between RTSP and WebRTC"** .

Real-Time Streaming: A Comparative Study Between RTSP and WebRTC

Iván Santos-González, Alexandra Rivero-García, Tomás González-Barroso, Jezabel Molina-Gil and Pino Caballero-Gil

Dapartamento de Ingeniera Informtica y de Sistemas, Universidad de La Laguna, 38271 La Laguna, Tenerife, Espaa
(jsantosg, ariverog, bicacaro, jmmolina, pcaballe)@ull.edu.es

Abstract. This work presents a comparative study between two of the most used streaming protocols, RTSP and WebRTC. It describes a system designed to evaluate times at live streaming: establishment time, response time and delay time from a single source to a large quantity of receivers with the use of smartphones. Two systems that use the commented protocols have been implemented, specifically, two Android applications that use these protocols in the scope of video surveillance at airports. Both systems are composed of a mobile application and a web service. The design of the systems has been done avoiding differences between these protocols for P2P communication in the same local network. Several simulations have been performed to compare them and the obtained results have been used for a comparative study between streaming establishment and sending package times of each protocol.

Keywords: Streaming video, RTSP, WebRTC, Android

1 Introduction

In the last years the high penetration of streaming video through Internet has suffered a huge increase of use. In [3] the company Cisco predicts that the Internet video traffic will be 80 percent of all consumer Internet traffic in 2019, up from 64 percent in 2014, without taking into account the Peer-to-Peer (P2P) communication. The video traffic will be from 80 to 90 percent of global consumer traffic by 2019. For this reason, some of the most important companies of multimedia content have tried to develop different strategies to improve the Quality of Experience (QoE) of their services. The fast evolution of performances and the low cost of mobile devices make them excellent candidates to provide new streaming features for different proposals.

Video streaming is the process where a video is transmitted from a point to one or multiple destinations. Video on Demand (VoD) is defined as a service in which high-speed of data services are used to provide customers with streaming content [9]. In this paper we analyse two of the most used protocols in streaming based on mobile phones. The objective is the evaluation of the speed in both systems using Android applications and similar contexts.

On the one hand, Real Time Streaming Protocol (RTSP) [16], based on Real-time Transport Protocol (RTP) [17] is analysed. This protocol is not based on the establishment of connection Instead, the server has a session associated to an identifier.

On the other hand, Web Real-Time Communications (WebRTC), a project maintained by Google based on RTP is analysed. This system allows real-time communications through some Application Programming Interfaces (API) with a high quality, low latency and low bandwidth consumption.

We are interested in the problem of live streaming establishment time, response time and delay time from a single source to a large quantity of receivers with the use of these protocols. The aim of this work is to know which protocol is better to be used in the mobile platforms and specifically in the Android platform. Moreover, we are interested in having quantitative measurements to know how much an protocol is better than the other to use in our future implementations

The paper is organized as follows. In Section 3, we analyse different solutions used currently. In Section 4 we describe the system based on RSTP in detail. Section 5 defines a similar system based on WebRTC. Finally, we present an in-depth comparison between theses systems in Section 6. Conclusions and future works close the paper in Section 7.

2 Preliminaries

Sharing information in a network between nodes can be done in different ways. The kind of the information that the system shares usually determines the method for the chosen communication. In this proposal, sharing audio and video media is the main objective For this reason, the most important point is the guarantee of a low latency, low jitter and efficient transmissions, but occasional losses could be tolerated.

The media streaming protocol is defined taking into account the structure of the packets and the algorithms used to send real-time media on a network. Different media streaming protocols are available today, which differ in a few implementation details. Two communication categories are: push-based and pull-based protocols [1].

- Push-Based Media Streaming Protocols: In these protocols, when a server and a client establish a connection, the server starts to stream packets to the client until the client stops or interrupts the session.
- Pull-Based Media Streaming Protocols: In these protocols, the media client is the active entity that requests content from the media server. Therefore, the server's response depends on the client requests when the server is idle or blocked for that client.

In table 1, some examples of different streaming protocols and their characteristics are shown. This information is based on [19], where there is an analysis of the transmission protocols for the H.265 encoder.

In [1] there is an analysis where the efficient, overhead-wise, of push-based protocol over pull-based streaming is explained. For this reason, in this paper we analyse the efficiency of two protocols based on RTP.

3 Related Work

There are some analysis based on the comparison between different streaming protocols. In [20] there is a study of HTTP, RTSP and Dynamic Adaptive Streaming over

Table 1: Comparison between Push-based and Pull-based streaming protocols.

Characteristic	Push-based	Pull-based
Source	Broadcasters and servers like Windows media, QuickTime, RealNetworks Helix Cisco CDS/DCM	Web servers such as LAMP, Microsoft BS RealNetworks Helix, Cisco CDS
Protocols	RTSP, RTP, UDP	HTTP
Bandwidth usage	Likely more efficient	Likely less efficient
Video monitoring	RTCP (RTP transport)	Currently propietary
Multicast support	Yes	No

HTTP (DASH) [18] protocols on a smartphone. That paper is based on the calculation of switch delay or the time between a user sends a command and the screen of the client suffers theses changes. They analyse these protocols and the response when the user switches the mode. In [12] HTTP, RTSP, and IMS are compared. That work describes some approaches promising the synthesis of IMS, HTTP, and RTSP based networks.

Authors of [7] describe measurements collected from DASH and WebRTC implementations while moving at walking speeds through an 802.16e WiMAX network. They collected data from an application, the network and physical layers under different wireless environments. The characteristics that directly impact the quality of video service in mobile data network were also identified. Finally, they conclude that to adapt the channel conditions, these services did not achieve acceptable service quality for the mobile users under different network conditions.

In the proposal [11], a QoE instrumentation for video streaming on smartphone, called VLQoE, has been presented. They use VLC media player to record a set of metrics from the user interface, application-level, network-level and from the available sensors of the device. They present two state models based on the time to the HTTP and RTSP protocols via streaming 3.5G.

4 Video Streaming Srotocols

In this Section, the analysed streaming protocols are explained. On the one hand, the RTSP scheme is described. On the other hand, WebRTC system based on API calls is analysed.

4.1 RTSP Protocol

The Real Time Streaming Protocol (RTSP) [16] is a non-connection oriented application layer protocol that uses a session associated to an identifier. RTSP usually uses the UDP protocol to share the video and audio data and TCP for the control (TCP is used just if it is necessary). The syntax of the RTSP protocol is similar to the syntax of HTTP protocol and supports the next operations:

- **Retrieval of media from media server:** The client can request a presentation description via HTTP or some other method. If the presentation is being multicast, the presentation description contains the multicast addresses and ports to be used for the continuous media. If the presentation is to be sent only to the client via unicast, the client provides the destination for security reasons.
- **Invitation of a media server to a conference:** A media server can be "invited" to join an existing conference, either to play back media into the presentation or to record all or a subset of the media in a presentation. This mode is useful for distributing teaching applications. Several parties in the conference may take turns "pushing the remote control buttons".
- **Addition of media to an existing presentation:** Particularly for live presentations, it is useful if the server can tell the client about additional media becoming available.

The structure of an URL for RTSP is very similar to the URL in HTTP, with the only difference in the used scheme *rtsp://* in RTSP instead of *http://* in the HTTP protocol but adds new request methods such as DESCRIBE, SETUP, PLAY, PAUSE and TEARDOWN. The DESCRIBE method is used to obtain a description of the presentation or object appointed by the URL RTSP. The server responds to this request with a description of the requested resource. This response corresponds to the initialization phase of RTSP and contains the list of the necessary multimedia streams. On the other hand, the SETUP method is used to establish how is the stream transported, the request contains the URL of the multimedia stream and a transport specification that usually includes the port to receive RTP data (video and audio) and another one to the RTCP [10] data (metadata). The server responds confirming the selected parameters and fill the other parts, as they are the ports selected by the server. Every stream has to be configured before sending a PLAY request. The PLAY request is used to start the data stream shipment by part of the server using the ports configured with the SETUP request method. Moreover, the PAUSE method stops one or all streams temporarily to resume it later with a PLAY request. Finally, the use of the TEARDOWN request method stops the shipment of data releasing all resources. Note that first of all a TCP connection is established between the client and the server, started by the client and typically over the well-known port of TCP, the 554 port.

The exact performance and request order of the RSTP protocol is shown in figure 1.

4.2 WebRTC Protocol

WebRTC [2] is an API created by the World Wide Web Consortium (W3C) that allows the browser apps to make calls, video chats and the use of P2P files without any plugin. The first implementation of WebRTC was created by Google and released as Open Source. Different organism as the Internet Engineering Task Force (IETF) to standardize the used protocols and the W3C with the browser APIs have been working on this implementation.

The main components of WebRTC are the next:

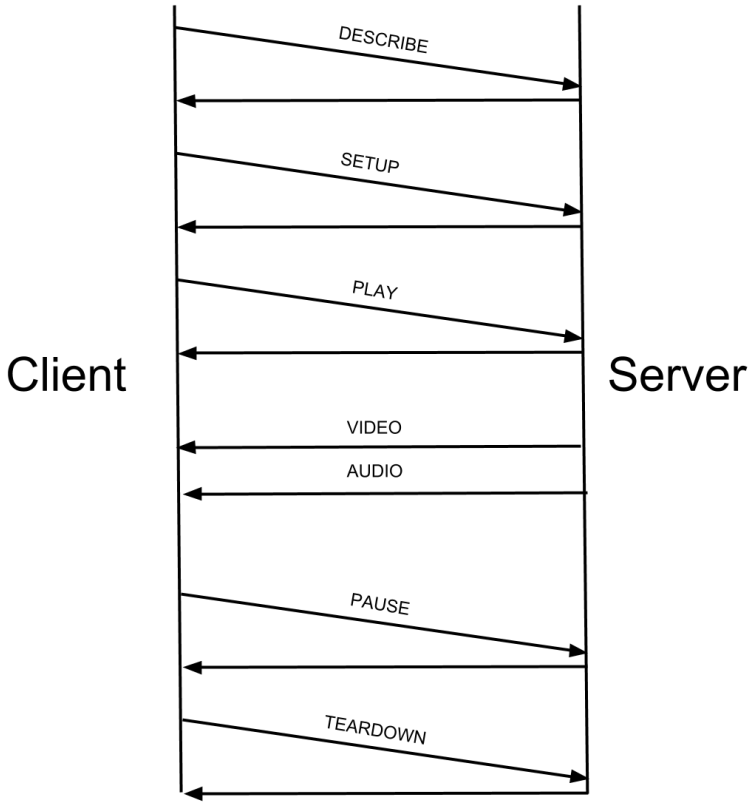

Fig. 1: RTSP Request Order

– **getUserMedia:** It allows obtaining video or audio streams from the hardware (microphone or camera). This API call allows us to get a screenshot or share our screen with other users too.
– **RTCPeerConnection:** It allows us to set up the audio/video stream. This consists in a lot of different tasks like the signal processing, the codec execution, the bandwidth administration, the security of the streaming, etc. This API call allows implementing this different task without the intervention of the programmer.
– **RTCDataChannel:** It allows sharing video or audio data between connected users. RTCDataChannel uses a bidirectional communication between peers and allows the exchange of any data type. To do this, RTCDataChannel uses Websockets, which allows us to get a bidirectional communication between the client and the server and allows using a slower and reliable communication over TCP or a faster and non-reliable communication over UDP.
– **geoStats:** API call that allows getting different statistics about a WebRTC session.

The performance of the WebRTC protocol used to make a call with the API calls commented before is shown in figure 2.

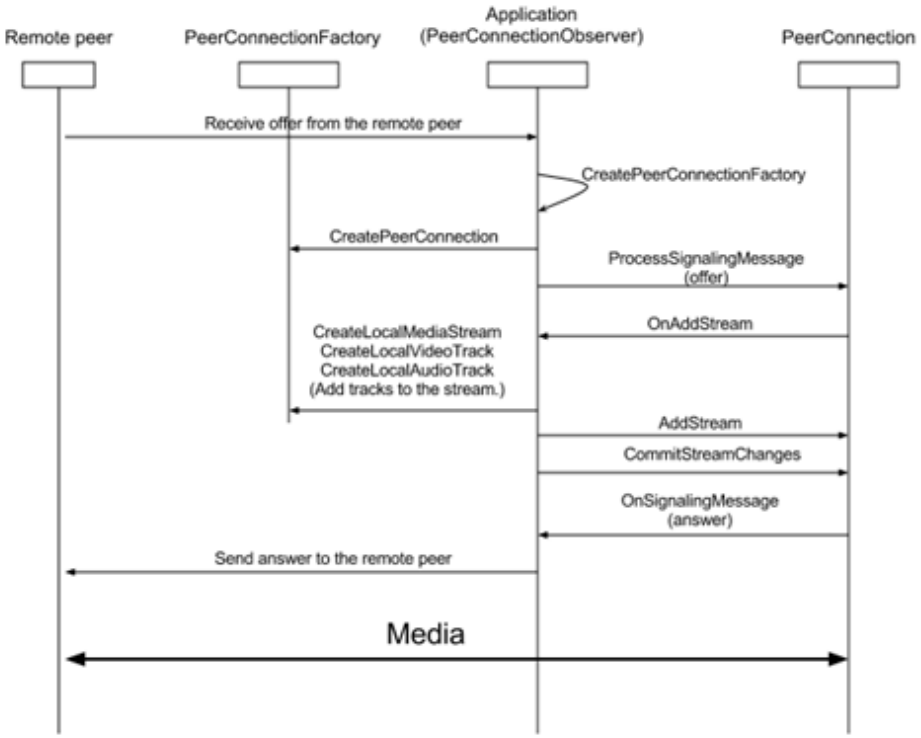

Fig. 2: WebRTC Request Order

5 Implemented Systems

The implemented system consists in an Android application that covers the use case of the video surveillance at an airport environment. This application has been developed on an Android platform and uses the RTSP protocol in the application called “A” and the WebRTC protocol in the application called “B”. It makes it possible that every airport employee can take videos at the airport from their smartphones. Furthermore, all these videos can be viewed by others in real time or by the security personnel in the control room. The developed application supposes a huge improvement on the security of the airport because, in combination with the previously installed video surveillance cams, the system allows the airport personnel to record every strange situation in the airport and inform other co-workers or the security personnel of the airport in real time in

order to be helped or assisted by them. In figure 3 we can see few screenshots of the developed application. Note that both media servers have been released in a Windows 10 X64 machine, with an Intel core i7-3612QM CPU, 8 GB of RAM memory and a 240 GB Intel SSD.

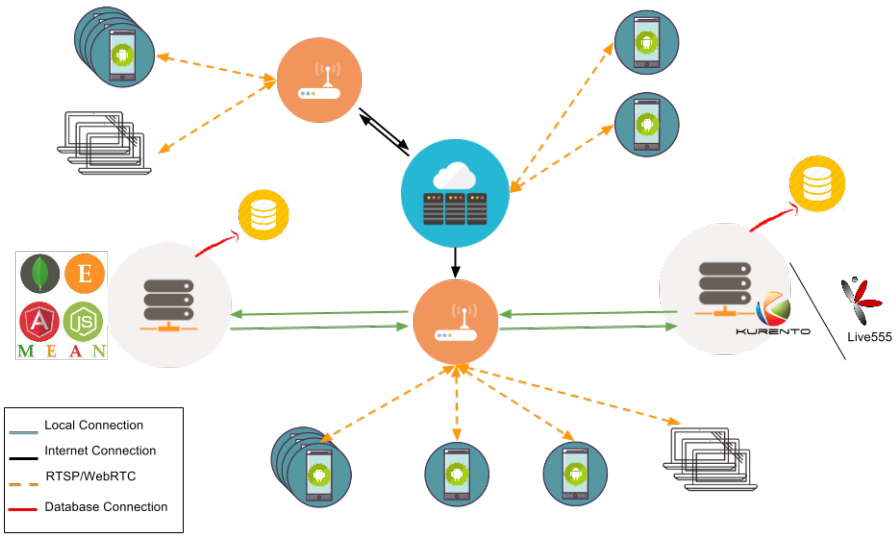

Fig. 3: System Global View

5.1 RTSP scheme

The architecture of the application “A” consists in the use of P2P communications through the RTSP protocol in places where the two users are in the same network, and through the use of a Live555 media server in cases where we have not a dedicated network and we have to use the Internet connection. Live555 media server [14] is a complete open source RTSP server that supports the most common kinds of media files. Moreover, this server can stream multiple streams from the same or different files concurrently, using by default the UDP protocol to transmit the streams, but we can transmit over the TCP protocol if we decide it.

The Android application we developed acts as the server and/or the client depending on the situation. This application allows us to connect with multiple users and send their real time stream. In the implementation of this system we decided to use the last

version of Android available, which was Android 6.0 that corresponds to the API 23 of the Android SDK. In this version of Android and consequently in no previous ones, the RTSP protocol is not implemented so we have to use to third party libraries that implement this protocol. At this point, we decided to use the libstreaming open source library [8] that implements the RTSP protocol both the server part and the client part. This library allows us to stream the camera and/or the microphone of an Android device. The library supports every device with an equal or higher version to Android 4.0 and supports the H.264, H.263, AAC and AMR media codecs. Moreover, to make the application compatible with a higher number of devices we decided to capture the video using the MediaRecorder API that it is available from the Android 4.0 version. The MediaRecorder API [15] is not thought to be used for streaming purposes but it can be used to get data from the peripherals of the phone. To do that, this method works fine, so we have to use a ParcelFileDescriptor to record the captured stream instead of using a file to do it. Apart from this method to record, we could use the MediaCodec API [4] to record the stream using a buffer-to-buffer method or a surface-to-buffer method. It would be necessary the use of the Android 4.1 or Android 4.3 versions respectively, with the consequent loss of potential users.

5.2 WebRTC Scheme

The architecture of the application "B" consists in the use of P2P communications through the WebRTSP protocol in places where the two users are in the same network, and through the use of a Kurento media server[13] in cases where we have not a dedicated network and we have to use the Internet connection. Kurento is a WebRTC media server and it has a set of client APIs that simplifies the development of advanced video applications for web and smartphone platforms. Kurento Media Server features include a group of communications, transcoding, recording, mixing, broadcasting and routing of audio-visual flows and it also provides advanced media processing capabilities involving computer vision, video indexing, augmented reality and speech analysis.

The developed application acts as client and/or server depending on the situation. It allows us to connect with one or multiple users and send their real time stream. We decided to use the last version of Android available that was Android 6.0 that corresponds to the API 23 of the Android SDK. In this version of Android and consequently in no previous ones, the WebRTC protocol is not implemented at the core of Android and so we have to use to third party libraries that implement this protocol. At this point, we decided to use the Libjingle library [6] that it is an open source library written in C++ language by Google and that allows to establish P2P connections and to develop P2P applications. To use this library in Android we had to compile the source code to generate the JNI API that lets the Java implementation reuse the C++ implementation of the same API. This library allows us to stream the camera and/or the microphone of an Android device. The library supports every device with a equal or higher version to Android 4.1 and supports the H.264, H.265, VP8 and VP9 media codecs. Moreover, we decided to use the Socket.io library too. This library allows us to establish communications through the Websocket protocol, for this reason it is used in the initial communications between the Android App and the SIP (Session Initiation Protocol) server. The developed application acts as client and/or server depending on the situation.

6 Comparative Study

In this section the results of the different measurements taken about the connection establishment time and stream reception time in the implementations of two stream protocols are analysed. The measurements have been taken using the system time that is implemented in Java programming language, and therefore in the Android platform, in the class System with the function currentTimeMillis(). This function returns the current time in milliseconds. To get the connection establishment time we take the time in the moment when the user A launches the streaming and then we take the time again when the connection is completely established. Then, we perform the subtraction between the second measurement and the first one to obtain the connection establishment time. On the other hand, the stream reception time is the time between the correct communication establishment and the time when we receive the first video packet.

The different measurements have been done using the same smartphone, a LG L9 with 1GB of RAM memory, CPU 1GHz Dual Core and 4 GB of storage memory and the 4.4.4 Android version. To perform these measurements, we implemented a series of tests using Expresso tests [5] that simulate the exact behaviour of using the application. To get a significant number of measurements we performed the same test for 80 iterations for each measurement. Figure 4 shows the comparative analysis between the two systems: RTSP in 4a and WebRTC in 4b.

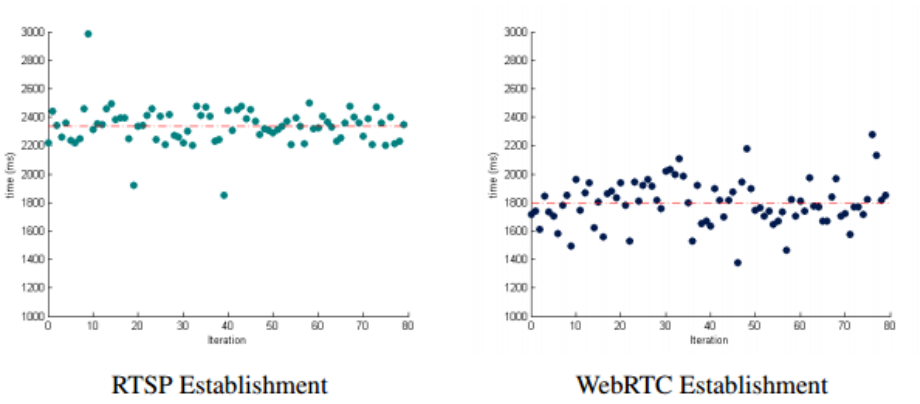

Fig. 4: Connection Establishment Time

We can see how the system that uses the RTSP protocol is slower than the WebRTC protocol having an average of connection establishment time of 2,304 seconds in the case of the RTSP protocol and an average of 1,835 seconds in the case of the WebRTC protocol. The difference between the two implemented protocols in the connection establishment time average is 0,469. In spite of having some outlier points in the graphs of the two systems, we can say that WebRTC protocol is more efficient in the connection establishment time than the RTSP protocol.

On the other hand, we have the stream reception time comparison that can be seen in figure 5, RTSP in 5a and WebRTC in 5b.

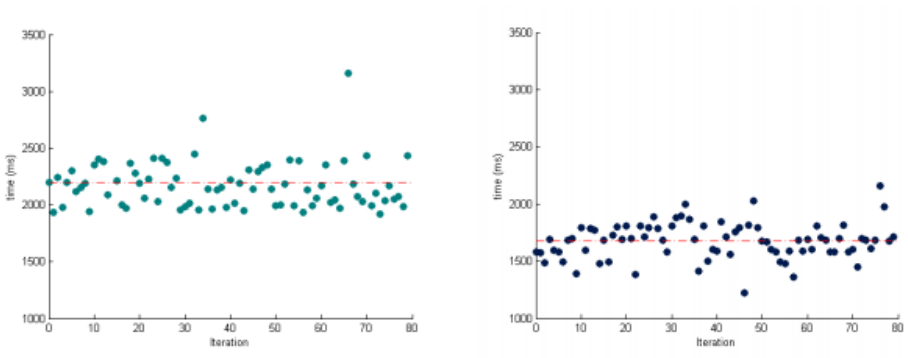

Fig. 5: Stream Reception Time

There, we appreciate a similar behaviour to that in case of the connection establishment time and we can see an average of 2,161 seconds in the case of RTSP and an average of 1,709 in the case of the WebRTC protocol being the average difference of 0.452. If we add the two difference, we obtain a total difference between the two protocols of almost 1 second favourable to the WebRTC protocol. This improvement can be viewed easily when we use the two implemented systems having a lag of about 1 second that sometimes is incremented due to delays in the packet sending in the case of the RTSP implemented system and having almost no lag in the case of the WebRTC implemented system. Moreover, we suppose that the delay in the stream reception time could be repeated in other moments of the streaming and increase the lag of the system.

We can observe the difference between both protocol times in figure 6. The difference in the times is pretty clear. The RSTP protocol always spend more time than WebRTC in both communication establishment and stream reception time.

7 Conclusions and Future Work

This work presents a comparative study between two of the most used live streaming protocols, WebRTC and RTSP. A system based on each protocol was implemented taking into account the same scheme and conditions, based on the use of Android smartphones. The obtained measures are based not only on sending packages of both protocols, but also on the communication establishment of each one. Several simulations have been performed to compare the streaming establishment and sending packages with each protocol, and the obtained results are promising for WebRTC. In the simulations, significant advantages in WebRTC protocol time over RTSP have been obtained for both communication establishment and sending packages. As future work, several

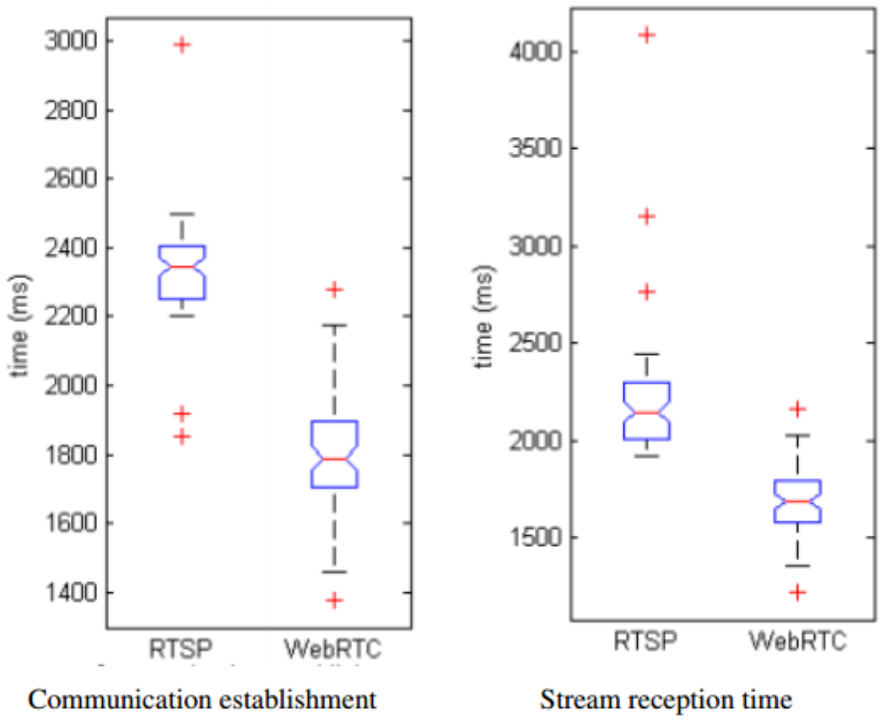

Fig. 6: Comparative Between Times of RTSP and WebRTC

possible future tasks can be mentioned, such as the creation of comparatives taking into account more protocols such as DASH streaming, Adobe HTTP Dynamic Streaming or Microsoft Smooth Streaming in the same environment with smartphones.

Acknowledgments

Research partially supported by TESIS2015010102, TESIS2015010106, RTC-2014-1648-8, TEC2014-54110-R, MTM-2015-69138-REDT and DIG02-INSITU.

References

1. Begen, A.C., Akgul, T., Baugher, M.: Watching video over the web: Part 1: Streaming protocols. Internet Computing, IEEE 15(2), 54–63 (2011)
2. Bergkvist, A., Burnett, D., Jennings, C.: A. narayanan," webrtc 1.0: Real-time communication between browsers. World Wide Web Consortium WD WD-webrtc-20120821 (2012)
3. Cisco: Consumer internet traffic report (2015), http://www.cisco.com
4. Codec, A.M.: Mediacodec api, https://developer.android.com/reference/android/media/MediaCodec.html?hl=es

5. Developers, A.: Testing with expresso, https://developer.android.com/training/testing/ui-testing/espressotesting.html?hl=es
6. Developers, G.: Libjingle library, https://developers.google.com/talk/libjingle/developer_guide
7. Fund, F., Wang, C., Liu, Y., Korakis, T., Zink, M., Panwar, S.S.: Performance of dash and webrtc video services for mobile users. In: Packet Video Workshop (PV), 2013 20th International. pp. 1–8. IEEE (2013)
8. fyhertz: Libstreaming library, https://github.com/fyhertz/libstreaming
9. Garfinkle, N.: Video on demand (Jun 25 1996), uS Patent 5,530,754
10. H. Schulzrinne, C.U.: Rtp: A transport protocol for real-time applications, https://www.ietf.org/rfc/rfc3550.txt
11. Ickin, S., Fiedler, M., Wac, K., Arlos, P., Temiz, C., Mkocha, K.: Vlqoe: Video qoe instrumentation on the smartphone. Multimedia Tools and Applications 74(2), 381–411 (2015)
12. Khan, S.Q., Gaglianello, R., Luna, M.: Experiences with blending http, rtsp, and ims [ip multimedia systems (ims) infrastructure and services]. Communications Magazine, IEEE 45(3), 122–128 (2007)
13. Kurento: Official web page, https://www.kurento.org/
14. Live555: Official web page, http://www.live555.com/
15. Recorder, A.M.: Mediarecorder api, https://developer.android.com/reference/android/media/MediaRecorder.html
16. Schulzrinne, H.: Real time streaming protocol (rtsp) (1998)
17. Schulzrinne, H., Casner, S., Frederick, R., Jacobson, V.: Real-time transport protocol. RFC1899 (2003)
18. Stockhammer, T.: Dynamic adaptive streaming over http–: standards and design principles. In: Proceedings of the second annual ACM conference on Multimedia systems. pp. 133–144. ACM (2011)
19. UMESH, A.: Performance analysis of transmission protocols for h. 265 encoder (2015)
20. Zhang, H., Al-Nuaimi, A., Gu, X., Fahrmair, M., Ishibashi, R.: Seamless and efficient stream switching of multi-perspective videos. In: Packet Video Workshop (PV), 2012 19th International. pp. 31–36. IEEE (2012)