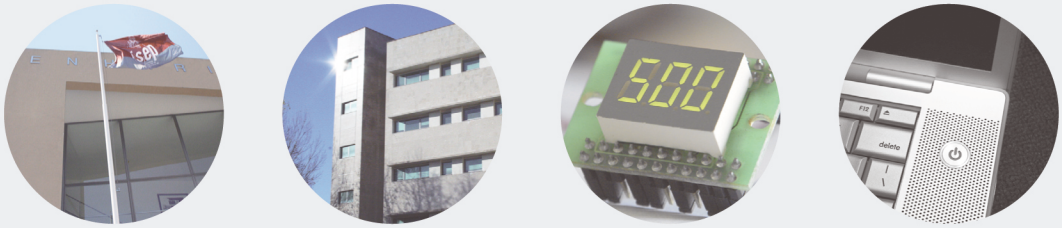
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES



SOLUÇÃO DE ASSISTÊNCIA REMOTA COM REALIDADE AUMENTADA VIA WEBRTC

ROBERTO CARLOS MOREIRA BRIOTE
novembro de 2021

POLITÉCNICO
DO PORTO

SOLUÇÃO DE ASSISTÊNCIA REMOTA COM REALIDADE AUMENTADA VIA WEBRTC

Roberto Carlos Moreira Briote

Departamento de Engenharia Eletrotécnica

Mestrado em Engenharia Eletrotécnica e de Computadores

Área de Especialização em Telecomunicações

Relatório elaborado para satisfação parcial dos requisitos da Unidade Curricular de Tese/Dissertação do Mestrado em Engenharia Eletrotécnica e de Computadores

Candidato: Roberto Carlos Moreira Briote, Nº 1120705, 1120705@isep.ipp.pt
Orientação científica: Professor Jorge Botelho Da Costa Mamede, jbm@isep.ipp.pt

Empresa: Accenture Technology Solutions

Supervisão: Ricardo Vilas Boas, ricardo.vilasboas@accenture.com

Departamento de Engenharia Eletrotécnica
Mestrado em Engenharia Eletrotécnica e de Computadores
Área de Especialização em Telecomunicações

2021

Agradecimentos

Quero agradecer, em primeiro lugar, aos meus pais, irmã e restante família, pelo apoio incondicional ao longo de todo o meu percurso académico, permitindo-me cumprir um dos meus objetivos de vida. À minha namorada Núria Romão, pela paciência e apoio constante em todos os momentos desta tese. Aos meus amigos por toda a ajuda prestada, através do enaltecimento das minhas capacidades e por me incentivarem a evoluir dia após dia.

Em segundo, agradeço a ajuda prestada pelo Professor Jorge Mamede, meu orientador no ISEP, pelo profissionalismo e acompanhamento na organização e coordenação desta tese. Gostaria também de agradecer, aos meus colegas de curso, com os quais partilhei imensos momentos de convívio e de estudo ao longo do ensino superior.

Por fim, mas não menos importante, gostava de agradecer ao Engenheiro Ricardo Vilas Boas e à Mariana Reis, pela oportunidade dada para a realização desta tese na empresa Accenture e pelo acompanhamento e apoio ao longo deste processo.

Resumo

Na atualidade, a sociedade recorre diariamente a canais de suporte para resolver os seus problemas. Com a evolução destes canais para plataformas digitais, a triagem está a tornar-se obsoleta, devido à lentidão do processo de identificação de problema e à sua tentativa de resolução. Neste seguimento, a Accenture decidiu propor este projeto de tese, que visa o desenvolvimento de uma solução de assistência remota com realidade aumentada, respondendo à necessidade dos seus clientes com canais de suporte, necessidade que se tornou mais evidente durante a pandemia mundial de COVID-19.

O ponto de partida desta tese, passou pelo estudo das soluções existentes. Deste modo, foi possível especificar a solução com base nesse estudo, nomeadamente, a sua arquitetura. A fase de implementação da solução, contempla a criação de uma aplicação *web*, provida de inúmeras funcionalidades, das quais se destaca a videochamada com anotações de realidade aumentada em tempo real no vídeo.

No que diz respeito aos testes de sistema efetuados para validar a solução desenvolvida, foi concluído que os casos de uso propostos na especificação da solução, se encontram operacionais e que as métricas estudadas, como a largura de faixa e o *round trip time* estão com valores esperados para aplicações de assistência remota em tempo real.

Concluiu-se que a solução desenvolvida promove uma assistência remota inovadora, facilitando quer os processos de triagem como a resolução dos problemas identificados. A funcionalidade das anotações de realidade aumentada é a que mais contribui para a inovação da solução final.

Palavras-Chave

Accenture, Assistência Remota, OpenCV, Realidade Aumentada, Videochamada, WebRTC.

Abstract

Currently, society uses support channels daily to solve its problems. With the evolution of these channels to digital platforms, triage is becoming obsolete, due to the slowness of the problem identification process and its attempt to solve it. Following this, Accenture decided to propose this thesis, which aims to develop a remote assistance solution with augmented reality, responding to the needs of its customers with support channels, this became more evident during the COVID-19 worldwide pandemic.

The starting point of this thesis was the study of existing solutions. In this way, it was possible to specify the solution based on this study, namely, its architecture. The solution implementation phase contemplates the creation of a web application, provided with numerous functionalities, among which the video call with real-time augmented reality annotations in the video stands out.

Regarding the system tests carried out to validate the developed solution, it was concluded that the use cases proposed in the solution specification are operational and that the metrics studied, such as bandwidth and round trip time have values expected for real-time remote assistance applications.

In conclusion, the developed solution promotes innovative remote assistance, facilitating both processes, the triage and the resolution of identified problems. In terms of innovation. The functionality of augmented reality annotations is what contributes most to the innovation of the final solution.

Keywords

Accenture, Remote Assistance, OpenCV, Augmented Reality, Videocall, WebRTC.

Índice

Índice de Figuras

Índice de Tabelas

Acrónimos

API	–	Application Programming Interface
CDN	–	Content Delivery Network
CMAF	–	Common Media Application Format
CSS	–	Cascading Style Sheets
DTLS	–	Datagram Transport Layer Security
EJS	–	Embedded JavaScript templating
FPS	–	Frames por segundo
GUI	–	Graphical User Interface
HLS	–	HTTP Live Streaming
HTML5	–	Hypertext Markup Language 5
HTTPS	–	Hypertext Transfer Protocol Secure
ICE	–	Interactive Connectivity Establishment
IETF	–	Internet Engineering Task Force
ITU-T	–	International Telecommunication Union-Telecommunication
MPEG-DASH	–	Moving Pictures Expert Group - Dynamic Adaptive Streaming over HTTP
NAT	–	Network Address Translation
NPM	–	Node Package Manager
OpenCV	–	Open Source Computer Vision Library

OSI	–	Open System Interconnection
P2P	–	Peer-to-peer
RA	–	Realidade Aumentada
RTCP	–	Real-time Transport Control Protocol
RTMP	–	Real Time Messaging Protocol
RTP	–	Real-time Transport Protocol
RTT	–	Round Trip Time
SCTP	–	Secure Control Transmission Protocol
SRTP	–	Secure Real-time Transport Protocol
SSL	–	Secure Sockets Layer
STUN	–	Session Traversal Utilities for NAT
TCP	–	Transmission Control Protocol
TURN	–	Transversal Using Relays around NAT
UDP	–	User Datagram Protocol
URL	–	Uniform Resource Locator
UUID	–	Universally unique identifier
VoIP	–	Voice over Internet Protocol
W3C	–	World Wide Web Consortium
WebRTC	–	Web Real-Time Communications
WSS	–	WebSocket over TLS

1. INTRODUÇÃO

Esta tese surge de uma proposta de estágio elaborada pela empresa Accenture, com o tema "SOLUÇÃO DE ASSISTÊNCIA REMOTA COM REALIDADE AUMENTADA VIA WEBRTC". Este tema permite abordar conceitos das áreas de Telecomunicações e Informática, uma vez que inclui, entre outros, o estudo de protocolos de redes e a toda a programação *web* por trás da solução.

Neste primeiro capítulo é efetuada a contextualização da tese, sendo expostos os motivos que levaram a Accenture a apresentar esta proposta de estágio. Além disso, são definidos os objetivos da tese e, por fim, é revelada a estrutura deste relatório de tese.

1.1. CONTEXTUALIZAÇÃO

Nos dias de hoje, quando uma pessoa tem a necessidade de uma intervenção de suporte em casa, seja por parte de uma operadora de telecomunicações, empresas de eletricidade, de água, seguradoras ou outras, é normal, recorrer-se a uma chamada, em que um especialista remoto tem de fazer uma triagem a um determinado problema apresentado por um cliente ou um operador no terreno.

Com a digitalização dos canais de suporte, a triagem está a tornar-se obsoleta. Uma chamada telefónica, um procedimento previamente estabelecido para sinalização do problema e, por fim, uma tentativa de resolução na mesma chamada é um processo lento e pouco objetivo.

A necessidade de desenvolver uma aplicação com este enquadramento sentiu-se ainda mais no período da pandemia mundial de COVID-19. Neste período, muitos clientes da Accenture sentiram dificuldades em resolver os problemas que não necessitavam do deslocamento de um técnico qualificado ao local, por exemplo, a troca de um *router* ou de uma *box*.

Uma solução de assistência remota com realidade aumentada (RA) pode ajudar, quer na triagem do problema como na sua resolução, evitando a deslocação de um técnico especializado, uma vez o problema é resolvido pelo próprio cliente.

A Accenture Portugal, como responsável de implementação e integração de soluções de *Field Force* em diversos clientes, com diferentes contextos e tecnologias, precisa de melhorar o seu portefólio de soluções, trazendo valor acrescentado para os seus clientes.

1.2. OBJETIVOS

O objetivo principal desta tese é desenvolver uma aplicação *web* de assistência remota com realidade aumentada, para auxílio no processo de triagem e na resolução de problemas entre especialistas remotos e clientes ou operadores no terreno. Face à complexidade intrínseca deste objetivo, foi necessário dividi-lo em múltiplos objetivos.

O primeiro objetivo, antes do início do desenvolvimento da aplicação, é conhecer as diversas soluções existentes, quer académicas, quer proprietárias. Em específico, protocolos de *streaming*, que permitam a realização de videochamadas entre utilizadores. Além disso, é necessário analisar as bibliotecas de RA que permitam a adição de ancoras numa aplicação *web*.

O objetivo seguinte é desenvolver uma aplicação *web* multiplataforma, a qual deve ter as seguintes funcionalidades:

- A definição de sala de reuniões, sendo permitida a entrada de apenas dois utilizadores, o especialista remoto e o operador no terreno.
- Envio de convite para aceder à sala através de URL, de forma a simplificar o processo de convite e a permitir, apenas, a entrada de pessoas com convite.
- Estabelecimento de uma videochamada após a entrada de ambos os utilizadores, para dar início à assistência remota.
- A partilha de ecrã durante a videochamada por parte do especialista remoto, facilitando o processo de resolução de problemas.
- A adição de anotações de realidade aumentada ancoradas em tempo real pelo especialista remoto no vídeo do operador no terreno, simplificando o processo de triagem e a resolução do problema.

O objetivo final é que a aplicação desenvolvida seja responsiva, isto é que seja capaz de se adaptar a diversos tipos de dispositivos, como computadores, *tablets* ou telemóveis, priorizando a usabilidade, garantindo uma ótima experiência ao utilizador.

1.3. ORGANIZAÇÃO DO RELATÓRIO

Este relatório de tese está organizado em 6 capítulos. No capítulo 1 é feita a introdução ao projeto, sendo contextualizado e apresentados os objetivos. No capítulo 2 é caracterizado o problema, através da apresentação dos requisitos do cliente, das soluções existentes, analisados os diversos protocolos de *streaming* e duas das bibliotecas de realidade aumentada. No capítulo 3 é especificada a solução, através da arquitetura e casos de uso. No capítulo seguinte, 4, é descrita a solução desenvolvida, quer as funcionalidades, como a sua interface gráfica. Quanto ao capítulo 5, são executados os testes de sistema, que englobam os testes dos casos de uso, a definição de métricas e a análise dos resultados. No último capítulo, o 6, são reunidas as principais conclusões da tese e perspetivadas futuras implementações.

2. CARACTERIZAÇÃO DO PROBLEMA

Nesta secção é feita uma caracterização do problema apresentado pela Accenture na sua proposta, a qual inclui um estágio curricular onde o objetivo é desenvolver uma aplicação *web* que permita assistência remota. Esta secção é composta por quatro subsecções, a primeira onde se identificam os requisitos do cliente, a segunda responsável por apresentar as soluções que existem para resolver o problema do cliente com base numa pesquisa extensa, a terceira subsecção enumera os protocolos de transporte e, por fim, a quarta revela os protocolos de *streaming* em voga na atualidade.

2.1. IDENTIFICAÇÃO DOS REQUISITOS DO CLIENTE

Esta subsecção é responsável pela identificação dos requisitos do cliente. De forma a esclarecer o objetivo da aplicação *web* a desenvolver foram especificados alguns requisitos aquando do início do projeto.

Neste projeto o principal objetivo é desenvolver uma aplicação *web* de assistência remota. Com esse propósito foram identificados alguns requisitos pelo cliente, que são apresentados nos parágrafos seguintes.

A aplicação *web* será desenvolvida com o intuito de ser utilizada via *browser*, permitindo o uso em diversos dispositivos, sistemas operativos e *browsers*, e inibindo a necessidade de *downloads* e instalação adicional de *software*. Esta aplicação tem de ser desenvolvida usando Node.js, como tal o JavaScript será a linguagem utilizada, tanto no *back-end* como no *front-end*. Sendo uma aplicação *web*, pode, em caso de necessidade, ser embebida em páginas *web*, nomeadamente, na solução de *call center* do cliente.

De forma, a facilitar a assistência remota, a aplicação terá de permitir a execução de videochamadas em tempo real entre dois utilizadores, mais concretamente, entre uma pessoa que precisa de ajuda e uma pessoa que presta auxílio remotamente, por exemplo, entre um cliente ou operador no terreno e um especialista remoto.

Para simplificar o processo de iniciação da videochamada, será necessário gerar um convite, através de um endereço *Uniform Resource Locator* (URL), que permitirá a entrada somente à pessoa que receber o convite.

Por fim, o último requisito, será a integração da realidade aumentada na aplicação, facultando ao especialista remoto novas ferramentas que auxiliam no processo de assistência, uma vez que facilitam a compreensão e a execução por parte do cliente. Um exemplo desta integração, será permitir ao especialista remoto adicionar anotações de realidade aumentada, que estão ancoradas a um ponto de interesse, em tempo real no vídeo captado pelo operador no terreno.

2.2. SOLUÇÕES EXISTENTES

Esta subsecção apresenta algumas das aplicações existentes para resolver o problema supramencionado, quer através de soluções académicas, quer de soluções proprietárias, isto é, desenvolvidas por empresas. As soluções apresentadas de seguida foram encontradas após uma pesquisa extensa, recorrendo a artigos, dissertações e *websites*.

Primeiramente, a pesquisa prendeu-se com a procura de aplicações similares que resolvam o problema, mais concretamente, aplicações que permitam a assistência remota e que permitam a utilização da realidade aumentada como funcionalidade extra. Desta forma, após a pesquisa, verificou-se a existência de aplicações com estas características, que são utilizadas com o intuito de solucionar os problemas de assistência técnica.

2.2.1. SOLUÇÕES ACADÉMICAS DE ASSISTÊNCIA REMOTA COM REALIDADE AUMENTADA

Neste subtópico serão apresentadas algumas das soluções encontradas nos diversos artigos e dissertações analisadas com o intuito de descobrir o estado da arte atual sobre este assunto e, consequentemente, de que forma deveria ser desenvolvida a solução desta tese. Para tal, de seguida, é apresentado o modo de funcionamento, arquitetura e protocolo de *streaming*, das soluções encontradas.

Um dos artigos analisados tem como título *An Augmented Reality-Based Method for Remote Collaborative Real-Time Assistance: from a System Perspective* [1]. Neste artigo, encontramos uma problemática idêntica à desta tese. A solução apresentada inclui uma aplicação *web*, com servidor Node.js, utilização de WebSockets e um módulo de comunicação de vídeo em tempo real via WebRTC, o que permite aos especialistas remotos observar o problema do ponto de vista do operador, que é a pessoa que necessita de assistência. Em segundo lugar, a aplicação contém um quadro virtual transparente desenvolvido via *canvas*, que permite aos especialistas remotos dar assistência visual, através de anotações de RA, como desenhos ou texto. Por último, anotações são exibidas num dispositivo de RA, como o Microsoft HoloLens [1].

A Figura 1 revela um exemplo de arquitetura de uma solução de assistência remota em tempo real.

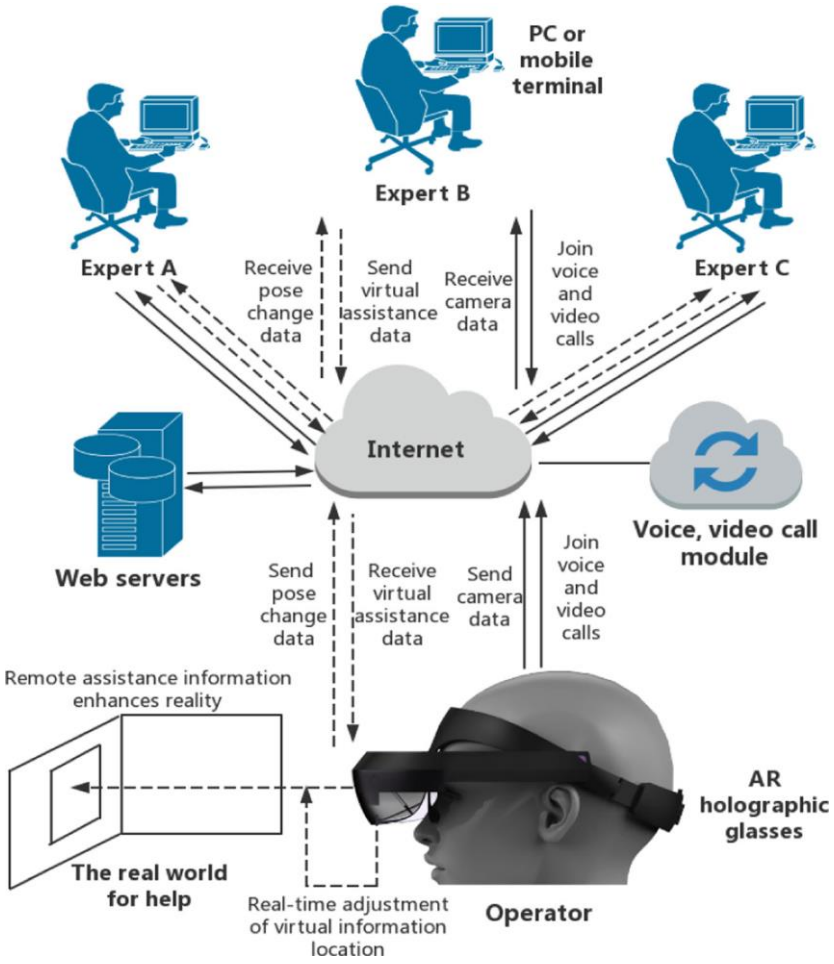


Figura 1 Arquitetura de uma Solução de Assistência Remota em Tempo Real [1]

Analisando a Figura 1, verifica-se um exemplo de arquitetura no âmbito destas aplicações, na qual é possível evidenciar a existência de diferentes módulos inter-relacionados.

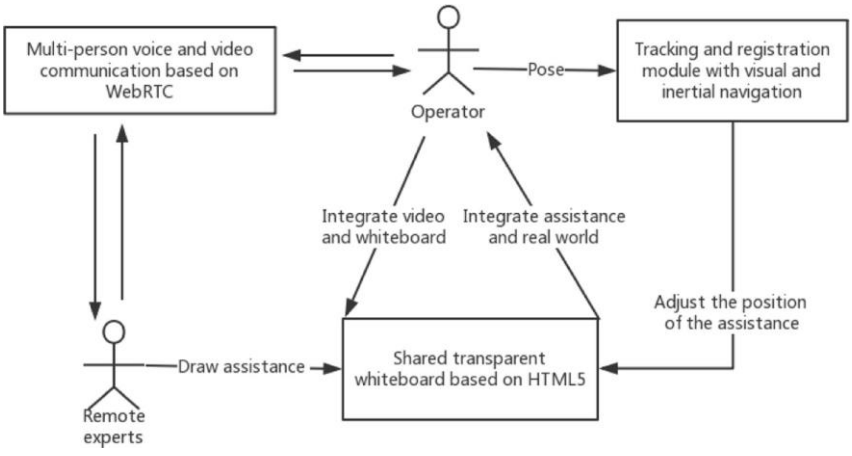


Figura 2 Módulos que Constituem o Sistema da Solução [1]

Na Figura 2, destacam-se os três módulos da solução proposta neste artigo. O módulo da chamada de áudio e vídeo via WebRTC, o módulo do quadro transparente baseado em *Hypertext Markup Language 5* (HTML5) através de um elemento *canvas*, e, por fim, o módulo que rastreia a postura do operador com base em sensores visuais e inerciais [1].

O artigo *Tailorable Remote Assistance with RemoteAssistKit: A Study of and Design Response to Remote Assistance in the Manufacturing Industry* [2], também faz referência a uma solução semelhante, na medida em que utilizam Node.js, WebSockets, WebRTC e um dispositivo de RA. A solução desenvolvida traduz-se numa aplicação *web*, em JavaScript, com servidor Node.js, que pode ser usada via *browser* num *desktop* pelo especialista ou num dispositivo móvel pelo operador. As diferenças, face à solução anterior, é que neste caso, existe um módulo de localização de orientação para a realidade aumentada, utilizando um dispositivo de RA, como o HoloLens, que é utilizado em conjunto e alinhado com o dispositivo móvel do operador, permitindo visualizar as anotações de RA. Em suma, o operador necessita de um dispositivo móvel e de um HoloLens para receber assistência com recurso a realidade aumentada. Além disso, para comunicar os diversos eventos durante a assistência, são utilizados WebSockets e o servidor Node.js, que também é responsável pela sinalização e configuração do *streaming* de vídeo via WebRTC [2].

Outra solução similar foi encontrada no artigo *Distance Learning and Assistance Using Smart Glasses* [3], onde utilizam uma *web app*, servidor Node.js e WebRTC. No lado do operador, a abordagem difere na medida em que o operador usa *smart glasses*, neste caso os Vuzix M100, através de uma aplicação Android desenvolvida [3].

Noutro artigo, sob o tema *Creating an Open-Source Augmented Reality Remote Support Tool for Industry: Challenges and Learnings* [4], são apresentados na secção *Related Work*, os diferentes tipos de anotações que podemos encontrar atualmente neste tipo de aplicações. Além disso, tal como no artigo anterior, é exposta uma solução de desenvolvimento de uma aplicação de assistência remota com realidade aumentada.

Normalmente, as anotações são realizadas pelo especialista remoto diretamente sobre a *stream* de vídeo. É importante salientar dois tipos de anotações, as anotações não ancoradas e as ancoradas. Nas aplicações em que as anotações não estão ancoradas, isto

é, não estão “presas” a uma âncora ou objeto, é necessário remover as anotações, assim que o ponto de vista do vídeo muda, uma vez que, a partir desse momento tornam-se inválidas. Este tipo de anotação, pontual, sem informação adicional, deve ser acompanhado por comunicação por voz. Esta abordagem de anotação é útil para instruções rápidas, orientando o cliente/operador no local para determinado objeto [4].

Quanto às anotações ancoradas, estas têm uma validade estendida e, desde que o *tracking* de realidade aumentada seja bom, as anotações permanecem na mesma posição independentemente do ponto de vista. Neste tipo de anotações, recorre-se, normalmente, a desenhos 2D, através de *free drawings* ou desenhos livres, texto e setas em vez de se colocar apenas pontos. Somente as anotações ancoradas podem ser consideradas anotações de RA [4].

Existe, também, a possibilidade de desenhar as anotações sobre *printscreens* em vez de sobre o vídeo. Neste caso, o especialista tem a possibilidade de tirar *printscreen* ao vídeo no momento oportuno, adicionar anotações e enviar a imagem anotada para o operador. Neste caso, as anotações são fixas, não sendo anotações de realidade aumentada [4].

A apresentada neste artigo resume-se a uma aplicação *mobile* utilizada pelo operador no local e a uma aplicação *desktop* utilizada pelo especialista remoto. Ambas as aplicações foram desenvolvidas recorrendo ao *software* Unity, o *framework* ARFoundation para a RA e, por fim, a tecnologia WebRTC para estabelecer a comunicação entre os dispositivos. As anotações podem ser efetuadas no vídeo ou em *printscreens* e ser armazenadas numa galeria. A Figura 3 revela a arquitetura genérica da solução deste artigo [4].

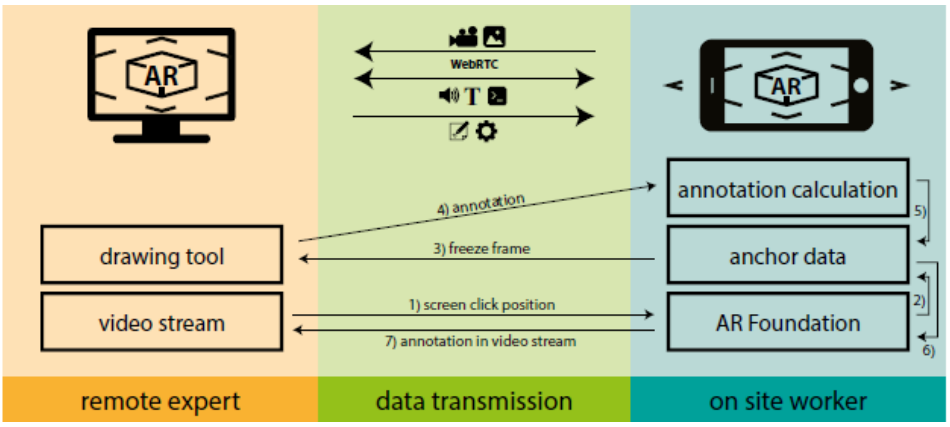


Figura 3 Arquitetura do Sistema que Utiliza *Software* Unity [4]

No artigo *Maintenance with Augmented Reality Remote Support in Comparison to Paper-Based Instructions: Experiment and Analysis* [5], é apresentada uma solução semelhante à anterior. Recorrendo ao Unity para o desenvolvimento da aplicação, ao ARFoundation para a ancoragem das anotações e o *tracking* do ambiente e o WebRTC para a comunicação de vídeo e áudio entre o operador e o especialista remoto [5].

Em termos de conclusão, nesta subsecção foi possível analisar algumas das soluções propostas academicamente para esta problemática. No entanto, nenhuma se adequa na totalidade aos requisitos e objetivos desta dissertação, uma vez que em termos da implementação da realidade aumentada, a mesma é feita recorrendo a dispositivos de RA, através do desenvolvimento de *software* para o efeito ou através do *software* Unity. Estas soluções não podem ser utilizadas neste projeto pois é imperativo que tanto o operador no terreno como o especialista utilizem apenas a aplicação *web* a desenvolver, sem recorrer a outros dispositivos. É importante salientar que algumas das soluções são aplicações *web*, com servidor Node.js, protocolo de *streaming* WebRTC, como tal estas soluções serão tidas em conta para a proposta de solução e desenvolvimento da aplicação desta Tese.

2.2.2. Soluções Proprietárias de Assistência Remota com Realidade Aumentada

Na atualidade, já várias empresas desenvolveram soluções para esta problemática. Algumas das aplicações com este propósito são a Cisco Webex Expert on Demand da Cisco [6], a Dynamics 365 Remote Assist da Microsoft [9], a IBM Augmented Remote Assist da IBM [12] ou a Vuforia Chalk da PTC [14]. Estas aplicações diferem entre si, permitindo o uso de diferentes dispositivos, entre eles, dispositivos de realidade aumentada, como o HoloLens da Microsoft ou o RealWear HMT-1 da Cisco, dispositivos *mobile*, sejam telemóveis ou *tablets*, e computadores através de aplicações nativas ou via *browser*. De seguida, são apresentadas as aplicações mencionadas, nomeadamente, o seu modo de funcionamento, a arquitetura e o protocolo de *streaming* utilizado.

A respeito da Cisco Webex Expert on Demand foi desenvolvida com o intuito de ser utilizada em dispositivos de realidade aumentada, como o RealWear HMT-1, em que o técnico utiliza este dispositivo no local da assistência e, simultaneamente, inicia uma videochamada com um especialista remoto que use aplicação Cisco Webex Teams, existindo integração entre aplicações, habilitando funcionalidades complementares, como a partilha de ficheiros e anotações do especialista [6].

A transmissão do vídeo e áudio é feita recorrendo à tecnologia WebRTC, que permite comunicações seguras em tempo real. A arquitetura genérica desta solução da Cisco é ilustrada na Figura 4 [7].

Figura 4 Arquitetura da Solução Cisco Webex Expert on Demand [8]

No caso da aplicação Dynamics 365 Remote Assist da Microsoft, a utilização é similar à da Cisco, uma vez que pode ser utilizada através de dispositivos de realidade mista, como o HoloLens, mas possibilita a utilização em dispositivos *mobile*, através de aplicações nativas, quer para iOS, quer para Android [9].

Nas aplicações *mobile* nativas são utilizados *kits* de desenvolvimento de *software*, que permitem adicionar experiências de realidade aumentada em dispositivos compatíveis, por exemplo, em Android é utilizado o ARCore, enquanto em iOS é utilizado o ARKit [10].

Tal como na solução da Cisco, na Dynamics 365 o técnico que utiliza a aplicação de assistência remota entrará em contacto com o especialista remoto, sendo que este faz uso da aplicação Microsoft Teams, existindo assim integração entre aplicações, providenciando várias funcionalidades que simplificam a resolução do problema. A transmissão do vídeo e áudio também é feita recorrendo à tecnologia WebRTC [9][11].

A solução IBM Augmented Remote Assist é utilizada através de duas interfaces, sendo que uma interface é a aplicação *mobile* iOS, usada pelo técnico ou cliente no terreno, e a outra é uma aplicação *web*, via *browser*, utilizada pelo especialista remoto, no entanto, o especialista remoto também pode usar a aplicação iOS para prestar a assistência [12].

No que concerne à transmissão de dados entre as aplicações, é usada a tecnologia WebRTC. A Figura 5 demonstra como é estabelecida a conexão entre um utilizador da aplicação *mobile* e um assistente remoto que utiliza a aplicação *web*. Primeiramente, as aplicações ligam-se a um servidor de sinalização e comunicam utilizando o protocolo *WebSocket over TLS* (WSS), de forma a trocarem endereços IP e portas, para, posteriormente, se iniciar a *stream* de áudio e vídeo com base na tecnologia WebRTC, caso sejam dadas as permissões [13].

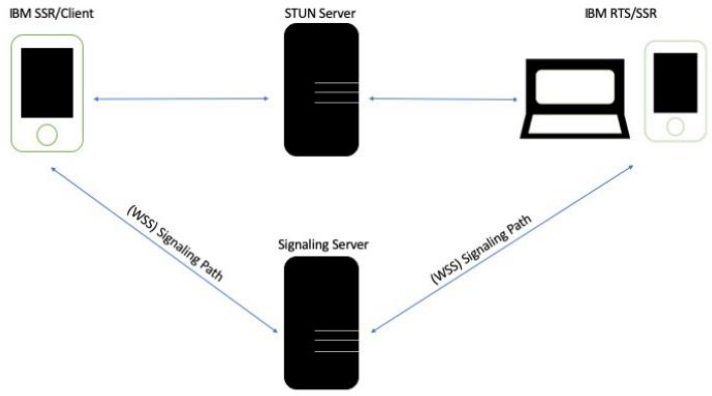


Figura 5 Processo de Sinalização da Solução IBM Augmented Remote Assist [8]

A ligação será estabelecida, sempre que possível, em *peer-to-peer* (P2P). Caso contrário, a conexão é feita usando um servidor *Transversal Using Relays around* NAT (TURN), como apresenta a Figura 6 [13].

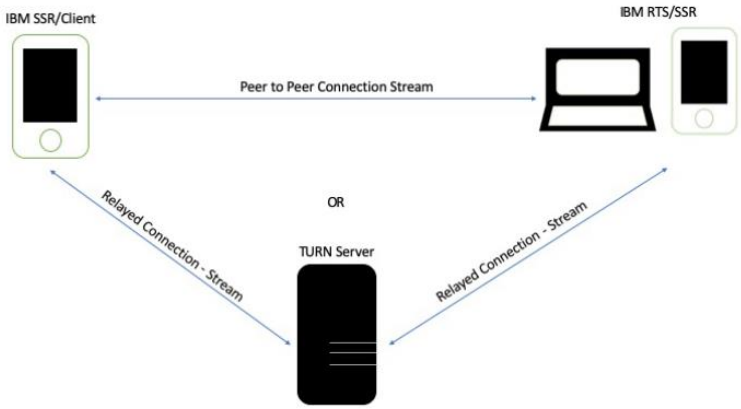


Figura 6 Tipos de Conexão da Solução IBM Augmented Remote Assist [13]

Quanto à solução Vuforia Chalk, esta contempla aplicações *mobile* nativas, tanto para iOS, como para Android. Apresenta, também, compatibilidade com o dispositivo de realidade aumentada RealWear HMT-1 da Cisco e dá a possibilidade ao especialista remoto de utilizar esta solução, além das aplicações *mobile*, através da aplicação *web browser*, evitando a necessidade de instalação de uma aplicação ou *plugin*, mas apenas com *browsers* compatíveis, neste caso, o Google Chrome ou o Microsoft Edge com base Chromium [14].

A solução Vuforia Chalk, tal como as soluções anteriores, faz uso da tecnologia WebRTC para a transmissão dos dados entre os utilizadores das suas aplicações. A Figura 7 revela a arquitetura de alto nível da solução Vuforia Chalk, onde se verifica que, no processo de sinalização, são utilizados os Chalk Cloud Services. Este processo estabelece *streams* de dados *peer-to-peer*, que permitem a entrega de dados encriptados, usando padrões WebRTC como os protocolos *Datagram Transport Layer Security* (DTLS) e *Secure Real-time Transport Protocol* (SRTP) [15].

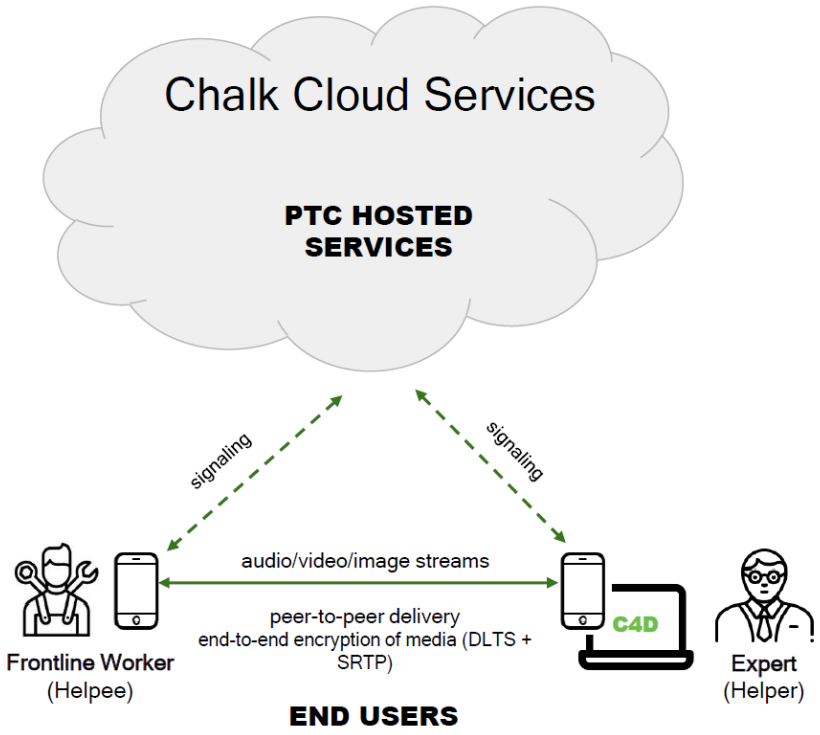


Figura 7 Arquitetura de Alto Nível da Solução Vuforia Chalk [15]

As soluções apresentadas são muitos similares, nomeadamente, nas funcionalidades e na arquitetura, visto que utilizam o protocolo WebRTC para o *streaming* dos dados, existindo algumas diferenças nos dispositivos compatíveis e no modo de funcionamento.

Em suma, estas soluções providenciam aos operadores no terreno aplicações *mobile* nativas ou aplicações para dispositivos de RA, sendo que algumas soluções oferecem a possibilidade ao especialista remoto de prestar a assistência via *browser*. Tal como anteriormente mencionado, um dos objetivos desta dissertação é desenvolver uma aplicação *web* de assistência remota, que seja utilizada via *browser*, quer pelo utilizador que necessita de assistência, quer por quem presta a assistência remotamente.

Assim conclui-se que, apesar das soluções estudadas permitirem solucionar problemas de assistência remota com possibilidade de recurso à realidade aumentada, nenhuma pode ser usada, somente, via *browser* e, além disso, são soluções proprietárias. Desta forma é justificado o desenvolvimento de uma nova solução, que na sua arquitetura deverá contar com a tecnologia WebRTC, presente nas soluções estudadas, devido às suas características, que são abordadas em seguida na secção dos protocolos de *streaming*.

2.3. PROTOCOLOS DE TRANSPORTE

Neste subtópico são descritos os protocolos de transporte, que, por sua vez, são utilizados pelos protocolos de *streaming*, possibilitando compreender a sua importância e justificar a escolha do protocolo de transporte a utilizar na aplicação a desenvolver.

Segundo o modelo *Open System Interconnection* (OSI), as redes são compostas por sete camadas com diversas funções. Uma das camadas mais importantes no âmbito de *streaming*, mais concretamente, nas aplicações de videochamadas, é a camada de transporte que é responsável pela transmissão dos dados, salientando-se dois protocolos, o *Transmission Control Protocol* (TCP) e o *User Datagram Protocol* (UDP).

O protocolo UDP é mais adequado para aplicações que requerem baixa latência como soluções de *Voice over Internet Protocol* (VoIP), jogos *online* e aplicações de *streaming* de vídeo em tempo real. Contudo, este protocolo é considerado não confiável. Não garante a entrega dos pacotes, podendo existir a perda, replicação ou a entrega não sequencial. Neste tipo de aplicações não é estritamente necessário receber todos os pacotes, a pontualidade na receção dos dados é a prioridade [17][18].

Por sua vez o TCP, é um protocolo confiável, fazendo uso do mecanismo *three-way handshake* apresentado na Figura 8.

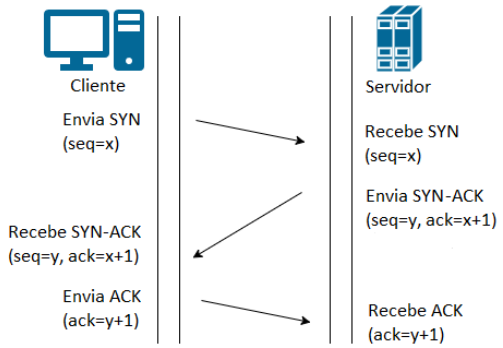


Figura 8 Mecanismo *Three-way Handshake* [19]

Neste mecanismo é estabelecida uma conexão, por exemplo, entre um cliente e um servidor, em que o cliente envia uma mensagem de sincronização (SYN) que contém o respetivo número de sequência (x), o qual permite identificar o pacote. Após a receção da mensagem, o servidor responde com uma mensagem (SYN-ACK), que contém o seu próprio número de sequência (y) e o número de confirmação (x + 1). Por fim, o cliente responde com uma mensagem de confirmação (ACK), com o número (y + 1) que, ao ser recebida pelo servidor, lhe permite conferir a receção da mensagem enviada anteriormente. Este mecanismo permite a entrega de todos os pacotes, possibilitando ótima qualidade de vídeo. No entanto, a entrega dos pacotes é mais demorada e, por este motivo, o protocolo TCP não é ideal para o uso em aplicações de tempo real [17][18].

2.4. PROTOCOLOS DE *STREAMING*

Após a pesquisa das aplicações existentes de assistência remota com RA, foi iniciada a recolha de informações sobre os protocolos de *streaming* existentes, uma vez que é necessário determinar qual é o mais adequado para implementar o mecanismo de videochamada e, posteriormente, a integração da RA. Neste subtópico apresenta-se a definição e as funcionalidades dos principais protocolos de *streaming*, realizada uma comparação e, por fim, é escolhido o protocolo para resolver a problemática.

Um protocolo de *streaming* pode ser caracterizado como um método que permite a entrega de dados através da *web*, sejam eles áudio, vídeo, texto ou outro. Mais especificamente, é um conjunto de regras que estipulam como os dados são transferidos e como lidar com os erros durante o processo [16].

No que toca aos principais protocolos utilizados no âmbito do *streaming* de vídeo, são de evidenciar, o *Real Time Messaging Protocol* (RTMP), o *HTTP Live Streaming* (HLS), o *Moving Pictures Expert Group - Dynamic Adaptive Streaming over HTTP* (MPEG-DASH) e o *Web Real-Time Communications* (WebRTC). A Figura 9 resume os protocolos mais utilizados em 2019, segundo o estudo "2019 *Video Streaming Latency Report*" realizado pela Wowza Media Systems, que é a maior empresa de soluções de *streaming* de vídeo [20].

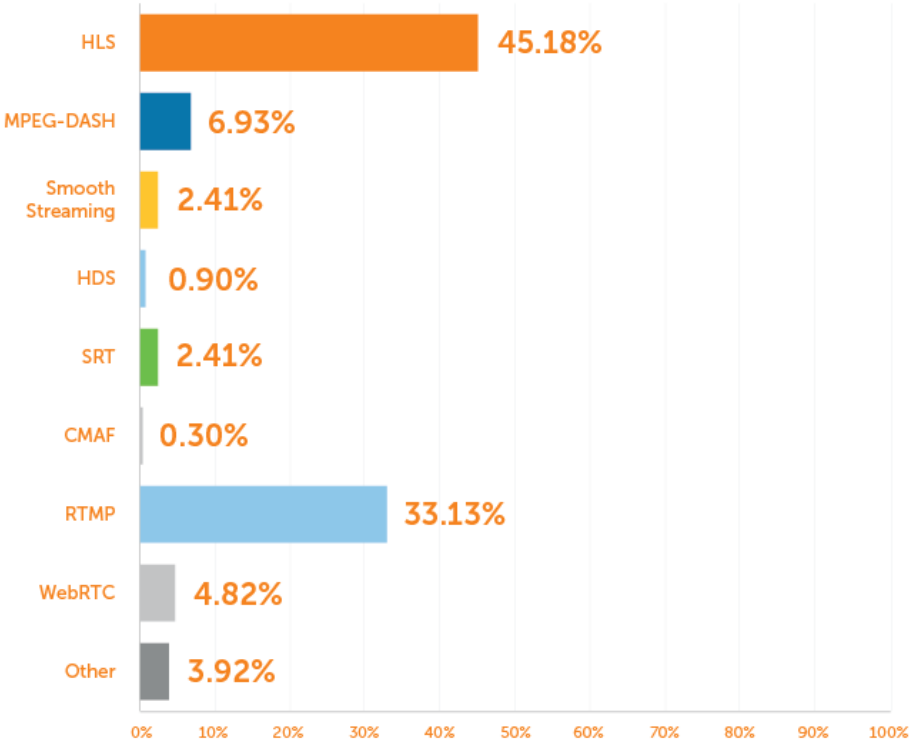


Figura 9 Utilização dos Protocolos de *Streaming* em 2019 [20]

Segundo a Figura 9, verifica-se que o protocolo mais utilizado é o HLS, com cerca de 45 %, seguido do RTMP com 33 %, do MPEG-DASH com 7 % e do WebRTC com 5 %. Com o fim de vida do Flash Player no final de 2020 anunciado pela Adobe em julho de 2017, o HLS rapidamente se tornou no protocolo de *streaming* preferencial, uma vez que o protocolo mais utilizado até então, o RTMP, faz uso da tecnologia Flash [20].

2.4.1. RTMP

Aquando do surgimento do *streaming*, o RTMP era o protocolo padrão para o transporte de vídeo e áudio pela Internet. Inicialmente, o RTMP era um protocolo proprietário, desenvolvido nos anos 90 pela Macromedia, que foi adquirida pela Adobe, a qual tornou o RTMP um protocolo aberto. Este protocolo é caracterizado por utilizar o protocolo de transporte TCP e foi desenhado para manter uma conexão estável e persistente, prevenindo a ocorrência de perdas de pacotes, com latências baixas, a rondar os cinco segundos, o que permite uma ótima experiência de *streaming* [21][22].

O RMTP é responsável por realizar a conexão entre um cliente com um reprodutor Flash instalado e um servidor, tal como demonstra a Figura 10.

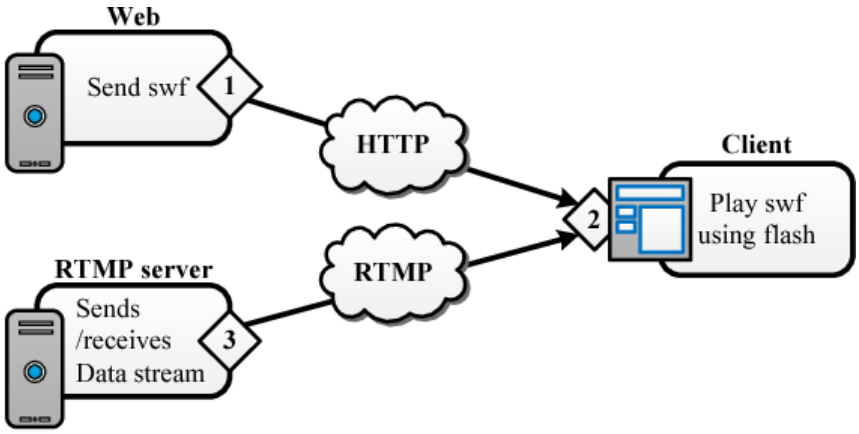


Figura 10 Mecanismo de Comunicação do RTMP [23]

Na Figura 10 mostra-se que o cliente recebe uma página com Flash de um servidor *web*. Este cliente deverá ter instalado uma aplicação Flash, como o Adobe Flash Player, para comunicar em tempo real com o servidor RTMP e reproduzir o *streaming* desejado.

O RTMP em conjunto com o Flash Player foram os mecanismos de entrega dominantes para *streaming* até o início de 2010. Contudo, com o aparecimento do *streaming* de vídeo HTML5, novos protocolos abertos e a velocidade de transmissão adaptável ultrapassaram o RTMP no processo de entrega ao utilizador final. O RTMP sempre teve problemas como ultrapassar *firewalls* e a necessidade de um servidor de *streaming* dedicado. Contudo, o maior entrave deste protocolo, surgiu após a Adobe ter anunciado o fim do suporte do Flash Player em 2020 e ter recomendado a alteração dos serviços para novos protocolos [22][24].

No entanto, mesmo com os problemas descritos, o RTMP em 2019, era usado em 33 % dos casos, isto deve-se ao facto do RTMP ser maioritariamente usado na atualidade para fazer a codificação e o transporte do áudio e vídeo para os servidores dos distribuidores de conteúdos. Quanto à descodificação e entrega do *streaming* ao utilizador final são utilizados protocolos baseados em HTML, como o HLS e o MPEG-DASH [22].

2.4.2. HLS

Hoje em dia, a indústria começou a adotar protocolos baseados em HTTP porque oferecem a melhor qualidade de vídeo e experiência de visualização possível, sendo o HLS e o MPEG-DASH os dois protocolos baseados em HTTP mais comuns [25].

O HLS é um protocolo de *streaming* baseado em HTTP, que foi desenvolvido pela Apple e lançado em 2009, usado para transportar vídeo e áudio de servidores para o ecrã dos utilizadores, quer para transmissões ao vivo quer previamente gravadas [25][26].

Os componentes do protocolo HLS estão identificados na Figura 11.

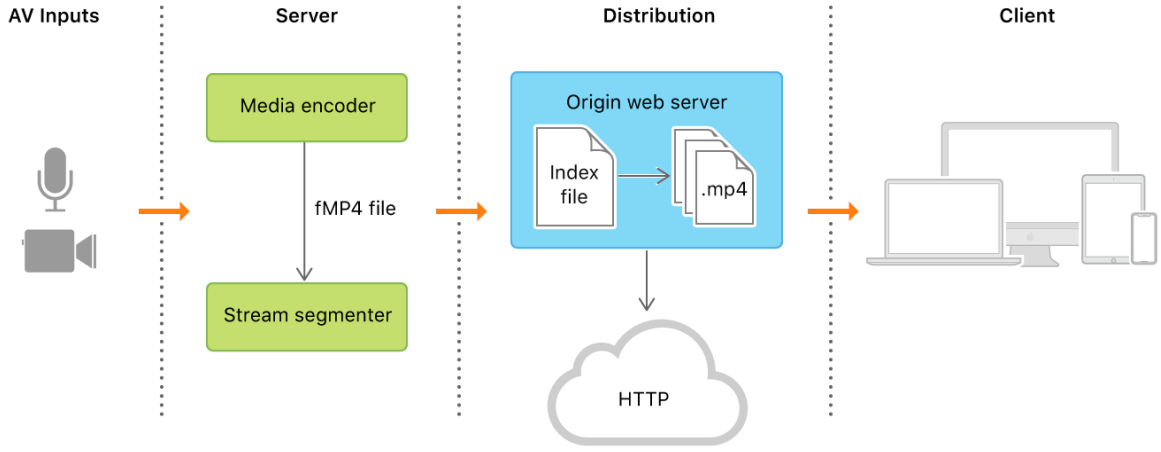


Figura 11 Componentes do Protocolo HLS [27]

O HLS divide-se em três componentes: o servidor, a distribuição e o *software* do cliente. Numa configuração típica, um codificador de *hardware* é responsável por codificar as entradas de áudio e vídeo e produzir um ficheiro MPEG-4 fragmentado ou uma *stream* MPEG-2. Seguidamente, um *software* de segmentação divide o ficheiro ou a *stream* em vários ficheiros de *media* curtos, que são colocados num servidor *web*. Além disso, este

software também cria e mantém um ficheiro de índice contendo uma lista dos ficheiros de *media*. O URL do ficheiro de índice é partilhado no servidor *web*. O *software* do cliente lê o índice, solicita os ficheiros de *media* listados de forma ordenada e mostra-os sem pausas ou intervalos entre os segmentos [27].

A popularidade do HLS face às alternativas pode ser atribuída à qualidade da experiência obtida através do método de entrega *adaptive bitrate streaming*, isto é, da velocidade de transmissão adaptável. A transmissão HLS adapta-se dinamicamente quer à largura de faixa disponível, quer às características do dispositivo de visualização do espectador, como a sua resolução máxima. Outra vantagem do HLS é a compatibilidade de reprodução, uma vez que é compatível com os diversos sistemas operativos existentes [25][26].

Quanto à escalabilidade, o HLS pode transportar os dados utilizando servidores pertencentes a uma *Content Delivery Network* (CDN), isto é, a uma Rede de Entrega de Conteúdos. Uma CDN permite obter cobertura global e uma enorme escalabilidade, reduzir os tempos de carregamento, poupar largura de faixa, lidar com picos de visualizações e melhorar a experiência do utilizador através do armazenamento de segmentos de vídeo e áudio em *cache* [25][28].

Contudo, o HLS apresenta uma latência elevada na transmissão, a rondar os trinta segundos. Para fazer face a este problema, em 2019 a Apple anunciou o lançamento da extensão *Low-Latency* HLS, que possibilita latências na ordem dos três segundos. Apesar desta melhoria significativa, esta latência impossibilita a utilização deste protocolo em aplicações de tempo real, que necessitam de latências inferiores a um segundo, como é o caso das aplicações de videoconferência [25].

2.4.3. MPEG-DASH

O protocolo MPEG-DASH é um protocolo de *streaming* baseado em HTTP que tem como função o transporte de *media* de servidores *web* para o utilizador final, sendo o protocolo alternativo ao HLS [29].

Este protocolo apresenta algumas características similares ao HLS, tal como a latência de cerca de trinta segundos. Além disso, também faz uso do método de transmissão de *bits* com uma taxa adaptável, permitindo a adaptação do vídeo apresentado face a realidades de diferentes espectadores. Faz uso dos servidores *web* que pertencem a Redes de Entrega de Conteúdo para distribuir os ficheiros de *streaming*. Algumas das maiores empresas de conteúdo *streaming* utilizam este protocolo nos seus serviços, por exemplo, a Netflix, a Hulu e o YouTube [29].

A grande diferença entre o HLS e o MPEG-DASH prende-se no facto do HLS ser especificado pela Apple, enquanto o MPEG-DASH é um protocolo aberto. Por este motivo, os dispositivos Apple suportam nativamente o protocolo HLS, uma vez que a Apple prioriza este protocolo face à solução de código aberto MPEG-DASH, o qual só é reproduzido nesses dispositivos recorrendo a reprodutores de vídeo HTML5 [29].

Além desta diferença, o HLS e o MPEG-DASH também se diferenciam nos *codecs* suportados e no método de entrega de baixa latência. Quanto aos *codecs*, o HLS especifica os *codecs* de vídeo e áudio que suporta, enquanto o MPEG-DASH é *codec-agnostic*, isto significa que lhe é indiferente o *codec* em uso, o que pode proporcionar transmissões de qualidade superior usando taxas de transmissão de *bits* inferiores, devido ao uso de *codecs* mais avançados. Quanto ao método de entrega de baixa latência, como vimos anteriormente, o HLS faz uso da extensão *Low-Latency* HLS, enquanto o MPEG-DASH usa o *Common Media Application Format* (CMAF), designado por *Low-Latency* CMAF, permitindo uma latência na ordem dos três segundos [29].

Concluindo, o MPEG-DASH oferece inúmeros benefícios por ser um protocolo aberto e ser desenvolvido por uma forte comunidade. Esta colaboração contraria a fragmentação existente na área, promovendo o compromisso de melhorar a interoperabilidade e eliminar a complexidade, através do uso de protocolos de *streaming* abertos [29].

2.4.4. WEBRTC

O WebRTC pode ser considerado mais que um protocolo *streaming*. É um projeto *open-source*, inicialmente desenvolvido pela *World Wide Web Consortium* (W3C) e, posteriormente, pelo *Internet Engineering Task Force* (IETF), contando atualmente com o suporte da Apple, Google, Microsoft e Mozilla. O WebRTC permite aos *browsers* e aplicações *mobile* nativas, a realização de comunicações em tempo real, através de API de JavaScript gratuitas e do HTML5 [30].

Esta tecnologia consegue aceder de forma segura à câmara e ao microfone de qualquer sistema, possibilitando a troca de informações *peer-to-peer* em tempo real entre dois *browsers*, não havendo necessidade de *download* ou instalação de *plugins*. Além disso, sendo P2P, elimina a necessidade de servidores intermediários para a transmissão dos dados. Em suma, o WebRTC permite a execução de chamadas de voz, vídeo e a partilha P2P, que inclui, por exemplo, a partilha do ecrã e de ficheiros [31].

Relativamente à arquitetura do WebRTC, esta divide-se em duas API, a WebRTC C++ API e a Web API, tal como é ilustrado na Figura 12. A WebRTC C++ API é utilizada pelos programadores que desenvolvem *web browsers*. Enquanto Web API, é utilizada para desenvolver aplicações *web* com as funcionalidades do WebRTC, e tem como base três API, denominadas RTCPeerConnection, MediaStream e RTCDataChannel.

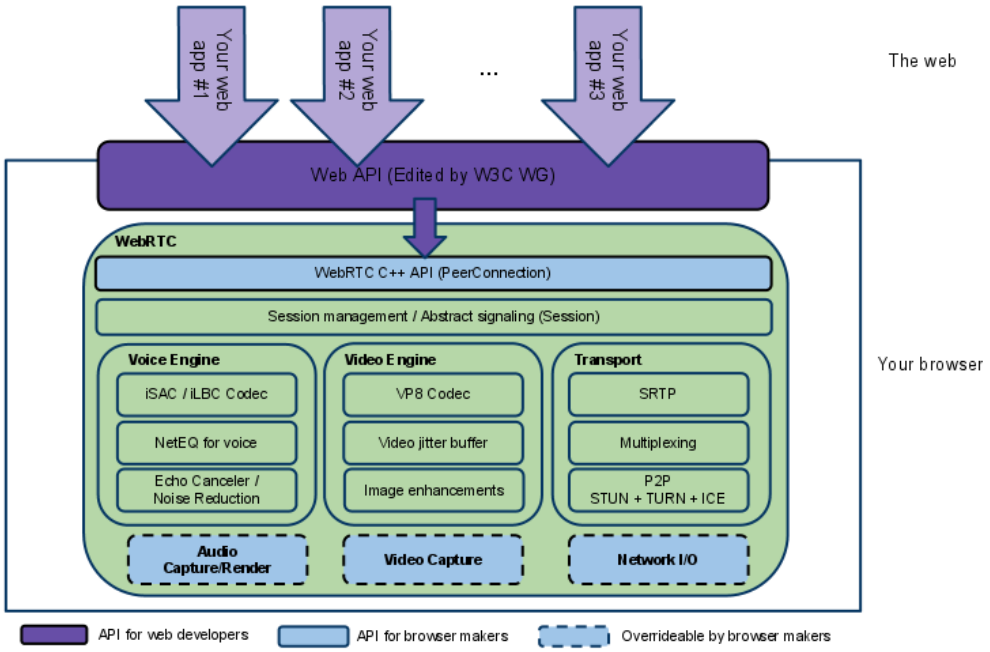


Figura 12 Arquitetura da tecnologia WebRTC [33]

A API RTCPeerConnection é responsável por estabelecer a comunicação *peer-to-peer* tão característica do WebRTC. Para estabelecer esta ligação é necessário um canal de sinalização, que pode ser implementado através de WebSockets. Este canal utiliza o protocolo *Interactive Connectivity Establishment* (ICE) que permite ao *browser* compreender a topologia da rede e encontrar a forma ideal de estabelecer a conexão, podendo ser utilizados servidores *Session Traversal Utilities for* NAT (STUN) e *Transversal Using Relays around* NAT (TURN), de modo a permitir que as *streams* de *media* ultrapassem *Network Address Translation* (NAT) *gateways* e *firewalls* [31].

A API MediaStream fornece os meios para controlar uma *stream* de áudio ou vídeo, através de vários métodos, como o *getusermedia()*, que permite aceder à câmara ou ao microfone, pedindo permissão à priori para o acesso, conferindo segurança ao sistema. O *output* de um objeto MediaStream pode ser mostrado em elementos HTML como, *video* e *audio*, ou ser enviado para um *peer* remoto através de um objeto RTCPeerConnection. A transmissão de *streams* recorre a protocolos como o *Secure Real-time Transport Protocol* (SRTP) ou o *Real-time Transport Protocol* (RTP) e, por fim, para monitorizar a transmissão é utilizado o *Real-time Transport Control Protocol* (RTCP) [34][35].

Quanto ao RTCDataChannel, é um canal de dados bidirecional para ligações P2P. Recorre ao protocolo *Secure Control Transmission Protocol* (SCTP), encapsulado em *Datagram Transport Layer Security* (DTLS), para realizar a transferência dos dados, aumentando a eficiência de transporte e segurança dos dados [31].

A Figura 13 resume a pilha protocolar do WebRTC, permitindo identificar os diferentes protocolos em uso e a relação que têm entre si.

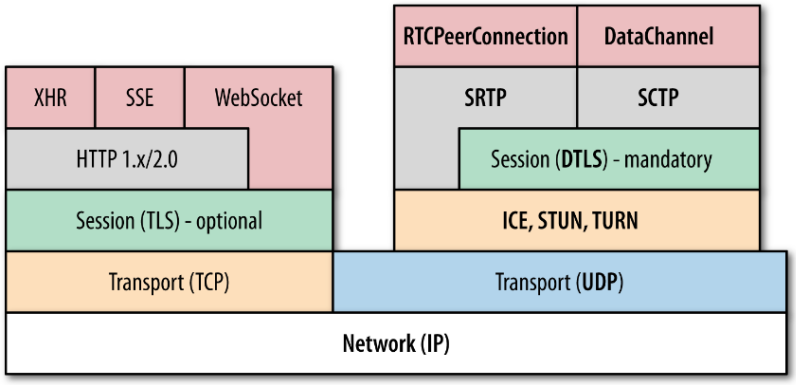


Figura 13 Pilha Protocolar do WebRTC [36]

Em termos de benefícios, o WebRTC, apresenta uma latência inferior a 500 milissegundos, a menor latência entre todos os protocolos de *streaming*, permitindo comunicações em tempo real. Além disso, o facto de ser uma tecnologia de código aberto e padronizada, permite eliminar problemas de interoperabilidade, sendo compatível com diversos sistemas operativos, *browsers* e dispositivos. Em termos da compatibilidade com *codecs* de áudio e vídeo de alta qualidade, trabalha com o *codec* de áudio Opus e o de vídeo VP9, estando previsto o suporte para o mais recente *codec* de vídeo AV1. Por fim, tem a capacidade de se adaptar às condições da rede dos utilizadores, através da técnica Simulcast, que codifica a *stream* de vídeo com diferentes resoluções e *bitrates*. Ao contrário da taxa de transmissão de *bits* adaptável, que podemos encontrar no HLS ou MPEG-DASH, onde a qualidade do *streaming* é ajustada durante a reprodução, no WebRTC a qualidade do vídeo é adaptada logo no processamento do vídeo, na sua codificação [32].

Tendo em conta o que esta tecnologia permite, são inúmeras as suas aplicações. Nomeadamente, no uso pessoal, para conviver com a família e os amigos, em qualquer parte do mundo, e no uso empresarial, como por exemplo, para efetuar reuniões, vender produtos ou até prestar suporte, como é o objetivo da solução a desenvolver.

As grandes empresas do ramo, como a Apple, a Google, a Microsoft e a Mozilla tiveram em conta as potencialidades e aplicações do WebRTC e decidiram suportar esta tecnologia, desenvolvendo *browsers* que suportam o WebRTC. Este desenvolvimento tem sido contínuo. Atualmente, a compatibilidade com *browsers desktop* pode ser verificada na Figura 14 e com *browsers mobile* na Figura 15.

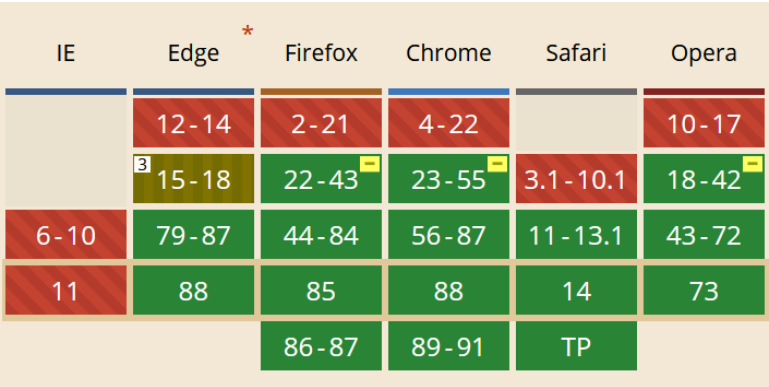


Figura 14 Compatibilidade do WebRTC nos *Browsers Desktop* em 2021 [37]

Analisando a Figura 14, verifica-se que, atualmente, todos os principais *browsers*, suportam a tecnologia WebRTC, à exceção do antigo e descontinuado Internet Explorer. É importante referir que os números apresentados com o fundo a vermelho, referem-se a versões, que não têm suporte WebRTC.

Contudo, existem alguns desafios que são mais evidentes no Safari e, também, melhorar a compatibilidade dos *codecs* do Safari, especificamente, permitir a utilização do *codec* de vídeo VP9 [38].

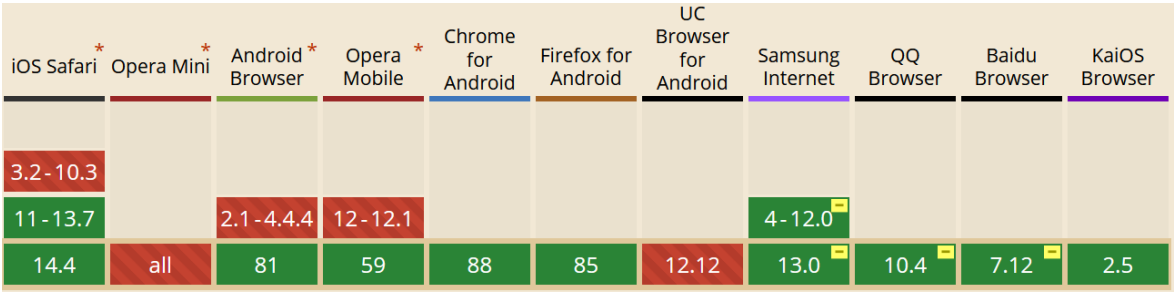


Figura 15 Compatibilidade do WebRTC nos *Browsers Mobile* em 2021 [37]

Analisando Figura 15, depreende-se que o WebRTC é compatível com praticamente todos os *browsers mobile*, exceto a versão do Opera Mini e o UC Browser.

Nos dispositivos iOS apenas o *browser* iOS Safari tem suporte WebRTC, contudo apresenta inúmeros *bugs* que podem colocar em causa o funcionamento da aplicação desenvolvida. Este comportamento é um motivo para que o uso do WebRTC em dispositivos iOS ocorra através de aplicações nativas. Porém, é de salientar, que estes *bugs* têm vindo a ser corrigidos nas últimas versões do iOS Safari, o que demonstra a importância que a Apple dá ao WebRTC [38].

Quanto aos *browsers* em Android, é aconselhado o uso dos principais *browsers*, como o Chrome ou Firefox, uma vez que os restantes, apesar de suportarem o WebRTC, podem usar implementações mais antigas, gerando problemas desnecessários [38].

Após a análise de cada um dos protocolos, segue-se uma abordagem comparativa. A Tabela 1 estabelece a comparação dos principais protocolos de *streaming* em uso, tendo em conta as vantagens e desvantagens, de forma a sintetizar e a justificar a escolha do protocolo a utilizar no desenvolvimento da Tese.

Tabela 1 Comparação entre Protocolos *Streaming* analisados

	RTMP	HLS	MPEG-DASH	WebRTC
Vantagens	Estabilidade Codificação e Transporte Eficiente	Escalabilidade Elevada (Servidores CDN) Alta Compatibilidade Taxa de Transmissão de *Bits* Adaptável	Protocolo Aberto Escalabilidade Elevada (Servidores CDN) Boa Compatibilidade Taxa de Transmissão de *Bits* Adaptável *Codec-agnostic*	Protocolo aberto Alta Compatibilidade Comunicações P2P Inexistência de *Plugins* Suporta *Codecs* de Alta Qualidade Latência (<500 ms)
Desvantagens	Necessidade de Servidor Dedicado Dependência de *Plugins* Fim do Suporte ao Flash Player Latência (5 s)	Protocolo Proprietário Suporte Limitado de *Codecs* Latência (3 s a 30 s)	Latência (3 s a 30 s)	Ainda em Desenvolvimento

Recorrendo à análise da Tabela 1, verifica-se que somente o WebRTC é adequado para a realização de comunicações em tempo real. Segundo a *International Telecommunication Union-Telecommunication* (ITU-T), que é uma entidade responsável por definir padronizações relacionadas com a área das telecomunicações, recomenda, na norma ITU G.1010, latências inferiores a 400 milissegundos para comunicações em tempo real.

As videochamadas encontram-se nessa categoria porque são caracterizadas pela troca bidirecional de voz e vídeo, necessitando, consequentemente, de uma latência muito reduzida para evitar constrangimentos na comunicação entre os utilizadores. O WebRTC é o único com a capacidade de originar latências inferiores a 500 milissegundos, sendo, por isso, a única opção viável para implementar na solução de assistência remota a desenvolver [39].

A baixa latência do WebRTC, o facto de ser um protocolo aberto e ter alta compatibilidade com *browsers* e sistemas operativos, satisfazendo, nomeadamente, o requisito da aplicação ser utilizada via *browser* e multiplataforma, são importantes vantagens do WebRTC..

Como o objetivo é desenvolver uma aplicação ou serviço em tempo real através do *browser*, a escolha recai para o WebRTC. É igualmente importante referir a existência de uma comunidade ativa de desenvolvimento do WebRTC, sendo um protocolo cada vez mais popular e adotado [40].

2.5. BIBLIOTECAS DE REALIDADE AUMENTADA

Nesta subsecção são apresentadas duas das bibliotecas existentes que permitem integrar a realidade aumentada em aplicações *web*, a WebXR [42] e a *Open Source Computer Vision Library* (OpenCV) [46].

2.5.1. WEBXR

A biblioteca WebXR permite o desenvolvimento de aplicações *web* de realidade aumentada e de realidade virtual. Sendo uma alternativa aos *kits* de desenvolvimento de RA, como o ARKit e o ARCore, que só podem ser utilizados em aplicações nativas [41].

A WebXR Device API é responsável por implementar as funcionalidades, sendo compatível com diversos dispositivos, nomeadamente, de realidade mista, como o HoloLens, que têm a capacidade de rastrear o movimento e a orientação, ou móveis, recorrendo às suas câmaras [42].

A WebXR API é composta por diversos módulos, sendo o mais relevante para este projeto, o módulo de realidade aumentada, designado por WebXR *Augmented Reality Module*. Este módulo permite adicionar as funcionalidades da WebXR API a dispositivos de RA. Para tal, é necessário verificar a compatibilidade do dispositivo com a sessão de realidade aumentada, "*immersive-ar" session*, criada pela WebXR API [43].

De forma a demonstrar a potencialidade desta API, é apresentada a Figura 16, um exemplo da utilização desta API numa aplicação *web* de realidade aumentada.

Figura 16 Aplicação webxr-ar-paint [44]

A Figura 16 é um *printscreen* da aplicação webxr-ar-paint, que foi desenvolvida utilizando a API WebXR e o *framework* Three.js. Esta aplicação permite adicionar anotações de realidade aumentada, nomeadamente, desenhos no vídeo captado por um telemóvel. Neste caso foi simulada uma assistência remota a um *router*, em que é adicionada uma seta no decorrer da assistência.

Esta aplicação é um bom exemplo do que é pretendido desenvolver na aplicação desta tese. No entanto, verificou-se que não seria possível implementar uma solução usando a biblioteca WebXR, uma vez que uma sessão de realidade aumentada WebXR não pode ser usada como *stream* e como tal, não é possível enviar para outro utilizador as anotações efetuadas no vídeo de uma sessão WebXR.

Além do problema mencionado no parágrafo anterior, é importante referir que a WebXR ainda se encontra em desenvolvimento, sendo considerada uma API instável, podendo ser alterada a qualquer momento, não sendo por isso recomendada a sua utilização em aplicações comerciais [43].

Em termos de compatibilidade, para além de ser necessário usar a aplicação num dispositivo compatível, é necessário usá-la num *browser* compatível. Atualmente, a WebXR Device API apresenta compatibilidade limitada com o Chrome, o Edge e o Samsung Internet e com o Firefox. No caso do Firefox, é necessário ativar através de uma *flag*, daí o aparecimento da bandeira verde e do número 2 na Figura 17, que demonstra a compatibilidade da WebXR Device API. Todos os casos em que a compatibilidade é limitada estão assinalados com o número 1 na Figura 17.

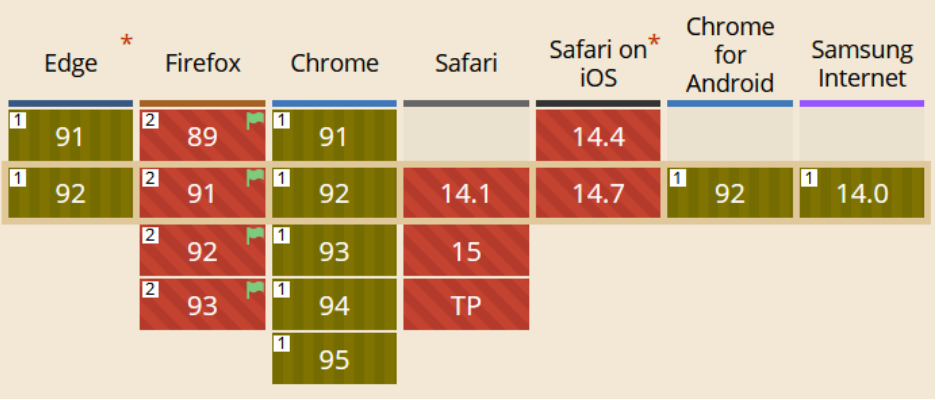


Figura 17 Compatibilidade da WebXR Device API com os Principais *Browsers* [45]

2.5.2. OPENCV

A OpenCV é uma biblioteca *open source*, multiplataforma e otimizada para aplicações de tempo real, nomeadamente, para processamento de imagens e vídeos, possuindo centenas de algoritmos de visão computacional na sua biblioteca [46].

A OpenCV está disponível em diversas linguagens de programação, como C++, Java, Python e JavaScript, que será a linguagem de programação a utilizar no desenvolvimento da aplicação *web*. Neste caso, será possível integrar esta biblioteca, recorrendo à OpenCV.js [47].

É importante evidenciar que esta biblioteca se encontra em desenvolvimento, sendo que atualmente a sua última versão estável é a 4.5.2. Além disso, apresenta uma comunidade enorme, existindo diversos manuais, tutoriais e fóruns para ajudar os programadores [48].

Esta biblioteca tem uma estrutura modular, sendo, de seguida, apresentados alguns dos seus principais módulos. O módulo das funcionalidades *core*, que é responsável pela estrutura de dados, como o *array* multidimensional Mat e as funções básicas utilizadas pelos restantes módulos. O módulo do processamento de imagem, com variados métodos, como histogramas, filtros lineares e não lineares de imagem ou a transformação geométrica. O módulo de deteção de objetos, capaz de detetar caras, pessoas, carros, entre outros. No entanto, o módulo de destaque para esta tese, é o módulo de análise de vídeo, que inclui, por exemplo, algoritmos de estimativa de movimento e de *tracking* de objetos [46].

A análise dos diversos tutoriais da OpenCV.js permitiu verificar as suas diferentes aplicações, das quais se salienta um exemplo na secção de análise de vídeo, designado Lucas-Kanade *Optical Flow*, que é apresentado na Figura 18 e na Figura 19. O *optical flow* ou fluxo ótico descreve o movimento aparente de objetos entre dois *frames* consecutivos causado pelo movimento dos objetos ou da câmara [49].

Este exemplo permite rastrear o movimento de determinados pontos num vídeo e desenhar o trajeto dos mesmos ao longo desse vídeo. Inicialmente, para definir os pontos a rastrear é utilizado o algoritmo de ShiTomasi, que permite detetar *corner points*, para serem rastreados pelo método Lucas-Kanade, uma vez que estes pontos são mais fáceis de identificar *frame* após *frame* [49].

Na Figura 18 e na Figura 19, encontra-se à esquerda o vídeo original e, à direita num *canvas*, o resultado do processamento efetuado pelo método de Lucas-Kanade. A Figura 18 evidencia a definição dos pontos a rastrear usando o algoritmo de ShiTomasi. A Figura 19 demonstra o deslocamento efetuado pelos pontos que estão a ser rastreados através do método Lucas-Kanade.

Figura 18 Exemplo Lucas-Kanade *Optical Flow* – Definição dos Pontos a Rastrear [49]

Figura 19 Exemplo Lucas-Kanade *Optical Flow* – Deslocamento dos Pontos a Rastrear [49]

Da análise do exemplo Lucas-Kanade *Optical Flow* conclui-se que provavelmente é possível utilizar este método para adicionar anotações de realidade aumentada ancoradas no vídeo do operador na aplicação a desenvolver. Para isso, os pontos a rastrear serão definidos pelo especialista remoto em vez de ser utilizado o algoritmo de ShiTomasi. Um problema que pode surgir é o especialista selecionar pontos difíceis de detetar pelo método Lucas-Kanade. Dado que na aplicação a desenvolver não é necessário desenhar o deslocamento dos pontos a rastrear, deve-se apenas desenhar, *frame* após *frame*, a nova posição dos pontos a rastrear. Desta forma, será possível ao operador no terreno perceber as instruções dadas através das anotações, mesmo que a câmara ou os objetos com os pontos a rastrear se movam.

3. ESPECIFICAÇÃO DA SOLUÇÃO

Nesta secção é especificada a solução para o problema proposto nesta Tese. Esta secção é composta por duas subsecções, a primeira onde se apresenta a arquitetura da aplicação a desenvolver, enumerando os diferentes módulos e as tecnologias a utilizar em cada um deles. Na segunda subsecção apresentam-se os diversos casos de uso a implementar nesta aplicação, de forma a satisfazer os requisitos do cliente.

3.1. ARQUITETURA

Nesta subsecção é exposta a arquitetura proposta para solucionar o problema. Para o efeito, é apresentada a Figura 20, que mostra a arquitetura geral da solução a desenvolver.

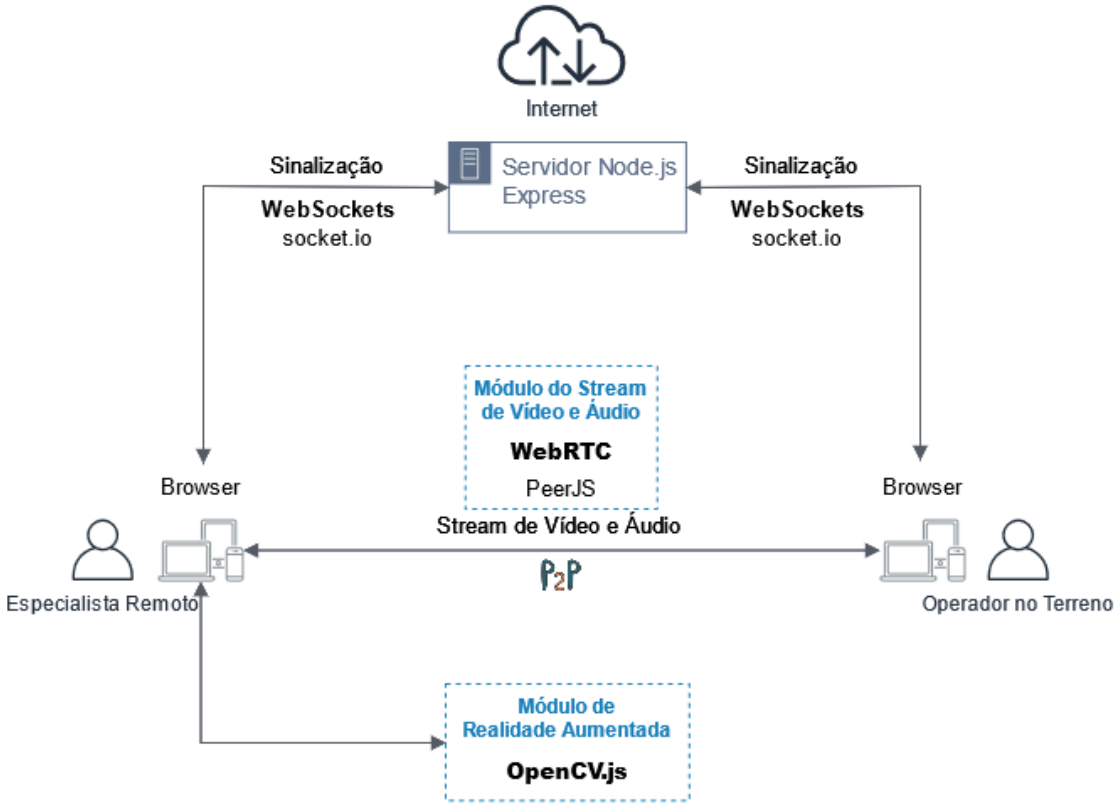


Figura 20 Arquitetura da Solução Proposta

Após analisar as diferentes soluções encontradas e de ter em conta os requisitos do cliente, foi proposta a arquitetura da Figura 20. Esta arquitetura revela algumas características da aplicação a desenvolver e as tecnologias que serão utilizadas.

Relativamente à aplicação, em termos de arquitetura de *back-end*, esta terá que usar obrigatoriamente Node.js por imposição do cliente, implicando que o servidor presente na arquitetura seja Node.js. De maneira a facilitar a criação da aplicação e do seu servidor será utilizada o *framework* Express. A aplicação *web*, será utilizada via *browser*, permitindo a utilização de diversos tipos de dispositivos, nomeadamente, móveis ou computadores, quer pelo especialista remoto, quer pelo operador no terreno.

No que toca ao processo de sinalização, este é feito no início da videochamada, onde o servidor receberá mensagens WebSocket dos clientes, através da biblioteca *socket.io*, que permite a comunicação bidirecional e em tempo real entre o cliente e o servidor, obtendo as informações necessárias para iniciar o *streaming* P2P de vídeo e áudio. Esta biblioteca é de fácil integração com a aplicação Node.js, utilizando o módulo *socket.io* presente no gestor de módulos *Node Package Manager* (NPM). Além de ser responsável pela sinalização, será responsável pela criação das salas de videochamada e pela comunicação de eventos entre os utilizadores da aplicação. Assim, será através desta biblioteca que serão comunicados eventos como entrada/saída de um utilizador da sala da videochamada, o desligar/ligar a câmara/micro, entre outros.

No módulo de *streaming* de vídeo e áudio, será utilizada a tecnologia WebRTC, de acordo com a análise efetuada na subsecção dos Protocolos de *Streaming*. Esta tecnologia será implementada através da biblioteca PeerJS, possibilitando a troca de dados P2P de vídeo e áudio e a integração na aplicação Node.js.

Por fim, o módulo responsável por introduzir a realidade aumentada na aplicação usará a biblioteca OpenCV.js. A OpenCV permite adicionar e rastrear âncoras ou pontos ao longo de um vídeo, utilizando algoritmos de visão computacional, como o método de Lucas-Kanade. Desta forma, será possível adicionar anotações de RA, isto é, anotações que se ajustam no vídeo caso o objeto anotado ou a câmara se mova.

A biblioteca OpenCV.js utiliza JavaScript e será implementada no *front-end* da aplicação, podendo ser utilizada apenas pelo especialista, sendo que o operador apenas vê o resultado do processamento efetuado pela OpenCV.js. Sucintamente, o especialista recebe a *stream* de vídeo do operador, utiliza esta *stream* como *input* para a OpenCV.js, que processa o vídeo, adicionando anotações de realidade aumentada, o *output* do processamento é apresentado num *canvas* do lado do especialista, que por sua vez envia por WebRTC o vídeo processado, presente no elemento *canvas*, para o operador.

3.2. CASOS DE USO

Nesta subsecção são apresentados de forma os casos de uso que estarão disponíveis na aplicação. A Figura 21 permite observar os diferentes atores e casos de uso da aplicação *web* de assistência remota a desenvolver.

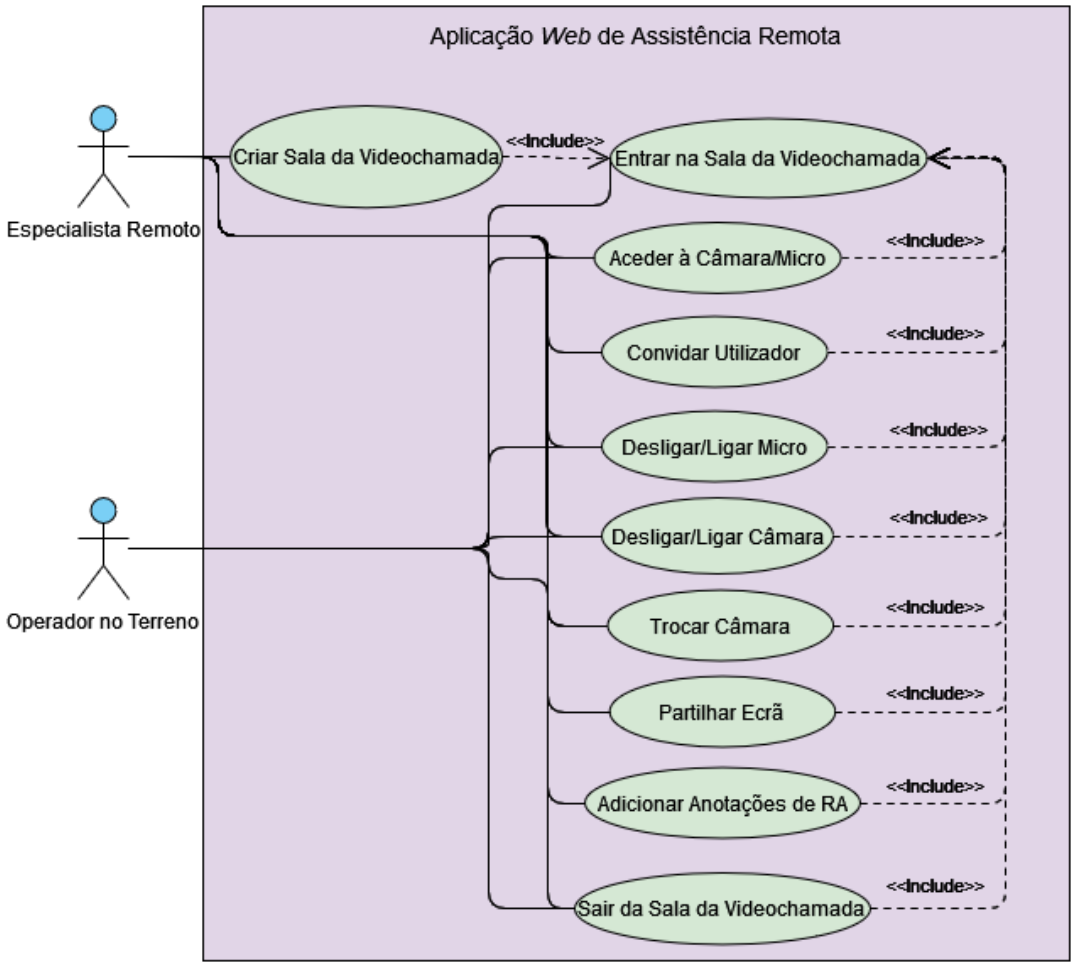


Figura 21 Casos de Uso da Aplicação *Web* de Assistência Remota

Analisando a Figura 21, verifica-se a existência de dois atores e dez casos de uso que serão parte integrante da aplicação a desenvolver. Desde logo, é necessário evidenciar os atores, sendo eles, o especialista remoto e o operador no terreno. O especialista será considerado administrador, tendo acesso a todas as funcionalidades e, por conseguinte, pode realizar todos os casos de uso existentes. Enquanto o operador no terreno, que é o utilizador que receberá a assistência, apenas terá acesso a parte das funcionalidades, mais concretamente as seguintes: Entrar na Sala da Videochamada, Aceder à Câmara/Micro, Desligar/Ligar Micro, Desligar/Ligar Câmara, Trocar Câmara e Sair da Sala da Videochamada.

O primeiro e principal caso de uso é Criar Sala da Videochamada. Sempre que um operador no terreno ou cliente necessitar de assistência poderá, através dos canais de comunicação pré-existentes, pedir ajuda a um especialista remoto. Este especialista terá a possibilidade de criar uma sala para a videochamada de assistência remota. Este caso de uso está associado ao caso de uso Entrar na Sala da Videochamada, uma vez que quando uma sala é criada pelo especialista remoto, ele entra automaticamente na sala criada.

Quanto ao caso de uso Entrar na Sala da Videochamada, este pode ser executado por ambos os atores da aplicação. Como mencionado no parágrafo anterior, o especialista entra automaticamente na sala no momento em que a cria. O operador no terreno necessita que o especialista lhe envie um convite para a sala da videochamada. Após a entrada dos dois atores, é iniciada a videochamada entre os dois utilizadores. Além disso, a sala não permite a entrada de mais nenhum utilizador, pois está limitada a um máximo de duas pessoas. É importante referir, que os restantes casos de uso só estão disponíveis para os utilizadores que estão na sala da videochamada.

Após a entrada na sala, deve estar disponível o caso de uso, Aceder à Câmara/Micro. Uma vez que esta aplicação tem como base uma assistência por videochamada, que necessita do acesso ao vídeo e áudio dos utilizadores, este caso de uso está disponível para ambos os utilizadores.

Em relação ao caso de uso Convidar Utilizador, este só pode ser executado pelo especialista remoto e está disponível logo que o especialista entre na sala da videochamada. Este caso de uso é de extrema importância porque permite enviar um convite através da partilha do endereço URL da videochamada, providenciando a respetiva entrada na sala da videochamada por parte do operador.

Os casos de uso Desligar/Ligar Micro e Desligar/Ligar Câmara têm características similares. Eles são caracterizados por permitirem desligar e ligar o microfone e a câmara no decorrer da assistência. Tanto o especialista quanto o operador têm acesso a ambos.

O caso de uso Trocar a Câmara é permitido a qualquer utilizador que aceda à aplicação *web* via dispositivo móvel, como um *tablet* ou um telemóvel, dando a possibilidade de trocar a câmara utilizada na videochamada, nomeadamente, entre a câmara frontal e a traseira do dispositivo sempre que o entenderem.

Quanto ao caso de uso Partilhar Ecrã, só o especialista tem permissão para o executar. Durante uma assistência remota, o especialista poderá partilhar alguma informação que esteja num documento e que seja facilmente acessível através da partilha do ecrã do especialista. Esta funcionalidade só é permitida se os utilizadores estiverem em videochamada.

Por ser o objetivo mais inovador desta Tese um dos mais importantes casos de uso é Adicionar Anotações de Realidade Aumentada. Esta funcionalidade só pode ser realizada pelo especialista remoto. Tal como a funcionalidade de partilhar ecrã, só é possível adicionar anotações se os utilizadores estiverem em videochamada, uma vez que o especialista fará anotações sobre o vídeo do operador.

Por último, o caso de uso de Sair da Sala da Videochamada pode ser realizado pelos dois atores, em qualquer momento.

4. IMPLEMENTAÇÃO

Nesta secção é descrita a implementação da solução para o problema proposto pelo cliente. Em primeiro lugar é abordado o desenvolvimento base da aplicação, nomeadamente os módulos do *Node Package Manager* (NPM) utilizados e as suas funções e, de seguida, é explicada a criação da aplicação. Esta secção é composta por duas subsecções, a primeira onde se esclarece, de forma sucinta, como foi efetuado o desenvolvimento das funcionalidades. Na segunda secção descreve-se o desenvolvimento da interface gráfica do utilizador, sendo explicado o seu desenvolvimento e apresentado o resultado final, quer para computador, como para dispositivos móveis.

O passo inicial no desenvolvimento da aplicação é implementar o seu *back-end*. Esta aplicação *web* assenta numa aplicação Node.js, permitindo utilizar a linguagem JavaScript, tanto no lado do servidor, como no lado do cliente.

Durante o desenvolvimento desta aplicação foram utilizados alguns módulos presentes no NPM, que permitem implementar os diferentes casos de uso. Como tal, de seguida são apresentados os módulos utilizados.

O primeiro módulo é o do *framework* Express, que permite o desenvolvimento de aplicações *web* de forma simples. Nomeadamente, oferece a possibilidade de, com poucas linhas de código, implementar um servidor [50].

O segundo é a biblioteca *socket.io*, que permite a comunicação bidirecional e em tempo real entre o cliente e o servidor, utilizado quer na sinalização, quer na criação de salas e emissão de eventos durante a videochamada [51].

O módulo seguinte é a biblioteca PeerJS, que permite criar o servidor *peerServer* que será integrado com o servidor Express. Esta biblioteca simplifica a implementação WebRTC para a troca de dados P2P de vídeo e áudio [52].

O quarto módulo é o *uuid*. Este módulo permite gerar um *universally unique identifier* (UUID) aleatório, que será utilizado para diferenciar as salas criadas [53].

Por fim, temos o módulo *ejs*. A *Embedded JavaScript templating* (EJS) é uma linguagem de representação simples, que permite gerar HTML com JavaScript, possibilitando a utilização de variáveis do *back-end* no *front-end* [54].

Verificou-se a necessidade de iniciar um servidor com *Hypertext Transfer Protocol Secure* (HTTPS), uma vez que, atualmente, os *browsers* só permitem aceder à câmara e ao micro do utilizador se as comunicações entre servidor e o *browser* forem seguras. Para tal, foi necessário gerar certificados *Secure Sockets Layer* (SSL), recorrendo à ferramenta OpenSSL, que permitem encriptar as comunicações entre o servidor e o *browser*.

De seguida, é apresentado um excerto de código, que é responsável pela criação da aplicação.

```
1  // Initializing Express
2  const express = require('express');
3  const app = express();
4
5  const fs = require('fs');
6  const https = require('https');
7  const options = {
8    key: fs.readFileSync('server-key.pem'),
9    cert: fs.readFileSync('server-cert.pem')
10 };
11
12 //Initialize https server and associate it with express
13 const httpsServer = https.createServer(options, app);
14
15 //Initialize socket.io For signalling in WebRTC
16 const io = require('socket.io')(httpsServer);
17 //Peer Server with express - HTTPS Server
18 const { ExpressPeerServer } = require('peer');
19 const peerServer = ExpressPeerServer(httpsServer, {
20   debug: true
21 });
22 app.set('view engine', 'ejs');
23 app.use(express.static('public'));
24 app.use('/peerjs', peerServer); // HTTPS
25 // HTTPS Server
26 httpsServer.listen(443 || process.env.PORT);
```

Verifica-se a iniciação do *framework* Express nas linhas 2 e 3. Já entre as linhas 7 e 10 são lidos os ficheiros da chave e certificado necessários para o HTTPS. Na linha 13 é iniciado o servidor e associado ao *framework* Express. Na linha 16 é iniciado o módulo *socket.io*, associando-o ao servidor criado. Por sua vez, entre as linhas 18 e 21 é iniciado o servidor *peerServer* e associado ao Express. Na linha 22 é definido o *view engine*, neste caso o módulo *ejs*. Na linha 23 dá-se o acesso aos ficheiros da pasta *public* e na linha 24 permite-se o uso do servidor *peerServer*. Por fim, na linha 26 define-se a porta do servidor.

4.1. DESENVOLVIMENTO DAS FUNCIONALIDADES

4.1.1. CRIAR E ENTRAR NA SALA DA VIDEOCHAMADA

De forma a desenvolver esta funcionalidade foi utilizado o módulo *uuid*, a biblioteca PeerJS e a biblioteca *socket.io*. Para facilitar a descrição e compreensão do desenvolvimento desta funcionalidade é apresentado o diagrama de sequência da Figura 22.

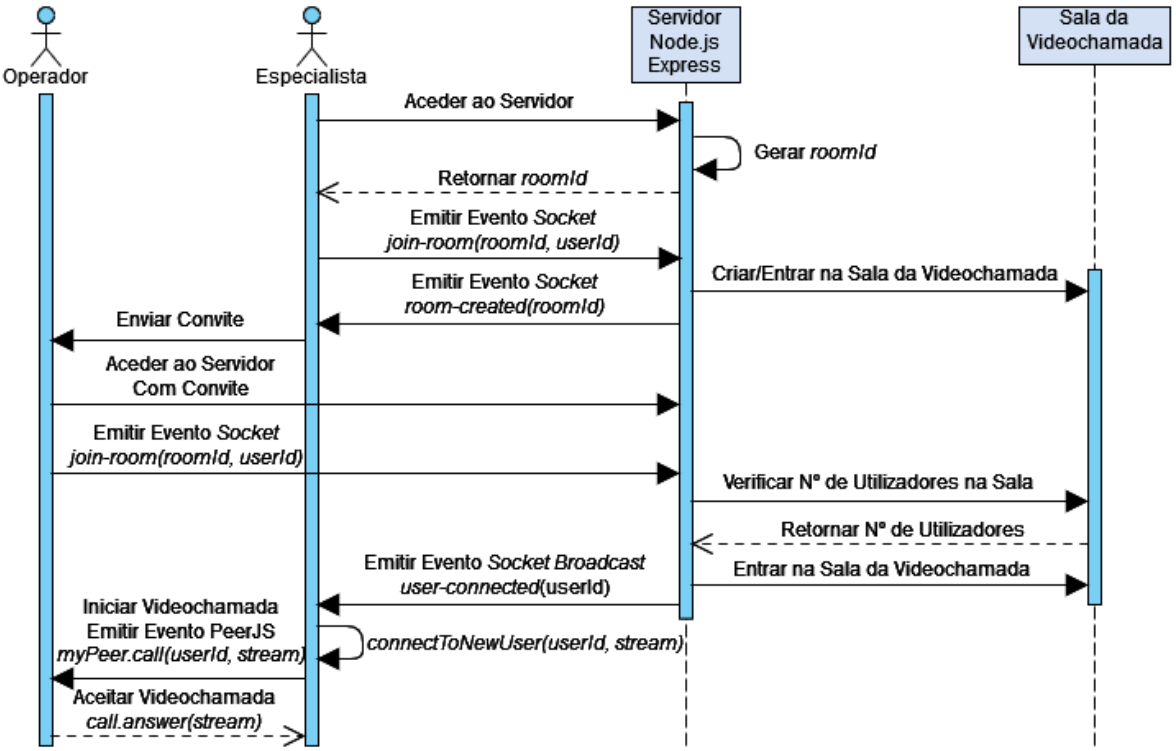


Figura 22 Diagrama de Sequência – Criar e Entrar na Videochamada

Para criar uma sala para a videochamada de assistência remota é necessário que o especialista remoto aceda ao servidor da aplicação, onde é gerado um *universally unique identifier* (UUID), designado por *roomId*, visto que será o identificador da sala de videochamadas a criar. Automaticamente, o especialista é reencaminhado para o endereço com esse identificador. Nesse momento, a biblioteca PeerJS deteta a conexão de um utilizador ao servidor *peerServer*, gera um identificador para esse utilizador, designado por *userId* e, consequentemente, é emitido um evento *socket* do tipo "*join-room*", passando como parâmetros o *roomId* e *o userId*.

Quando o especialista emite o evento *"join-room"* a sala é criada e o especialista entra na sala automaticamente, sendo emitido um evento *socket* do tipo "*room-created*" pelo servidor, com o parâmetro *roomId*, atribuindo ao especialista as permissões de administrador da sala. Quando um utilizador acede ao servidor com convite e emite o evento *"join-room"*, é verificada a quantidade de utilizadores presentes na sala com o identificador *roomId*. Caso esteja apenas um utilizador na sala, o utilizador entra na sala e são-lhe dadas as permissões de operador no terreno. Por fim, caso já estejam dois utilizadores na sala, é bloqueada a entrada de um novo utilizador.

Em relação ao processo de início da videochamada, esta só se inicia a partir do momento em que estão dois utilizadores na mesma sala. No exato momento em que entra o segundo utilizador é emitido um evento *socket* de *broadcast* do tipo "*user-connected*". Um evento *broadcast* é enviado para todos os utilizadores presentes naquela sala, excetuando o *sender*. Portanto, neste caso apenas o especialista remoto recebe este evento.

Do lado do cliente, sempre que um utilizador recebe um evento do tipo "*user-connected*", é executada a função *connectToNewUser()*, responsável por efetuar a videochamada usando a biblioteca PeerJS. Como consequência, o especialista remoto liga para o operador no terreno, passando como parâmetros o seu *id* de utilizador e a sua *stream* de vídeo e áudio. O operador no terreno, recebe a chamada através de um evento PeerJS, *myPeer.call()* e responde com a sua *stream* de vídeo e áudio. A partir deste momento é apresentada a *stream* recebida na interface gráfica, estabelecendo-se a videochamada.

4.1.2. ACEDER À CÂMARA/MICRO

Esta funcionalidade é imprescindível nesta aplicação de assistência remota que tem como base uma videochamada. Como tal, esta funcionalidade é executada automaticamente após a entrada de um utilizador na sala da videochamada, sendo iniciado o acesso à câmara e micro do utilizador para a obtenção da *stream* de vídeo e áudio.

No que toca ao desenvolvimento desta funcionalidade, foi necessário utilizar o método *getUserMedia()* da API *MediaDevices*, que solicita ao utilizador permissão para aceder à sua câmara e ao seu micro. Caso a permissão seja concedida, é definido um objeto *MediaStream* contendo a *stream* de vídeo e áudio do utilizador. Caso o utilizador não dê permissões ou a *media* pretendida não esteja disponível, são geradas as respetivas mensagens de erro, *NotAllowedError* ou *NotFoundError*. De forma a esclarecer o desenvolvimento desta funcionalidade é divulgado o excerto de código utilizado [55].

```
1  If(/Android|webOS|iPhone|iPad|iPod|BlackBerry|IEMobile|O
   pera Mini|SamsungBrowser/i.test(navigator.userAgent)){
2     var constraints = { audio: true, video: { facingMode:
   "user" } };
3  } else {
4     var constraints = { audio: true, video: true };
5  }
6  navigator.mediaDevices.getUserMedia(constraints)
7  .then(stream => {
8     addVideoStream(localVideoElement, stream);
9  })
10 .catch(function (err) {
11    console.log(err.name + ": " + err.message);
12 }); // Check Errors
```

No código anterior, entre a linha 1 e 5 é definida a variável *constraints* que depende do tipo de dispositivo utilizado, *mobile* ou não. Esta variável é responsável por selecionar o áudio e o vídeo. Caso seja *mobile*, é definida a propriedade *facingMode* como *user*, selecionando-se a câmara frontal. Na linha 6, é invocado o método *getUserMedia()*, com *constraints* como parâmetro. Se o utilizador der permissão, a *stream* de vídeo e áudio obtida será tratada pela função *addVideoStream()*, que irá adicionar esta *stream* à interface gráfica do utilizador (GUI). A partir da linha 10 encontra-se o código que tratará dos erros, gerando uma mensagem de erro na consola.

É de salientar que o método *getUserMedia()* permite definir as propriedades do vídeo, como a *facingMode*, responsável pela escolha entre a câmara frontal e traseira. Além disso, é possível definir a resolução do vídeo. No entanto, é mantida a resolução predefinida de 640x480 pois apresenta um bom equilíbrio entre a qualidade do vídeo e a largura de faixa. O mesmo sucede com a propriedade *frames* por segundo (FPS), com valor predefinido de 30 FPS.

4.1.3. CONVIDAR UTILIZADOR

Após a entrada do especialista remoto na sala da videochamada, este tem permissão para convidar um utilizador para, posteriormente, prestar a assistência remota. Para efetuar esta funcionalidade, basta que o especialista partilhe o endereço URL da página a que está ligado. Este endereço contém o UUID, que é o *id* da sala, previamente gerado. Em suma, ao partilhar este endereço, o destinatário conseguirá entrar diretamente na sala, caso estejam menos de duas pessoas na sala.

Na interface gráfica do utilizador, o especialista tem ao seu dispor um botão no menu das funcionalidades que permite efetuar o convite. Em *browsers* compatíveis com a *Web Share* API, é apresentado o mecanismo de partilha nativo do dispositivo, tal como é visível na Figura 23, usando o Google Chrome num computador com o sistema operativo Windows 10.

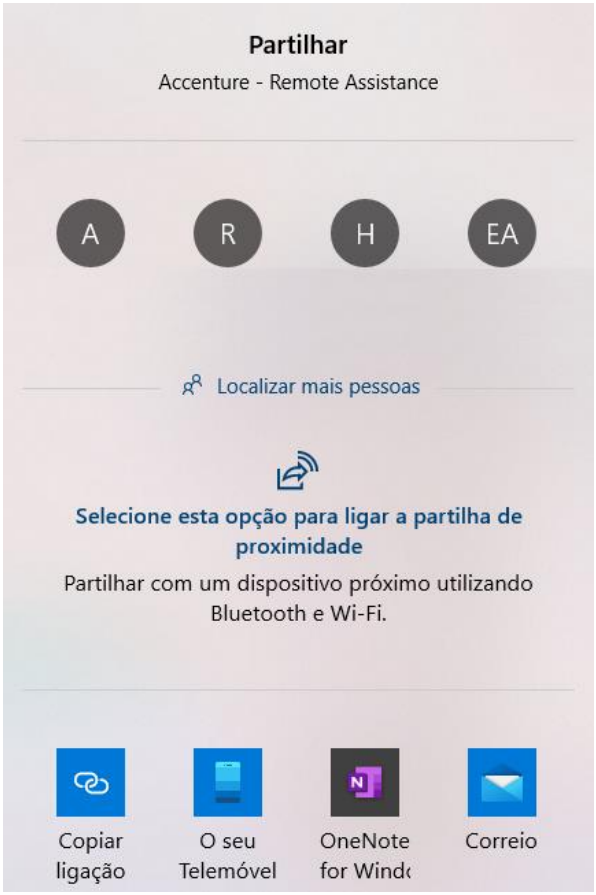


Figura 23 Convidar Utilizador – Google Chrome - Windows 10

Caso o *browser* não seja compatível, é apresentada uma caixa com o endereço, que pode ser copiado com apenas um clique no respetivo botão, como é apresentado na Figura 24. Neste exemplo, a aplicação estava a ser usada através do *browser* Mozilla Firefox num computador com Windows 10.

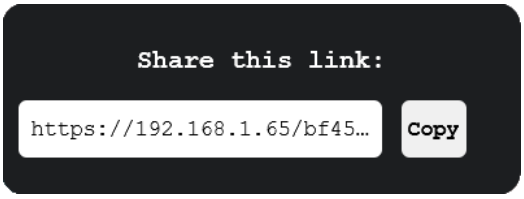


Figura 24 Convidar Utilizador – Mozilla Firefox – Windows 10

4.1.4. DESLIGAR/LIGAR MICRO

Esta funcionalidade é extremamente útil, permitindo ao utilizador da aplicação desligar e ligar o micro sempre que assim o entender. Para acionar esta funcionalidade o utilizador tem de se encontrar dentro de uma sala de videochamada. No *front-end* é apresentado um menu de funcionalidades que contém um botão para desligar e ligar o micro. Quando o utilizador pressiona esse botão, é chamada a função *microOnOff()*, apresentada no seguinte excerto de código.

```
1  function microOnOff(stream = myVideoStream) {
2      const enabled = stream.getAudioTracks()[0].enabled;
3      let html;
4      if (enabled) {
5          stream.getAudioTracks()[0].enabled = false;
6          // Emit Socket Event micro-off
7          socket.emit('micro-off', ROOM_ID, myUserId);
8          microOnOffButton.title = "Turn microphone on";
9          html = '<i class="fas fa-microphone-slash"
   style="color:red;"></i>';
10     } else {
11         stream.getAudioTracks()[0].enabled = true;
12         // Emit Socket Event micro-on
13         socket.emit('micro-on', ROOM_ID, myUserId);
14         microOnOffButton.title = "Turn microphone off";
15         html = '<i class="fas fa-microphone"></i>';
16     }
17     microOnOffButton.innerHTML = html;
18 }
```

Analisando o excerto anterior, verifica-se que na linha 2 utiliza-se o método *getAudioTracks()*, que confirma se a *stream* tem o áudio ativo ou não. De seguida, se o áudio estiver ativo, é executado o código entre as linhas 5 e 9. Na linha 5 é des25ativado o áudio, por consequência, a *stream* de *media* enviada via WebRTC deixa de enviar o áudio. Na linha 7, é emitido o evento *socket* do tipo "*micro-off*", que tem como parâmetros o *id* da sala e o *id* do utilizador que desligar o micro. Este evento será tratado pelo servidor. As linhas 8 e 9 são responsáveis por alterar o *front-end*, quer o título do botão, quer o ícone apresentado.

Se o áudio não estiver ativo, é executado o código entre as linhas 11 e 15. O código é similar ao explicado no parágrafo anterior, a diferença é que é ativado o áudio, emitido o evento *socket* do tipo "*micro-on*", com os mesmos parâmetros. Por fim, também é alterado o *layout* da interface gráfica do utilizador.

Os eventos *socket* emitidos são tratados pelo servidor. O servidor irá, por conseguinte, emitir outro evento *socket* de *broadcast* do tipo "*alert-micro-on*" ou "*alert-micro-off*", dependendo do evento que tenha recebido. Este evento de *broadcast* será enviado para o outro utilizador presente na respetiva sala, onde será tratado. Se o evento for "*alert-micro-off*" será apresentado um ícone de micro desligado por cima do vídeo do respetivo utilizador que desligou o micro. Caso contrário, se o evento for "*alert-micro-on*", significará que o micro foi ligado novamente e, consequentemente, será retirado o ícone de micro desligado.

A informação dos parágrafos anteriores é resumida pela Figura 25, que apresenta um diagrama de sequência exemplo do que acontece quando o especialista desliga o micro.

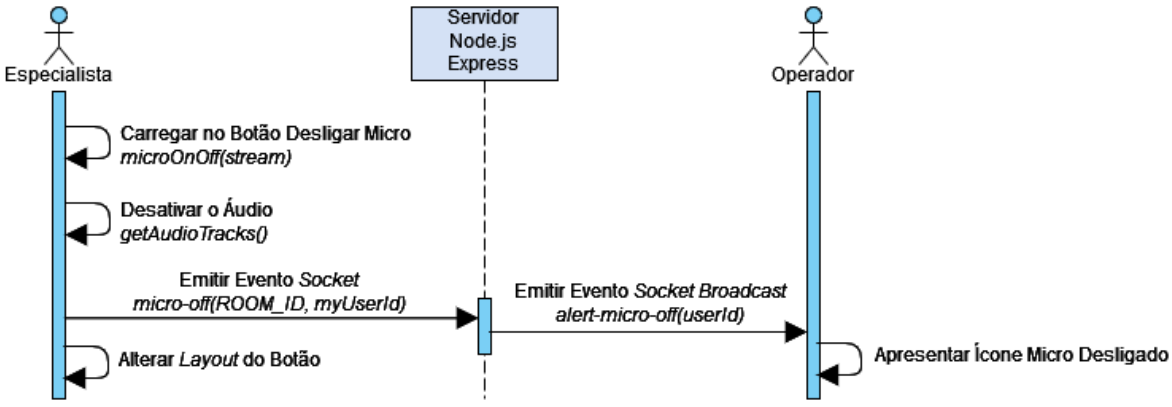


Figura 25 Diagrama de Sequência – Desligar Micro

4.1.5. DESLIGAR/LIGAR CÂMARA

Esta funcionalidade é similar à anterior, sendo que permite ao utilizador da aplicação desligar e ligar a câmara em vez do micro. De igual forma, o utilizador só pode acionar esta funcionalidade se estiver dentro de uma sala de videochamada, bastando premir o respetivo botão presente no menu das funcionalidades.

De forma a sintetizar o desenvolvimento da funcionalidade Desligar/Ligar Câmara é apresentada a Figura 26, que revela o diagrama de sequência para o exemplo do especialista desligar a câmara durante a assistência remota com o operador no terreno.

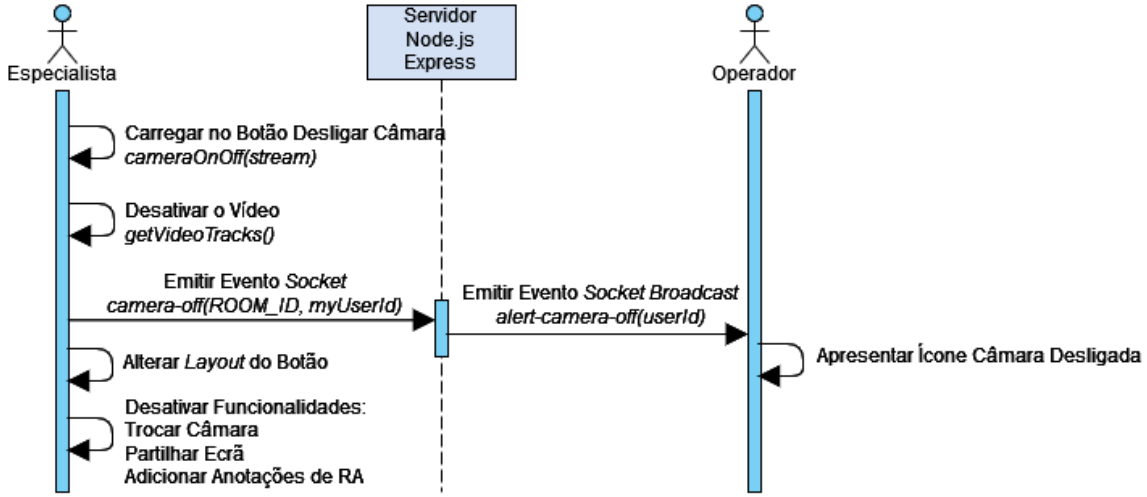


Figura 26 Diagrama de Sequência – Desligar Câmara

Com base na Figura 26, é possível verificar que esta funcionalidade foi desenvolvida de forma similar à funcionalidade Desligar/Ligar Micro. Neste exemplo, o especialista decide desligar a câmara durante a videochamada, premindo o botão presente para o efeito no menu das funcionalidades, que invoca a função *cameraOnOff()*.

Durante a execução desta função é desativado o vídeo do especialista, recorrendo ao método *getVideoTracks()*, deixando de ser enviado via WebRTC. Além disso, é emitido um evento *socket* do tipo "*camera-off*", com o *id* da sala e o *id* do utilizador em questão, o qual será tratado pelo servidor. O servidor, que por sua vez, emitirá um evento *socket* de *broadcast* do tipo "*alert-camera-off*". Sendo este tratado no lado do operador, permitindo a apresentação do ícone de câmara desligada no *front-end* da aplicação. No lado do especialista, também é modificado o *front-end*, nomeadamente, o *layout* do botão e a desativação das funcionalidades associadas à transmissão de *streams* de vídeo.

4.1.6. TROCAR CÂMARA

Esta funcionalidade foi desenvolvida a pensar nos utilizadores em dispositivos móveis, como um telemóvel ou um *tablet*, uma vez que, normalmente, estes dispositivos têm câmara frontal e câmara traseira. Por conseguinte, através desta funcionalidade o utilizador pode alternar a fonte da sua *stream* de vídeo, trocando de câmara, desde que esteja dentro da sala da videochamada.

Para executar esta funcionalidade o utilizador tem que premir no respetivo botão no menu das funcionalidades, que irá invocar a função *flipCamera()*. Nesta função, volta-se a fazer uso do método *getUserMedia()* para se obter acesso ao vídeo da outra câmara, através da alteração do valor da propriedade *facingMode* entre *user* e *environment*, responsáveis por selecionar a câmara frontal e traseira, respetivamente. Durante a ativação desta funcionalidade não é necessário recolher novamente o áudio. Caso o acesso seja bem sucedido, o novo vídeo é, imediatamente, apresentado no *front-end* do utilizador que trocou a câmara, substituindo o vídeo em emissão. Simultaneamente, o novo vídeo passa a ser enviado via WebRTC, substituindo o antigo, através do método *replaceTrack(),* para o outro utilizador presente na videochamada, sendo, de igual forma, apresentado no *front-end* desse utilizador.

4.1.7. PARTILHAR ECRÃ

No que diz respeito à funcionalidade Partilhar Ecrã, esta só se encontra disponível para o especialista remoto, isto é, somente este pode partilhar o seu ecrã e só se não estiver a usar a aplicação num dispositivo móvel. Portanto, terá de utilizar a aplicação num computador para usufruir desta funcionalidade.

Em termos de desenvolvimento desta funcionalidade, existe alguma similaridade com a funcionalidade Trocar Câmara. Quando o especialista se encontre numa videochamada pode iniciar a partilha de ecrã, premindo no botão presente no menu das funcionalidades, sendo chamada a função *screenShare()*.

Nesta função recorre-se ao método *getDisplayMedia()* da API *MediaDevices* para se ter acesso ao ecrã do especialista e posterior partilha. Caso o acesso seja bem sucedido, são desativadas as funcionalidades de Desligar/Ligar Câmara e de Adicionar Anotações de RA, uma vez não tem sentido utilizar estas funcionalidades quando a partilha de ecrã está ativa. Além disso, é imediatamente substituído o vídeo do especialista na interface gráfica do utilizador pelo vídeo da partilha de ecrã e alterado o *layout* do botão de partilha de ecrã. Simultaneamente, é alterada a *stream* de vídeo enviada para o operador pela *stream* de vídeo da partilha de ecrã. Através do método *replaceTrack()*, a nova *stream* de vídeo passa a ser apresentada na aplicação do operador.

Quando o especialista decidir, pode parar a partilha de ecrã, através do respetivo botão, invocando a função *stopScreenShare()*. Esta função é responsável por voltar a colocar a *stream* de vídeo original ativa, substituindo a partilha de ecrã pela *stream* de vídeo da câmara, voltando a permitir as funcionalidades de Desligar/Câmara e de Adicionar Anotações de RA e atualizando o *layout* do botão de partilha.

4.1.8. Adicionar Anotações de Realidade Aumentada

Esta funcionalidade é, sem dúvida, a mais inovadora e complexa, tendo exigido um estudo aprofundado das soluções existentes antes do início da sua implementação.

Após o estudo das bibliotecas apresentadas na subsecção Bibliotecas de Realidade Aumentada, constatou-se que a biblioteca OpenCV, através da sua versão JavaScript, poderia ser usada no desenvolvimento desta aplicação, usando-se o método Lucas-Kanade para adicionar esta funcionalidade.

O primeiro passo foi estudar este método e decidir a abordagem a seguir. Com esse intuito, um objetivo foi definido, desenvolver um simples protótipo provido, somente, com esta funcionalidade.

Para tal, foi desenvolvido um protótipo composto por uma simples página *web* com acesso à câmara do utilizador e, consequentemente, ao seu vídeo, apresentando-o ao utilizador, usando o método *getUserMedia()*, da API *MediaDevices*. Desta forma, o utilizador deste protótipo pode, através de um clique de rato ou um simples toque num ecrã compatível, ir escolhendo um ou vários pontos para rastrear. A partir desse momento é iniciado o processamento pela OpenCV.js, usando o método de Lucas-Kanade, e, por fim, é apresentado o *output* do processamento num elemento *canvas* no *front-end* do protótipo. O protótipo desenvolvido na implementação desta funcionalidade é apresentado na Figura 27.

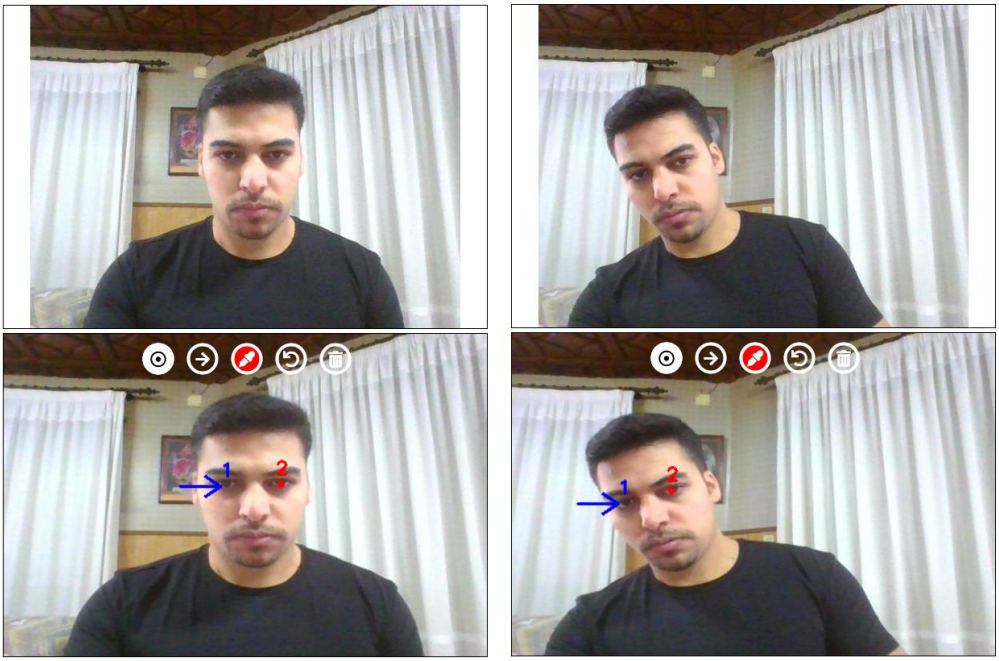


Figura 27 Protótipo da Funcionalidade Adicionar Anotações de RA

A Figura 27 revela duas perspetivas do protótipo desenvolvido, mostrando que as anotações acompanham os pontos a que estão associadas. A interface gráfica do protótipo é constituída por um elemento de vídeo, na parte superior, que contém o vídeo captado pela câmara do utilizador, sem alteração. Na parte inferior, é apresentado o resultado do processamento executado pela OpenCV num elemento *canvas*. Neste caso, o utilizador selecionou com o rato dois pontos para serem rastreados, localizados sobre os olhos, tendo cada ponto associada uma anotação de RA. Verifica-se, ainda, a existência de um menu de funcionalidades das anotações de RA, que contém as opções Definir o Tipo de Anotação, podendo escolher entre um ponto ou uma seta, Definir a Cor, Remover a Última Anotação, Remover Todas as Anotações.

Este protótipo foi submetido a testes de utilização que se revelaram muito positivos, sendo iniciada a sua implementação na aplicação da Tese. Nos próximos parágrafos é explicado, sucintamente, o desenvolvimento desta funcionalidade, desde a sua ativação até à sua desativação, clarificando todo o processo.

Para resumir a funcionalidade Adicionar Anotações de Realidade Aumentada nesta aplicação é apresentado o diagrama de sequência da Figura 28.

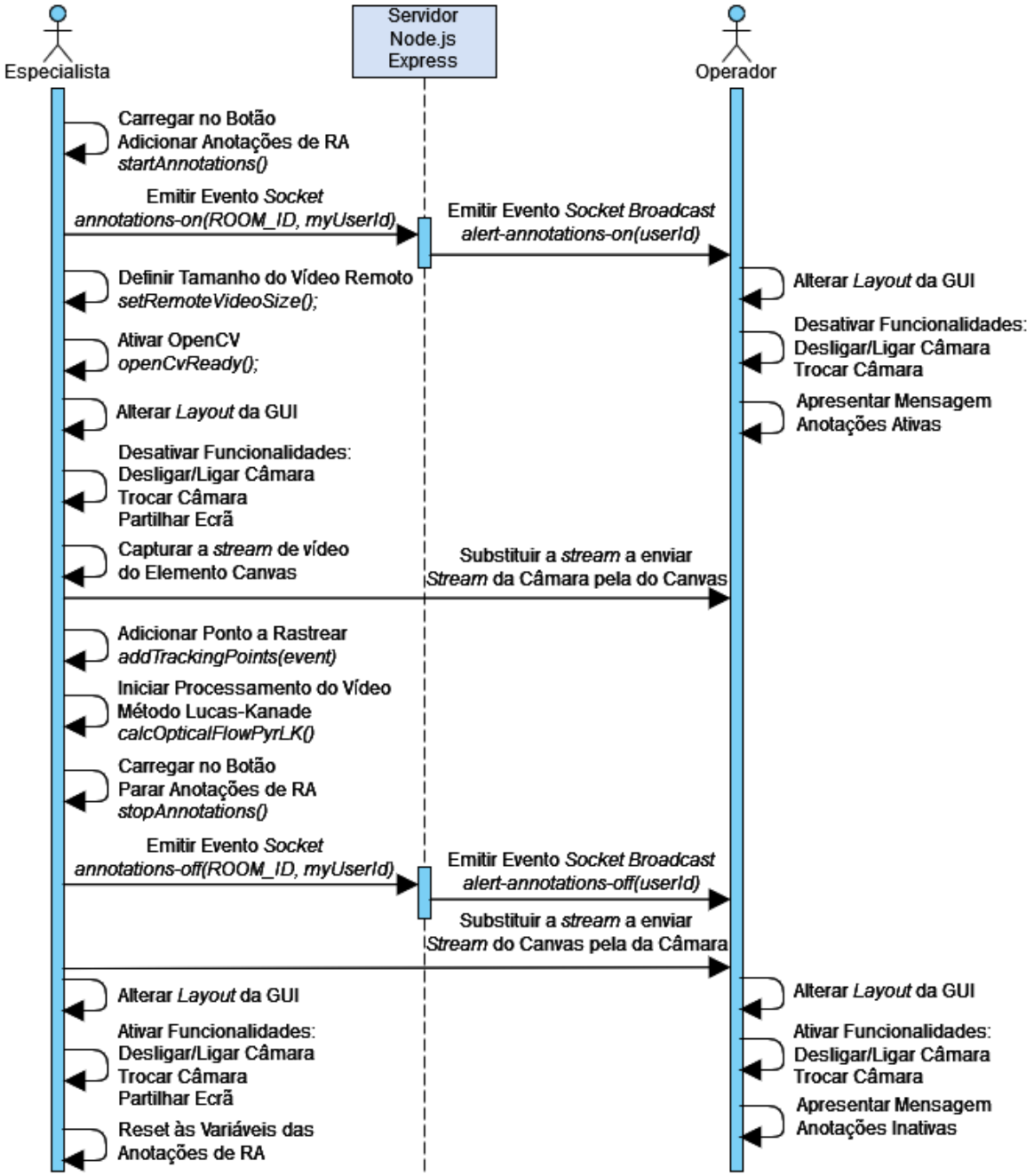


Figura 28 Diagrama de Sequência – Adicionar Anotações de Realidade Aumentada

Só o especialista tem permissão para adicionar anotações e é necessário que, tanto o especialista, como o operador, se encontrem na sala em videochamada. Analisando o diagrama da Figura 28, verifica-se que para ativar esta funcionalidade é necessário o especialista carregar no respetivo botão, presente no menu das funcionalidades, botão este responsável por chamar a função *startAnnotations()*.

Esta função é responsável por inúmeras ações. A primeira é o envio de um evento *socket* do tipo "*annotations-on*" do especialista para o servidor, que tem como parâmetros o *id* da sala e o *id* do utilizador que ativou a funcionalidade. Por sua vez, o servidor irá tratar deste evento e enviar um evento *socket broadcast* do tipo "*alert-annotations-on*", que será recebido e tratado apenas no lado do operador. Este evento permite informar o operador que o especialista ativou esta funcionalidade, sendo por isso necessário adaptar a GUI. Assim, é escondido o elemento do vídeo local e definido como vídeo principal o elemento do vídeo remoto, elemento que apresenta o vídeo enviado pelo especialista. Além disso, são desativadas as funcionalidades Desligar/Ligar Vídeo e Trocar a Câmara e é apresentada a mensagem "Anotações ativas!" na GUI.

Voltando ao lado do Especialista, a segunda ação executada na função *startAnnotations()* é definir o tamanho do elemento do vídeo remoto. Para tal, foi desenvolvida a função *setRemoteVideoSize()* que, tendo em conta o *aspect radio* do vídeo, tipo de dispositivo em uso e a largura do ecrã, vai calcular e definir o tamanho. Esta definição de tamanho é necessária para a ação seguinte.

A terceira ação protagonizada na função *startAnnotations()* é a ativação da OpenCV, sendo executada a função *openCvReady()*. Nesta função é realizado todo o processamento da biblioteca OpenCV. São definidas as variáveis a utilizar, nomeadamente a largura e a altura do elemento do vídeo remoto, permitindo, assim, sobrepor o elemento *canvas*, que terá o resultado do processamento, por cima do elemento *video*, que contém o vídeo remoto.

De seguida, é invocada a função *processVideo()*, que fica em *loop* enquanto esta funcionalidade estiver ativa. Enquanto não houver pontos a rastrear definidos pelo especialista, esta função é responsável apenas por preencher o elemento *canvas* com o vídeo do utilizador em tempo real, a uma taxa de 30 *frames* por segundo (FPS).

Em simultâneo, são realizadas as alterações no *layout* da GUI do especialista, sobrepondo o elemento *canvas* ao vídeo remoto, com o vídeo processado pela biblioteca OpenCV. Além disso, é apresentado o menu das anotações, sobre o *canvas*.

Outra ação realizada na função *startAnnotations()* é a desativação das funcionalidades Desligar/Ligar Câmara, Partilhar Ecrã e Trocar Câmara. Esta ação é realizada porque durante a adição de anotações não deve ser permitida a alteração da *stream* de vídeo a ser enviada.

Por fim, as últimas ações a executar nesta função são capturar e enviar a *stream* de vídeo presente no elemento *canvas* para o operador. De forma a capturar a *stream* é utilizado o método *captureStream()* da API *HTMLCanvasElement*, com o parâmetro *framerate* definido a 30 FPS. Este método permite obter um objeto *MediaStream*, que acede à *stream* de vídeo do elemento *canvas* através do método *getVideoTracks()*. Quanto ao envio desta *stream*, é realizado via WebRTC através da biblioteca PeerJS, mais concretamente, usando o método *replaceTrack()*, capaz de substituir a *stream* de vídeo da câmara, enviada até este momento, pela *stream* do *canvas*.

Esta foi a melhor solução encontrada para enviar o *output* do processamento da OpenCV do especialista para o operador. As vantagens desta solução passam por reaproveitar mais uma vez o mecanismo de envio de *stream* via WebRTC e fixar o processamento apenas ao lado do especialista remoto, que em casos normais, usará a aplicação num computador, caracterizado por uma maior capacidade de processamento quando comparada com a dos dispositivos móveis, que na teoria serão utilizados mais frequentemente pelo operador no terreno.

Após as ações executadas pela função *startAnnotations()*, o Especialista pode, a qualquer momento, escolher de um até cinco pontos para rastrear. Para isso, basta clicar com o rato ou toque de dedo no ponto do vídeo do operador que pretenda que seja rastreado, cada anotação está associada a um ponto a rastrear. É importante salientar que, durante o processo de anotações, o especialista pode fazer uso do menu das anotações, apresentado na Figura 29, que providencia as seguintes funcionalidades: Definir o Tipo de Anotação, Definir a Cor da Anotação, Inserir Texto da Anotação, Remover a Última Anotação e Remover Todas as Anotações.

Figura 29 Menu das Anotações de Realidade Aumentada

No Tipo de Anotação encontram-se duas possibilidades, um ponto ou uma seta, sendo que a seta pode apontar para cima, baixo, esquerda e direita. Para desenhar o ponto é utilizada a função *cv.circle()* da biblioteca OpenCV, enquanto a seta é desenhada pela função *drawArrow()* desenvolvida especificamente durante o projeto, visto que a função *cv.arrowedLine()* ainda não tinha sido implementada na versão OpenCV.js usada. A função *drawArrow()* faz uso da função *cv.line()*, que permite desenhar linhas retas, sendo a seta desenhada através da união de 3 linhas retas, cada uma com as suas propriedades.

Em relação à Cor da Anotação, esta pode ser escolhida na GUI através do *color picker* proveniente do elemento HTML *input* do tipo *color* introduzido, sendo que o comportamento deste elemento varia de *browser* para *browser* e até consoante o sistema operativo. Depois da seleção da cor pelo especialista, esta é passada como parâmetro nas funções responsáveis pelo desenho das anotações.

A funcionalidade Inserir Texto foi a última a ser desenvolvida, não estando presente no protótipo de Adicionar Anotações de RA. Permite facultar mais informações ao operador, sendo uma mais-valia para videochamadas em ambientes de muito ruído ou em casos em que o operador seja um cliente com problemas auditivos. Para inserir o texto foi utilizada a função *cv.putText()*, também utilizada para numerar as anotações, possibilitando ao operador executar as instruções pela ordem correta.

Quanto à funcionalidade Remover a Última Anotação, esta foi desenvolvida a pensar no caso do especialista se enganar a introduzir uma anotação ou simplesmente a quiser apagar. Para tal foi desenvolvida a função *undoLastAnnotation()*, responsável por remover o último elemento dos *arrays* que guardam os pontos a rastrear, designados no código por *trackingPoints* e *oldPoints*. O *array trackingPoints* é utilizado para guardar algumas das informações de cada anotação, nomeadamente, o tipo de anotação, a sua cor e o texto, caso exista. Enquanto o *array oldPoints* guarda as coordenadas das anotações no vídeo, este *array* está em constante atualização, pois é utilizado indiretamente como *input* e atualizado pelo *output* do método Lucas-Kanade. A funcionalidade Remover a Última Anotação só se encontra disponível se existirem pontos a serem rastreados.

Por fim, a funcionalidade Remover Todas as Anotações possibilita excluir as anotações adicionadas até ao momento, recorrendo à função *clearAnnotations()*, que esvazia os respetivos *arrays*. Tal como a funcionalidade anterior, esta só se encontra disponível se existirem pontos a serem rastreados.

Desta forma o especialista pode configurar as anotações a inserir no momento em que escolhe um ponto para rastrear. Estes pontos são adicionados através da função *addTrackingPoints()*. Esta função identifica a posição no vídeo onde foi realizado o clique pelo especialista, sendo armazenada numa variável que será utilizada como *input* o cálculo de *optical flow*. A função *addTrackingPoints()* é responsável por iniciar o processamento pelo método Lucas-Kanade.

O método de *Optical Flow* Lucas-Kanade permite rastrear em tempo real o movimento de determinados pontos do vídeo enviado pelo operador. Para a implementação deste método foi analisado o exemplo mencionado na subsecção OpenCV. O cálculo do *optical flow* neste método é realizado pela comparação dois *frames*, sendo que no primeiro *frame* sabe-se a localização dos pontos a rastrear e no segundo a localização é desconhecida. Após a comparação dos *frames*, é descoberta a localização desses pontos no segundo *frame*, possibilitando a ancoragem das anotações efetuadas.

Após o cálculo do *optical flow* é necessário verificar e guardar os resultados obtidos. Para tal, é percorrido o vetor *status* que permite saber o resultado da comparação entre os *frames* sobre cada ponto a rastrear. Se o *status* para um determinado ponto for igual a 1, significa que foi encontrada a localização desse ponto no segundo *frame*, sendo guardada a localização. Por outro lado, se o resultado do *status* sobre um determinado ponto a rastrear for igual a 0, significa que esse ponto não foi localizado no segundo *frame*, sendo esse ponto perdido. Esta anotação deixa de ser desenhada sobre o vídeo, uma vez que o método Lucas-Kanade deixou de conseguir calcular a sua localização no segundo *frame*.

Quando os pontos são localizados com sucesso no segundo *frame*, é iniciado o desenho das anotações associadas a esses pontos na respetiva localização descoberta. Estas anotação são desenhadas pelas funções supramencionadas nesta subsecção, sejam pontos, setas, com ou sem texto, e numa determinada cor. Por fim, o resultado final é inserido no elemento *canvas* presente no *front-end* da aplicação.

Ao mesmo tempo que o vídeo resultante do processamento pelo método Lucas-Kanade é apresentado no elemento *canvas* no *front-end* da aplicação do Especialista, é, também captado, enviado e apresentado no *front-end* da aplicação do operador. Durante este processo, o especialista pode continuar a adicionar anotações ou simplesmente esperar que o operador realize as instruções dadas através das anotações. De salientar que o processamento pelo método Lucas-Kanade é contínuo enquanto existirem pontos para rastrear.

O especialista remoto tem a capacidade de desativar a qualquer momento as Anotações de Realidade Aumentada, bastando carregar no correspondente botão, que invoca a função *stopAnnotations()*. Nesta função é enviado do especialista para o servidor um evento *socket* do tipo "*annotations-off*", com o *id* da sala e o *id* do utilizador, com o objetivo de informar que a funcionalidade das anotações foi desativada. O servidor irá tratar do evento e emitir um evento *socket* de *Broadcast*, informando o operador que a funcionalidade foi parada, alterando o *layout* e ativando novamente as funcionalidades Desligar/Ligar Vídeo e Trocar a Câmara e apresentando uma mensagem informativa.

Simultaneamente, no lado do especialista e ainda na função *stopAnnotations()* é alterado a *stream* de vídeo a enviar para o operador, voltando a ser enviada a *stream* de vídeo captada pela câmara do especialista. Além disso, são efetuadas alterações no *layout*, ativando as funcionalidades Desligar/Ligar Câmara, Partilhar Ecrã e Trocar Câmara e, por fim, efetuando o *reset* às variáveis associadas à funcionalidade das anotações.

Em suma, na funcionalidade de Adicionar Anotações de RA, o especialista pode escolher pontos para rastrear no vídeo do operador, podendo cada ponto tem uma anotação associada. Estes pontos devem ser *corner points* para o seu rastreio ser eficaz. Após a escolha do primeiro ponto, é iniciado todo o processamento da biblioteca OpenCV, recorrendo ao método Lucas-Kanade. Este procura as localizações dos pontos a rastrear *frame* após *frame*, desenha as anotações a partir dessas localizações e o vídeo resultante é apresentado na GUI do especialista e enviado, simultaneamente, para o operador. Aí, é apresentado na interface gráfica do operador, permitindo-lhe executar as instruções dadas para a resolução dos problemas que motivaram a assistência remota.

4.1.9. SAIR DA SALA DA VIDEOCHAMADA

Por fim, a funcionalidade de Sair da Sala da Videochamada não possui nenhum botão específico na interface gráfica da aplicação, bastando o utilizador fechar o *browser* ou o respetivo separador onde acede à aplicação. Para sintetizar esta funcionalidade é apresentado um diagrama de sequência através da Figura 30.

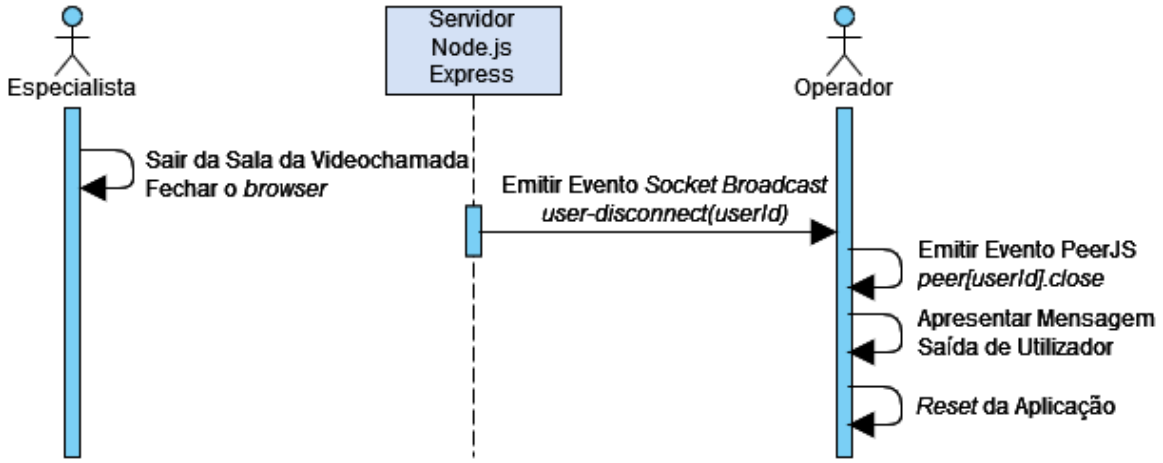


Figura 30 Diagrama de Sequência – Sair da Sala da Videochamada

Analisando a Figura 30, verifica-se o exemplo de um especialista a sair da sala da videochamada. No exato momento em que o especialista decide sair, o servidor deteta essa saída, emitindo um evento *socket broadcast* do tipo "*user-disconnected*" com o *id* desse utilizador como parâmetro. Esse evento é recebido e tratado no lado do utilizador que permanece na sala. Nesse momento é emitido o evento *peer[userId].close* do módulo PeerJS que informa da saída do utilizador. Este evento permite apresentar uma mensagem na interface gráfica da aplicação informando a saída do utilizador e reiniciar a aplicação, nomeadamente, a interface gráfica e as funcionalidades. É importante referir que não interessa o utilizador que sai em primeiro lugar da sala da videochamada porque as ações despoletadas serão as mesmas.

4.2. DESENVOLVIMENTO DA INTERFACE GRÁFICA DO UTILIZADOR

Neste subtópico é explicado o desenvolvimento da interface gráfica do utilizador (GUI) da aplicação, isto é, o *layout* do *front-end* da aplicação.

A GUI foi desenvolvida tendo em conta algumas premissas, nomeadamente, a responsividade, utilizável quer em dispositivos móveis, como *tablets* e telemóveis, como em computadores e compatível com os principais *browsers*. Outra premissa, foi a usabilidade, que se traduz na facilidade do utilizador usar a aplicação através de um *layout* intuitivo.

A GUI foi desenvolvida recorrendo à linguagem de marcação HTML, à linguagem de estilo *Cascading Style Sheets* (CSS) e à linguagem de programação JavaScript. O HTML permite estruturar o conteúdo da interface gráfica da aplicação, através de diferentes elementos, como, *div*, *span*, *video*, entre outros.

A CSS foi utilizada para adicionar estilos aos elementos HTML, através de classes, *ids* ou código *inline*, melhorando o aspeto da interface gráfica, tornando-a mais apelativa e intuitiva. Além disso, o uso de *media queries* viabilizou a responsividade na GUI.

Por fim, o JavaScript permite atualizar o *layout* da GUI ao longo da utilização da aplicação, adicionando interatividade através de inúmeras funções, capazes de alterar ícones, cores ou outras propriedades.

O resultado final do desenvolvimento da GUI para computadores é ilustrado na Figura 31 e para dispositivos móveis na Figura 32.

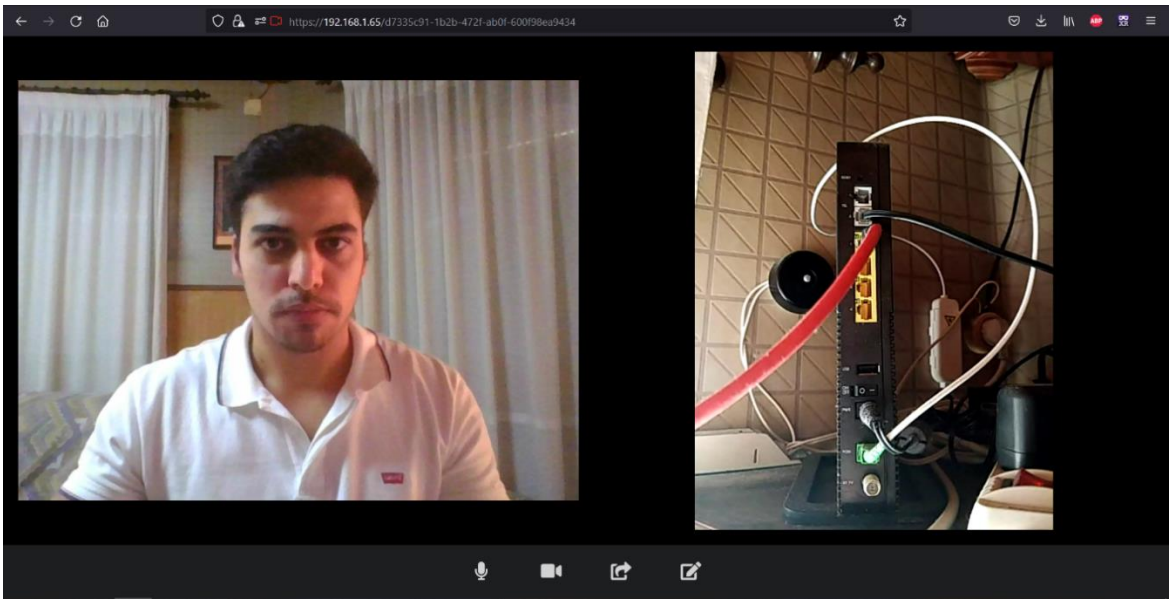


Figura 31 Interface Gráfica do Utilizador – Computador

A Figura 31 apresenta a GUI da aplicação num computador portátil através do *browser* Firefox. A GUI da aplicação pode ser dividida em duas secções. A primeira ocupa cerca de 90 % da altura da GUI e contém os vídeos, lado a lado, quer do próprio utilizador, no vídeo à esquerda, quer do outro utilizador, no vídeo à direita. A segunda secção pode ser vista na zona inferior da GUI e é composta pelo menu das funcionalidades. Na Figura 31 estamos perante a GUI de um especialista, tendo o menu 4 botões responsáveis pelas funcionalidades de Desligar/Ligar Micro, Desligar/Ligar Câmara, Partilhar Ecrã e Adicionar Anotações de Realidade Aumentada, respetivamente.

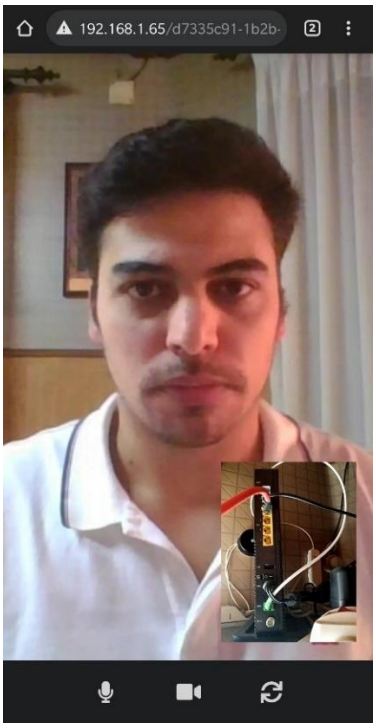


Figura 32 Interface Gráfica do Utilizador – Dispositivos Móveis

A Figura 32 apresenta a GUI da aplicação em dispositivos móveis, neste caso num telemóvel, usando o *browser* Chrome. Esta GUI é similar à do computador, diferindo nos vídeos. Neste caso, um deles está em foco, ocupando praticamente o ecrã todo, e outro vídeo surge, em pequena dimensão, no canto inferior direito. É de salientar que o utilizador pode definir o vídeo em foco, bastando clicar em cima do vídeo de menores dimensões, passando o que está em foco para tamanho reduzido e vice versa. Na parte inferior da GUI, contém o menu das funcionalidades. Neste caso, possui apenas 3 botões/funcionalidades visto que se trata da GUI de um operador no terreno, com acesso às funcionalidades de Desligar/Ligar Micro, Desligar/Ligar Câmara e Trocar Câmara.

5. TESTES DE SISTEMA E ANÁLISE DE RESULTADOS

Nesta secção são apresentados e analisados os testes de sistema efetuados para caracterizar a aplicação desenvolvida. Inicialmente, é caracterizada a rede de testes e os dispositivos utilizados. Na primeira subsecção são testados os diversos casos de uso desenvolvidos, através de um exemplo real de utilização da aplicação, servindo como um teste geral à aplicação. Quanto à segunda subsecção, é responsável por analisar diferentes métricas, como o *Round Trip Time* (RTT), a largura de faixa, entre outros, permitindo definir as condições ideais e as condições mínimas necessárias para a existência de uma boa experiência de utilização da solução de assistência remota.

Primeiramente, é importante fazer a caracterização da rede e dos dispositivos utilizados nos testes realizados nas subsecções 5.1 e 5.2. Nesse sentido, são apresentadas várias tabelas. Na Tabela 2 caracteriza-se a rede, enquanto as Tabelas 3, 4 e 5 apresentam as características dos dispositivos utilizados para o servidor, pelo especialista e pelo operador, respetivamente.

Tabela 2 Caracterização da Rede dos Testes de Sistema

Modelo do *Router*	Meo FiberGateway GR241AG	
Velocidade Máxima da Rede	*Download*	*Upload*
	100 Mb/s	100 Mb/s
Média da Velocidade no Portátil - Wi-Fi 2.4 Ghz	*Download*	*Upload*
	90.7 Mb/s	80.4 Mb/s
Média da Velocidade no Telemóvel - Wi-Fi 2.4 Ghz	*Download*	*Upload*
	43.2 Mb/s	68.8 Mb/s

A Tabela 2 caracteriza a rede, nomeadamente, o modelo do router utilizado na rede de testes, a velocidade máxima disponível nesta rede, que está limitada a 100 Mb/s. Além disso, revela os valores médios das cinco medições efetuadas via Wi-Fi na frequência de 2.4GHz, através do *website* www.speedtest.net/pt, no portátil do especialista e no telemóvel do operador. É importante salientar que, durante a realização dos testes desta secção, a aplicação estava apenas acessível a utilizadores dentro da rede do servidor, portanto, quer o especialista como o operador, estavam ligados à mesma rede do servidor.

Tabela 3 Caracterização do Computador Portátil do Servidor

Computador Portátil	ASUS
Processador	Intel(R) Core(TM) i7-4700HQ CPU @ 2.40 GHz
Memória RAM	8 GB
Sistema Operativo	Windows 10

Tabela 4 Caracterização do Computador Portátil do Especialista

Computador Portátil	OMEN by HP
Processador	Intel(R) Core(TM) i7-8750H CPU @ 2.20 GHz
Memória RAM	16 GB
Sistema Operativo	Windows 10
Browser	Google Chrome versão 94

Tabela 5 Caracterização do Telemóvel do Operador

Telemóvel	OnePlus 5T
Processador	Snapdragon 835
Memória RAM	8 GB
Sistema Operativo	OxygenOS versão 10.0.1 baseado em Android 10
Browser	Google Chrome versão 94

5.1. TESTES AOS CASOS DE USO

Nesta subsecção são apresentados os testes aos casos de uso desenvolvidos para esta aplicação. Foi definido um contexto de teste geral que consiste na simulação de uma assistência remota utilizando a aplicação desenvolvida.

O problema real representado no teste geral da aplicação traduz-se num problema de ligação à *Internet*, no qual um cliente entrará em contacto com um técnico de uma operadora de telecomunicações para receber instruções que possibilitem resolver o seu problema de ligação à Internet e, até, mitigar outras dúvidas do cliente. Os atores deste sistema são o especialista remoto e o operador no terreno. De maneira a manter essa nomenclatura, o técnico da operadora será tratado por especialista e o cliente por operador.

Após o contacto do operador com o suporte técnico da operadora, o especialista remoto decide dar seguimento à assistência utilizando a aplicação de assistência remota desenvolvida. Para tal, utilizando um *browser*, ele acede à aplicação através do endereço https://192.168.1.81.

Nesse momento, a aplicação deteta a conexão e redireciona o especialista para um endereço que contém o UUID aleatoriamente gerado. Assim, o especialista cria e entra na sala da videochamada, sendo-lhe pedida autorização para aceder à sua câmara e microfone. Após o especialista autorizar o acesso aos seus média, é-lhe apresentada a GUI da aplicação como se ilustra na Figura 33.

Figura 33 Teste Geral à Aplicação – Criar/Entrar na Sala da Videochamada

De seguida, o especialista convida o operador para a videochamada. Para tal, clica no botão adicionar utilizador presente no menu das funcionalidades, aparecendo diversas opções, como se mostra na Figura 34, nomeadamente, *email*, telemóvel ou por cópia e partilha do endereço com o cliente. O *layout* apresentado é o do Chrome executado sobre Windows 10.

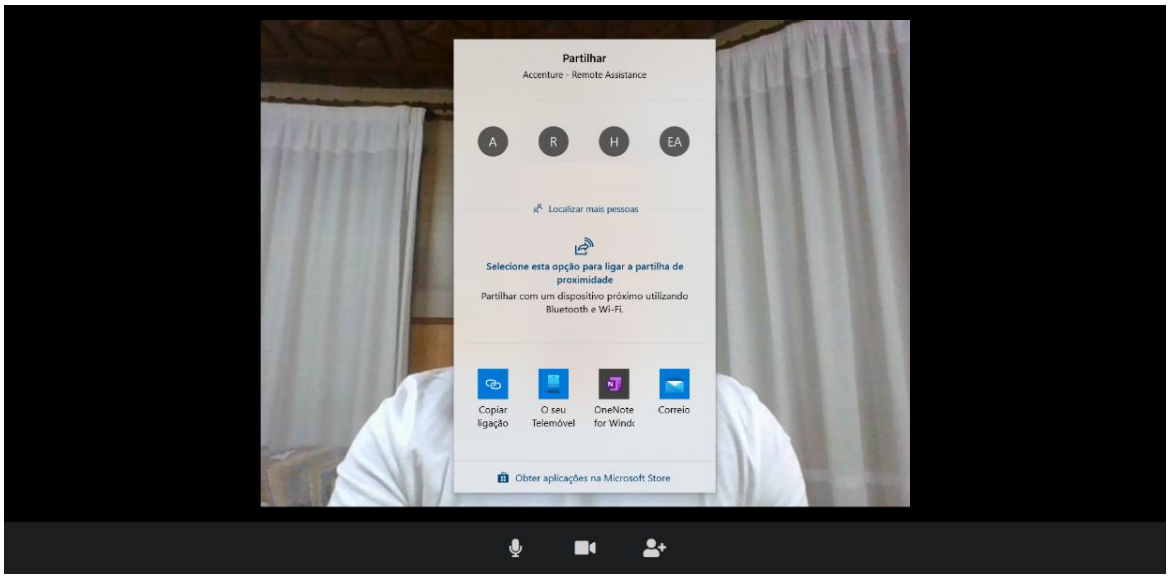


Figura 34 Teste Geral à Aplicação – Convidar Utilizador

Quando o operador aceita o convite dá-se início à videochamada, tal como retrata a Figura 35. A GUI do especialista altera-se, nomeadamente, passando a apresentação dos vídeos a ficar lado a lado e surgem os botões das funcionalidades só permitidas durante a videochamada, como Partilhar Ecrã e Adicionar Anotações de Realidade Aumentada.

Figura 35 Teste Geral à Aplicação – Entrar na Sala da Videochamada

Na Figura 35 verifica-se a entrada do operador na sala da videochamada utilizando um dispositivo móvel, neste caso um telemóvel. De imediato é iniciada a videochamada, os vídeos de ambos os utilizadores são apresentados, estando o vídeo do especialista remoto em maior destaque e o do operador no canto inferior direito, sendo o vídeo do operador captado pela câmara frontal do telemóvel, tal como configurado no código.

No início da assistência remota o especialista questiona o operador sobre o problema, de forma a resolver e esclarecer as questões. Neste caso, o operador questionou o especialista sobre como resolver o problema de acesso à *Internet* e o especialista pediu ao operador para trocar para a câmara traseira. A Figura 36 demonstra a GUI do operador após ter trocado para a câmara traseira, sendo mostrado o *router* do operador.

Figura 36 Teste Geral à Aplicação – Trocar Câmara

Após a troca de câmara efetuada pelo operador, o especialista reconheceu o modelo do *router* e partilhou o seu ecrã para elucidar o operador sobre a interface do *router*, nomeadamente as suas portas e botões. A Figura 37 revela a visão do operador após a ativação da funcionalidade Partilhar Ecrã por parte do especialista. Além disso, é possível verificar a ausência do menu de funcionalidades, uma vez que em *mobile* é possível ocultá-lo, permitindo ao operador ver melhor a partilha de ecrã.

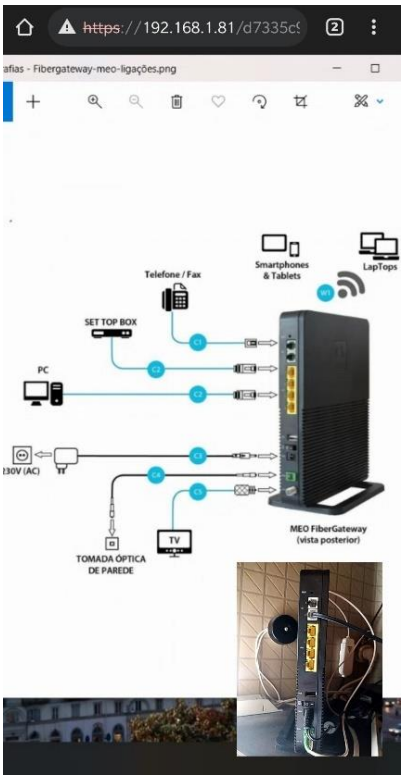


Figura 37 Teste Geral à Aplicação – Partilhar Ecrã

Depois de partilhar o ecrã, o especialista optou por ativar a funcionalidade de Adicionar Anotações de Realidade Aumentada, para facilitar a compreensão das instruções. Decidiu adicionar uma seta no botão de desligar/ligar o *router*, dando a informação necessária ao operador para reiniciar o dispositivo e possivelmente resolver o seu problema de conexão à *Internet*. A Figura 38 demonstra a adição desta anotação por parte do especialista.

Figura 38 Teste Geral à Aplicação – Adicionar Anotações de RA

Através da visualização da anotação, o operador desligou e ligou o *router*, resolvendo o problema associado à sua *Internet*. Aproveitando a ocasião, o operador questionou o especialista sobre como poderia ligar o seu computador por cabo ao *router*. Para responder a esta questão, o especialista adicionou mais uma anotação, indicando o local onde deveria ligar o cabo, tal como é demonstrado na Figura 39.

Figura 39 Teste Geral à Aplicação – Adicionar Anotações de RA 2

Na Figura 39 é possível verificar as duas anotações de RA adicionadas. As anotações mantêm-se fixas ao ponto escolhido pelo especialista mesmo que seja alterada a posição da câmara ou do objeto que contém o ponto a ser rastreado. Isso é visível através da Figura 40.

Figura 40 Teste Geral à Aplicação – Adicionar Anotações de RA 3

Para testar as restantes funcionalidades, nomeadamente, Desligar/Ligar Câmara e Desligar/Ligar Micro, o especialista pediu para o operador desligar o seu micro e câmara, selecionando os botões no menu das funcionalidades, sendo o resultado visível na Figura 41.

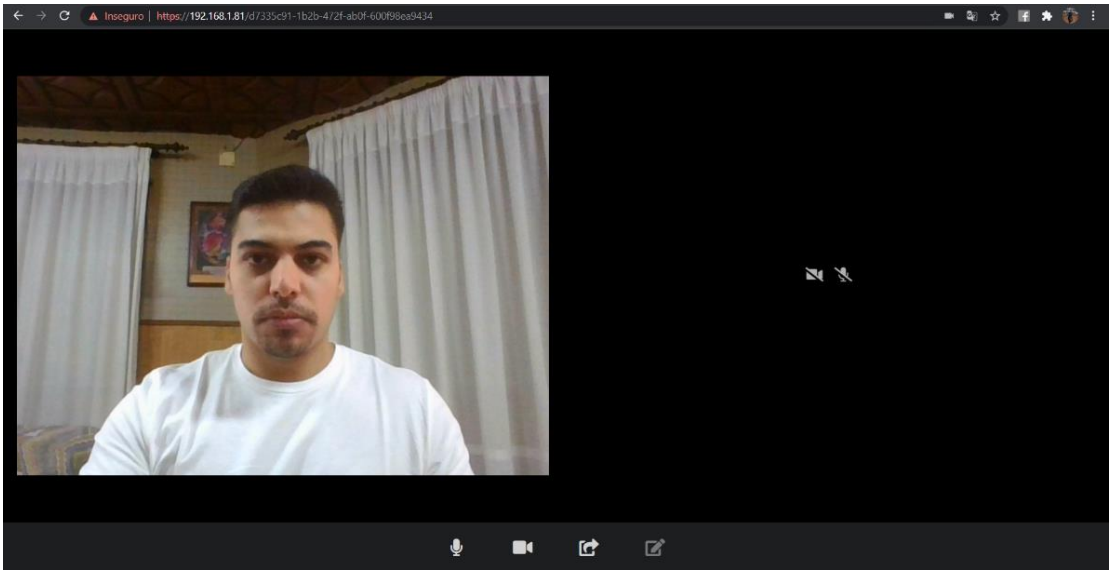


Figura 41 Teste Geral à Aplicação – Desligar/Ligar Micro e Câmara

Desta forma foi possível confirmar o seu correto funcionamento, deixando de receber o áudio e o vídeo do operador e sendo apresentados os ícones na GUI do especialista.

Por fim, tendo sido finalizada a assistência remota, o operador abandonou a sala da videochamada, aparecendo a mensagem da sua saída no ecrã do especialista, como é revelado na Figura 42.

Figura 42 Teste Geral à Aplicação – Sair da Sala da Videochamada

Em suma, o teste geral à aplicação demonstrou o correto funcionamento de todos os casos de uso, estando a aplicação operacional e cumprindo os objetivos estabelecidos no início do projeto. É de salientar, também, que a aplicação foi testada com sucesso em vários dispositivos, nomeadamente, computadores portáteis com sistema operativo Windows, em *tablets* e telemóveis com Android e iOS, tendo sido utilizados vários *browsers* nesses testes como o Chrome, o Firefox, o Safari e o Samsung Internet.

5.2. DEFINIÇÃO DE MÉTRICAS

Nesta subsecção são estudadas e definidas diferentes métricas, como o *Round Trip Time* (RTT), a perda de pacotes, a largura de faixa, a resolução do vídeo e o *frame rate*, de forma a caracterizar o funcionamento do sistema. Para isso, é analisada a aplicação, sem e com restrições nas várias métricas. Em suma, esta análise permite definir requisitos mínimos para que o utilizador da aplicação tenha uma boa experiência.

A análise das métricas é suportada por dados recolhidos durante os primeiros 60 s de uma assistência remota, recorrendo à média das cinco iterações efetuadas. Em cada iteração foram recolhidos os dados necessários, através da ferramenta *webrtc-internals* e do *Wireshark*, que é o programa mais utilizado para analisar redes. Neste caso, foram identificadas e analisadas as *streams* RTP, responsáveis pelo envio do áudio e vídeo entre os utilizadores da aplicação, possibilitando recolher informações sobre a largura de faixa e as perdas de pacotes de cada *stream*.

A *webrtc-internals* é uma ferramenta presente no Google Chrome. Disponibiliza as estatísticas das sessões WebRTC em curso, permitindo analisar em tempo real os dados na página chrome://webrtc-internals/, incluindo diversos gráficos que são gerados e atualizados em tempo real a partir desses dados, ou exportar esses dados para um ficheiro de texto. Esta análise possibilita descobrir eventuais problemas decorrentes do desenvolvimento ou da implantação da tecnologia WebRTC em aplicações. Nesta análise a ferramenta permitiu estudar e definir inúmeras métricas, como o *Round Trip Time* (RTT), as resoluções e o *frame rate* do vídeo enviado pelos utilizadores.

De forma a perceber o comportamento geral da aplicação, é analisado, primeiramente, o uso da aplicação sem restrições. Para tal, é iniciada uma assistência remota entre o especialista, no computador da Tabela 4, e o operador com o telemóvel da Tabela 5.

Relativamente ao *Round Trip Time* (RTT), o valor médio obtido foi cerca de 7 ms. Este valor é extremamente baixo e pode ser justificado pelos testes terem sido realizados na rede local e pelas *streams* de vídeo e áudio serem enviadas diretamente entre dois utilizadores, *peer-to-peer*, sem a necessidade de servidor, devido ao uso do WebRTC, que possibilita latências inferiores a 500 ms, como foi descrito na subsecção WebRTC. A Figura 43 revela o gráfico da média dos dados relativos ao RTT.

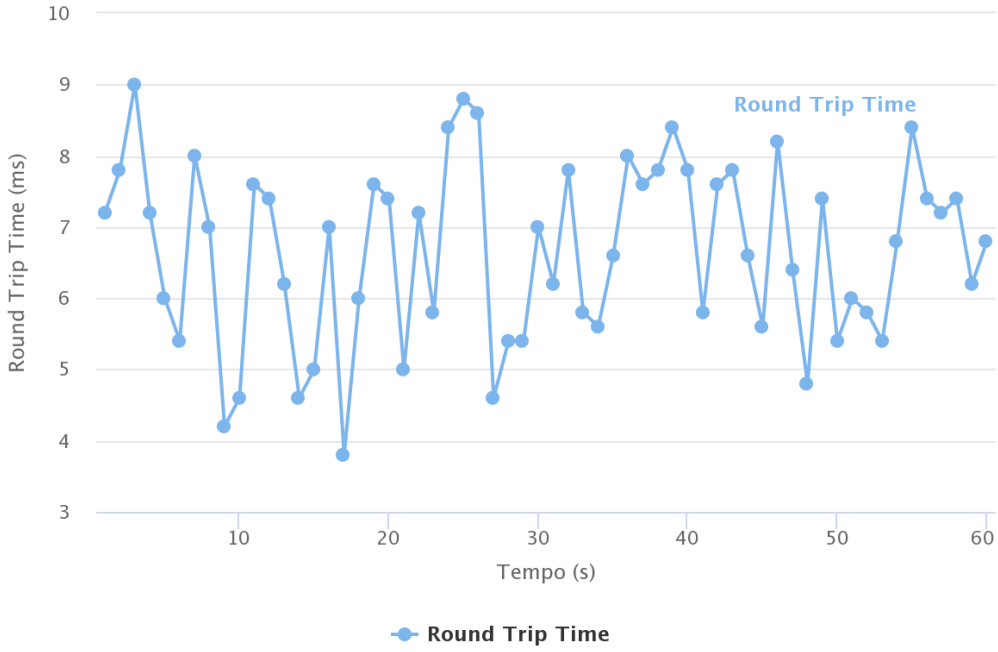


Figura 43 Análise da Aplicação Sem Restrições – RTT

Quanto à métrica da largura de faixa, o valor médio utilizado pela *stream* de vídeo, durante o teste sem restrições foi aproximadamente de 1.9 Mb/s, quer pelo especialista como pelo operador, como se pode observar na Figura 44. Quanto à largura de faixa utilizada pela *stream* de áudio, por cada utilizador, é cerca de 50 kb/s, um valor muito baixo em comparação com o da *stream* de vídeo, e por esse motivo, não será alvo de análise detalhada.

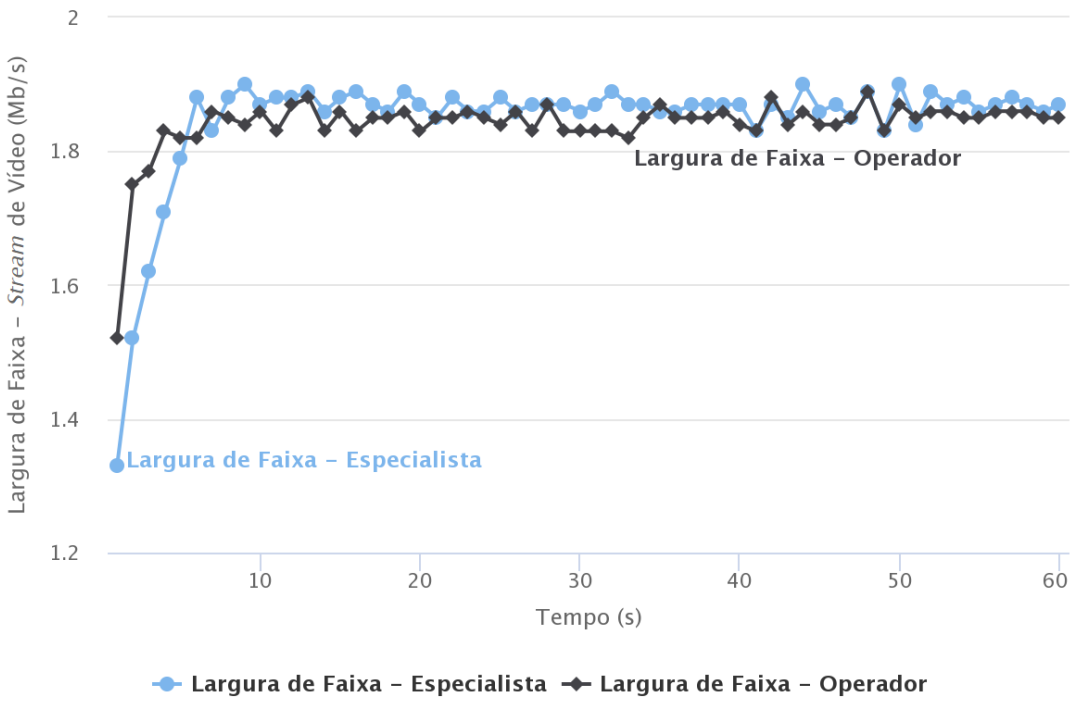


Figura 44 Análise da Aplicação Sem Restrições – Largura de Faixa

A proximidade do valor médio da Largura de Faixa utilizada por ambos os utilizadores é justificada pelas características do vídeo enviado serem semelhantes, sendo que o *frame rate* ronda os 30 *frames* por segundo (FPS) para ambos, enquanto a resolução do vídeo, no caso do especialista é de 640x480 pixéis e no caso do operador é de 480x640 pixéis. Estes dados são confirmados pelas medições presentes na Figura 45 e na Figura 46.

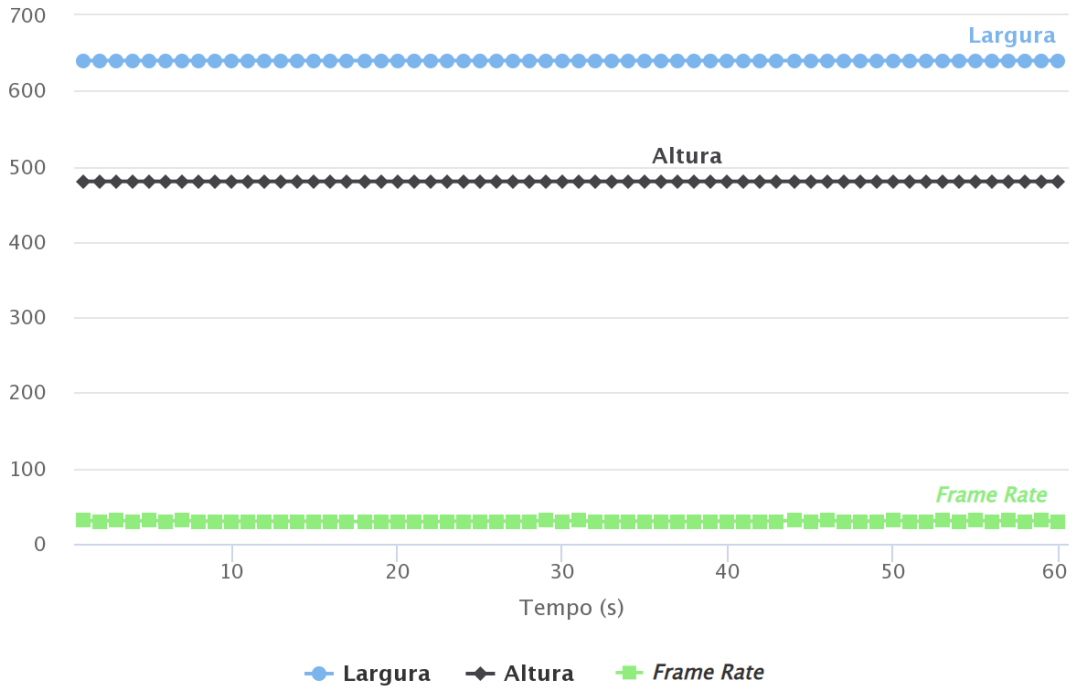


Figura 45 Análise da Aplicação Sem Restrições – Resolução e *Frame Rate* do Especialista

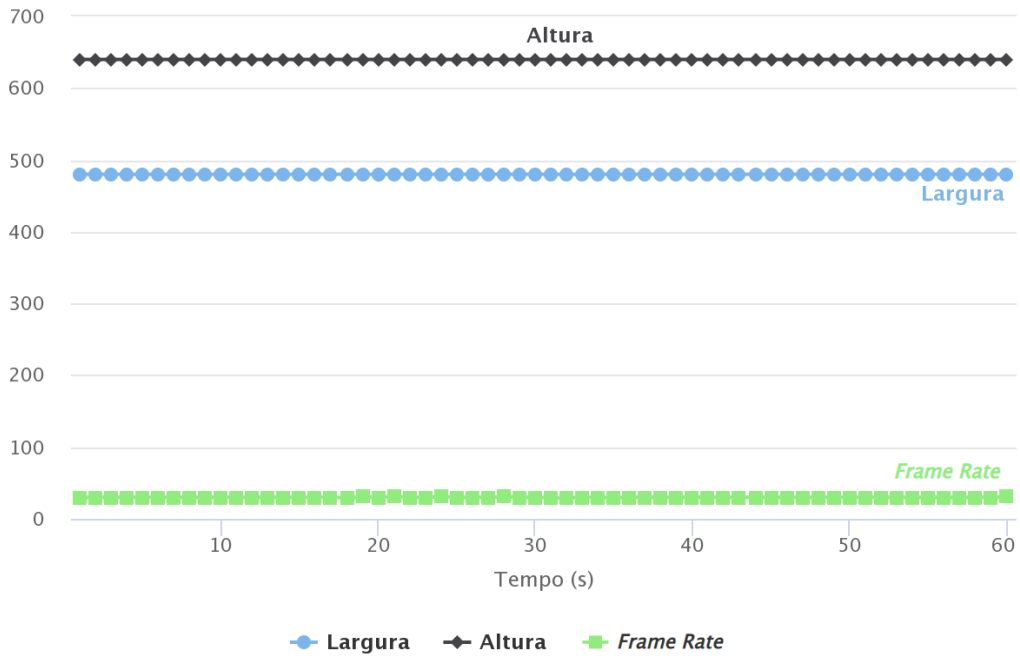


Figura 46 Análise da Aplicação Sem Restrições – Resolução e *Frame Rate* do Operador

Nesta análise da aplicação sem restrições também foi analisada a métrica perda de pacotes, que no decorrer das cinco iterações efetuadas teve o valor de 0 %, isto é, não existiu qualquer perda de pacotes.

De forma a resumir os dados anteriores, a Tabela 6, apresenta a síntese dos resultados do teste da aplicação sem restrições.

Tabela 6 Síntese dos Resultados da Análise da Aplicação Sem Restrições

RTT	7 ms	
Perda de Pacotes	0 %	
Largura de Faixa – Especialista *Stream* de vídeo	1.9 Mb/s	
Largura de Faixa – Operador *Stream* de vídeo	1.9 Mb/s	
Resolução de Vídeo - Especialista	Largura: 640 px	Altura: 480 px
Resolução de Vídeo - Operador	Largura: 480 px	Altura: 640 px
Frame Rate - Especialista	30 FPS	
Frame Rate - Operador	30 FPS	

Para estudar mais aprofundadamente o impacto que as métricas podem ter no sistema, procedeu-se à análise do uso da aplicação com restrições ao nível da largura de faixa e da latência. O objetivo desta análise é identificar a largura de faixa mínima para que o utilizador tenha uma boa experiência e descobrir as consequências da latência numa aplicação de tempo real.

O procedimento para a realização do teste com restrições à largura de faixa é similar ao sem restrições, uma vez que é executada uma assistência remota e analisados os primeiros 60 s. Nos primeiros 30 s não há qualquer restrição e nos restantes 30 s é introduzida a restrição da largura de faixa, nomeadamente, limitando-a a 1 Mb/s, 512 kb/s e 256 kb/s. Para tal foi utilizada a versão trial da aplicação *softPerfect Connection Emulator*, que permite restringir a rede durante 30 s, tendo em conta várias métricas como a largura de faixa e a adição de latência. Todas restrições foram efetuadas na rede Wi-Fi do especialista, quer nos pacotes de entrada, quer nos pacotes de saída.

A Figura 47 apresenta a variação da largura de faixa utilizada pela *stream* de vídeo de cada utilizador ao longo dos 60 s medidos nos testes com restrição da largura de faixa a 1 Mb/s, sendo comprovado o funcionamento da aplicação *softPerfect Connection Emulator*.

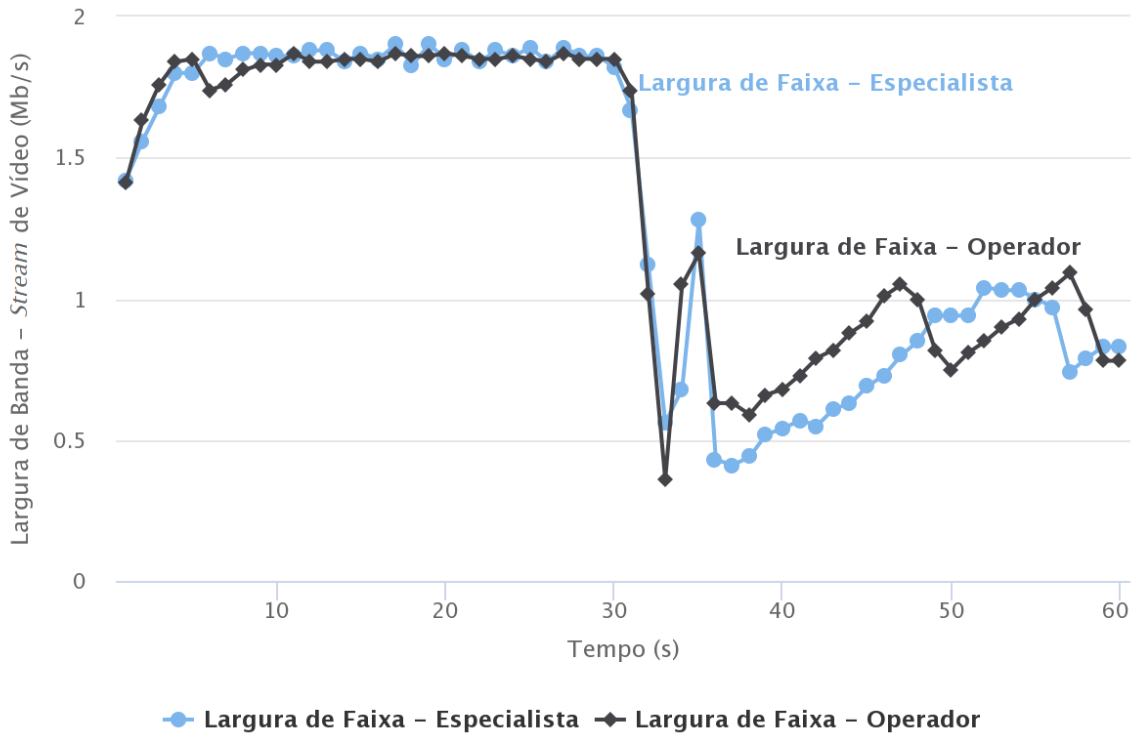


Figura 47 Análise da Aplicação Com Restrição – Largura de Faixa 1 Mb/s

Analisando a Figura 47, verifica-se que, no começo, o valor da largura de faixa sobe até estabilizar em 1.9 Mb/s, valor este normal para a utilização da aplicação sem restrições, que é o caso até aos 30 s. Após 30 s, nota-se uma queda abrupta no valor da largura de faixa, devido ao início da restrição a 1 Mb/s. Durante os últimos 30 s, vê-se a aplicação a ajustar-se à limitação da largura de faixa. A média da largura de faixa, nos últimos 30 s, utilizada pela *stream* de vídeo do especialista é de 0.8 Mb/s, enquanto a do operador ronda os 0.9 Mb/s. Estes valores estão de acordo com a restrição a 1 Mb/s imposta.

A Figura 48 possibilita verificar o impacto que a restrição da largura de faixa a 1 Mb/s tem impacto na métrica RTT.

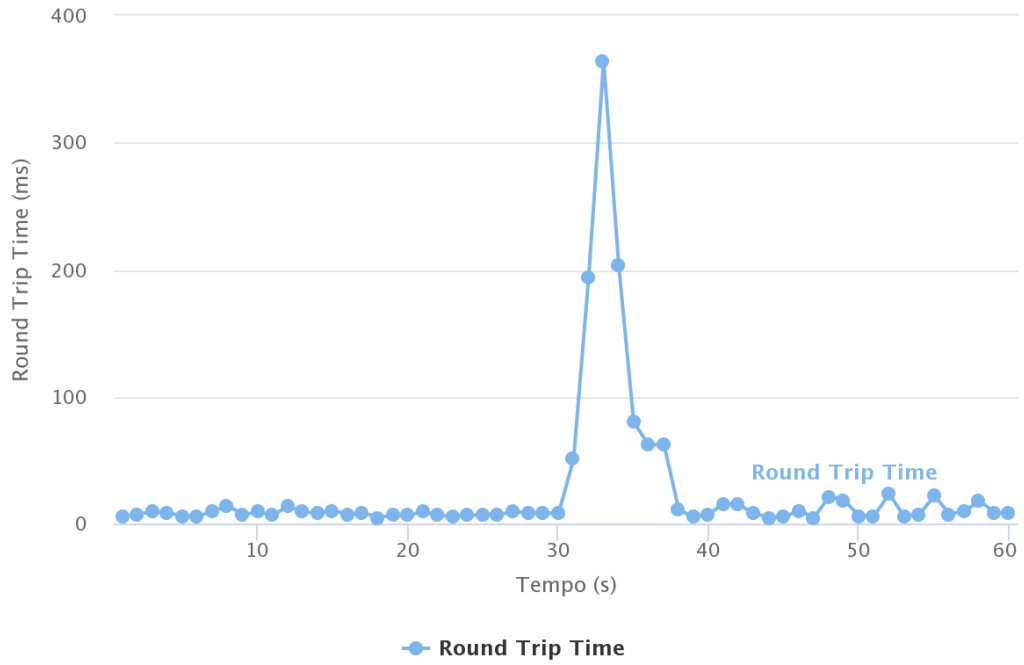


Figura 48 Análise da Aplicação Com Restrição – Largura de Faixa 1 Mb/s – Impacto no RTT

Na Figura 48 verifica-se a oscilação do RTT, a partir dos 30 s, momento em que foi iniciada a restrição na largura de faixa. Após cerca de 10 s, a aplicação conseguiu voltar a estabilizar o RTT, terminando com valores a rondar 8 ms.

Em termos de impacto na usabilidade da aplicação para o utilizador final, durante este teste não se notou qualquer problema na assistência remota, uma vez que os mecanismos do WebRTC conseguiram corrigir os problemas associados à restrição de Largura de Faixa. É, também, importante referir que não existiram perdas de pacotes nesta restrição.

De forma a estudar os limites de adaptabilidade da aplicação às restrições de largura de faixa de seguida é apresentada a Figura 49 que revela os resultados dos testes com restrição de largura de faixa a 512 kb/s.

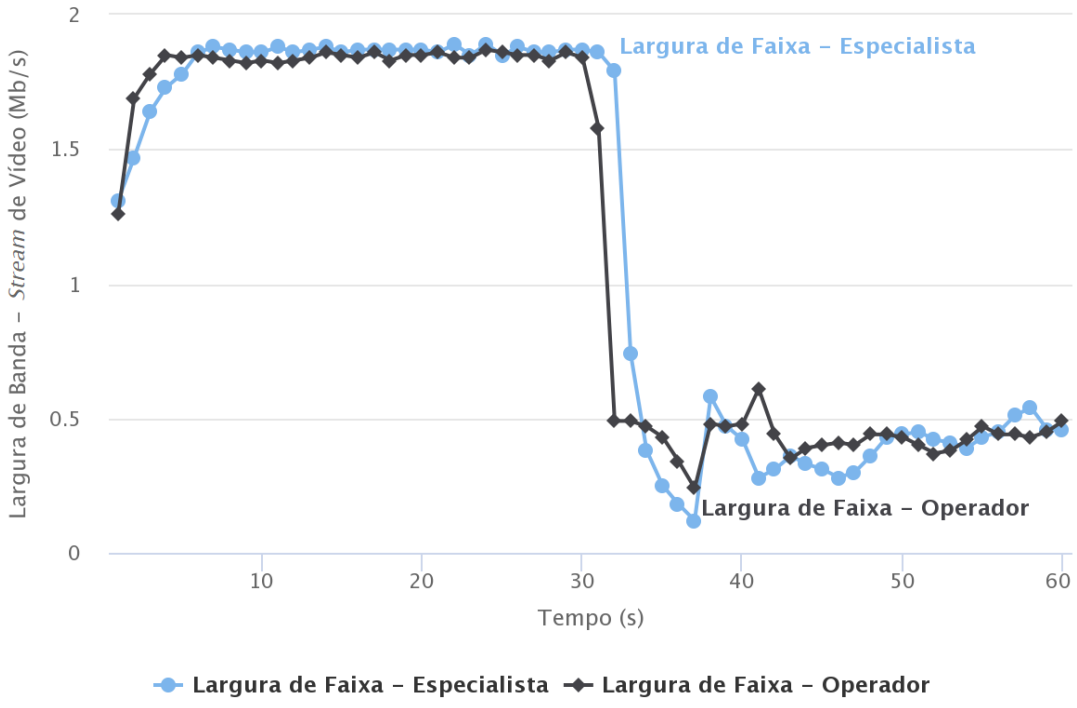


Figura 49 Análise da Aplicação Com Restrição – Largura de Faixa 512 kb/s

A Figura 49 apresenta resultados similares à Figura 47: até aos 30 s o uso de largura de faixa encontra-se estável e a partir dos 30 s há novamente queda abrupta da utilização da largura de faixa devido à restrição imposta. Neste caso, entre os 30 s e os 60 s verifica-se que os valores da largura de faixa rondam o limite de 512 kb/s. Em termos de médias nestes últimos 30 s, a *stream* de vídeo do especialista utilizou 490 kb/s e a do operador 469 kb/s, valores congruentes com a restrição imposta.

Quanto ao impacto desta restrição na métrica RTT, verifica-se um impacto similar ao da Figura 48. É de referir, que voltou a existir uma subida abrupta no valor do RTT, perto do segundo 30. Após cerca de 20 s, a aplicação conseguiu voltar a estabilizar o RTT, terminando com valores a rondar 8 ms.

No que toca ao impacto na usabilidade, durante este teste, verifica-se a existência de atraso no vídeo e áudio aquando da ativação da restrição, sendo dissipado rapidamente nos segundos seguintes. As perdas de pacotes são bastante reduzidas, ou seja cerca de 0.1 %.

Quanto ao teste relativo à restrição de largura de faixa que contempla um limite de 256 kb/s o resultado é exposto na Figura 50.

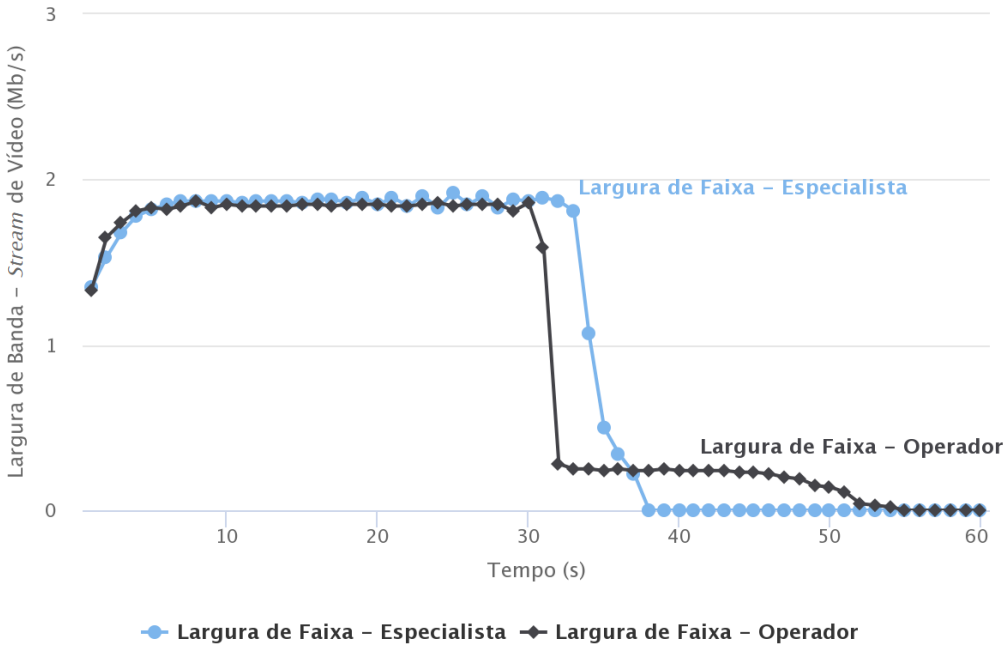


Figura 50 Análise da Aplicação Com Restrição – Largura de Faixa 256 kb/s

Com base na Figura 50 é possível detetar uma enorme diferença face às restrições de largura de faixa anteriores. Nas cinco iterações realizadas neste teste, o especialista desligou-se espontaneamente da videochamada após a ativação da restrição de largura de faixa. A partir dos 30 s verifica-se uma queda vertiginosa da largura de faixa utilizada. O especialista deixa de conseguir enviar a sua *stream* de vídeo, uma vez que sai da videochamada. O operador continua a enviar a *stream* de vídeo até receber a informação que o especialista remoto já não se encontra na videochamada.

No que diz respeito ao impacto desta restrição no RTT, não é apresentado qualquer gráfico uma vez que, no momento em que esta restrição é imposta, deixa de existir troca de pacotes entre os utilizadores, visto que o especialista sai da videochamada. Pelo mesmo motivo, a perda de pacotes é nula, uma vez que eles nem sequer chegam a ser enviados.

Quanto à perspetiva do utilizador da aplicação numa restrição desta magnitude, verifica-se que, no momento da ativação da restrição, há uma paragem do vídeo recebido e a saída do utilizador com limitações de largura de faixa, originando uma má experiência de utilização, visto nem ser possível realizar a assistência pretendida.

De forma a definir o valor mínimo de largura de faixa necessário, foram efetuados testes adicionais. Foi possível concluir que o valor mínimo de largura de faixa necessário é cerca de 380 kb/s. Com o valor de largura de faixa restrito a 375 kb/s, um dos utilizadores desliga-se espontaneamente da videochamada, tal como se sucedeu no teste com 256 kb/s. Com o valor de largura de faixa a 380 kb/s, nenhum dos utilizadores se desliga da videochamada, existindo apenas algum atraso na entrega dos pacotes, que não impossibilita a assistência remota.

Em resumo, a análise do uso da aplicação com restrição da largura de faixa permite concluir que cada utilizador deve ter, no mínimo, cerca de 380 kb/s de largura de faixa disponível para ter uma boa experiência de utilização. Na atualidade estes valores de largura de faixa estão disponíveis praticamente para qualquer utilizador, exceto para utilizadores que habitem num local remoto onde exista apenas *Internet* móvel. Nestes casos o requisito mínimo é recorrer a uma rede móvel de terceira geração uma vez que as redes de gerações anteriores não oferecem estes valores de largura de faixa.

Apresenta-se na Figura 51, a análise que contempla o aumento da latência nos pacotes enviados e recebidos pelo especialista remoto, recorrendo novamente à aplicação *softPerfect Connection Emulator*. Nos testes com esta restrição, também, são analisados os primeiros 60 s de uma assistência, sendo que a restrição é introduzida apenas aos 30 s.

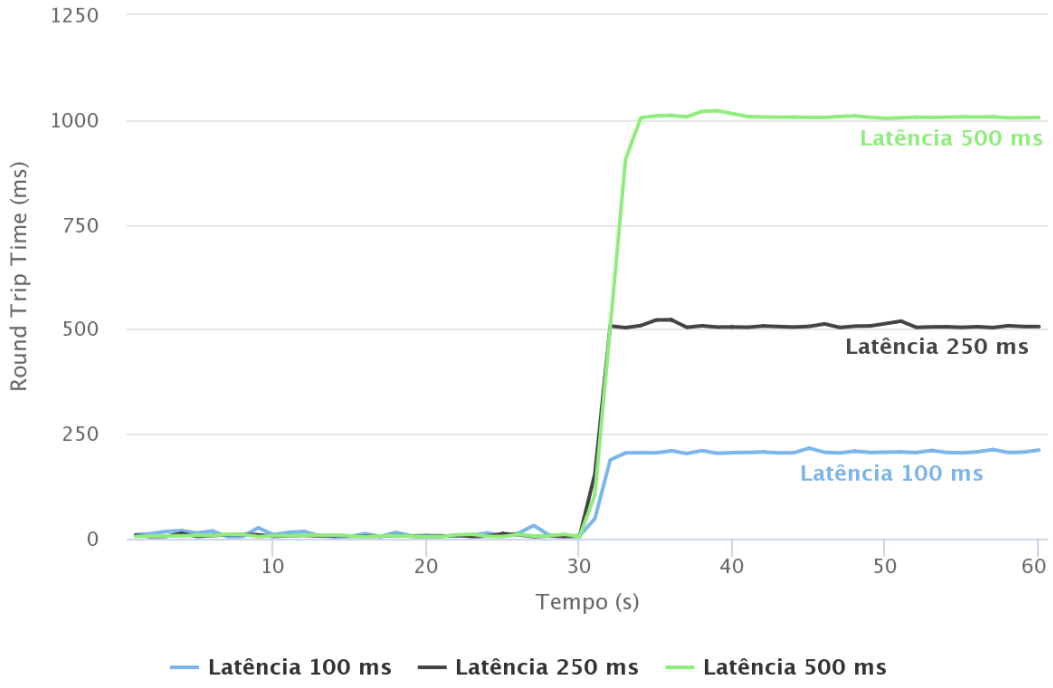


Figura 51 Análise da Aplicação Com Restrição – Adição de Latência

Nesta análise as latências introduzidas foram de 100 ms, 250 ms e 500 ms. Teoricamente, o valor do RTT é dobro da latência, tal facto é comprovado pelos resultados da Figura 51. Nos primeiros 30 s do gráfico apresentado verifica-se um RTT a rondar os 7 ms, valor comum à utilização da aplicação sem restrições. A partir dos 30 s, face à ativação da adição de latência, o valor de RTT subiu drasticamente em todos os testes, atingindo o dobro da latência adicionada. As médias do RTT rondam os 206 ms, 508 ms e 1004 ms para as adições de latência de 100, 250 e 500 ms, respetivamente.

Em termos de *front-end*, verifica-se o atraso na entrega da *stream* de vídeo e áudio, sendo mais notória quanto maior for a latência adicionada aos pacotes. Em termos de impacto para o utilizador final, o aumento de latência pode trazer complicações para a assistência remota, uma vez que os atrasos na entrega das *streams* podem dificultar a compreensão das instruções dadas, quer por voz, quer através das anotações de realidade aumentada. No caso da latência de 500 ms, é possível verificar um atraso considerável, capaz de prejudicar a assistência, nomeadamente, na adição precisa de anotações de RA. Conclui-se que o limite máximo de latência 400 ms definido pela ITU-T, previamente referido na secção WebRTC, deve ser respeitado para potencializar a experiência de utilização da solução de assistência remota desenvolvida.

Em suma, esta subsecção evidencia a importância das métricas e define os valores ideais de várias métricas resultantes da análise da aplicação sem restrições, que permitem atingir patamares elevados de usabilidade. As análises com restrições permitem definir um valor mínimo de largura de faixa 380 kb/s e o valor máximo de latência de 400 ms.

O facto do ambiente de testes ser local impede a generalização dos resultados. Não existiram os obstáculos de um ambiente de produção, elevada complexidade da rede, NAT *gateways* e *firewalls*, elevado tráfego, grandes distâncias entre utilizadores e reais limitações de largura de faixa. Considera-se, portanto, que de forma a melhorar a análise efetuada, deveriam ser realizados testes em ambientes de produção, permitindo credibilizar e impulsionar o uso da aplicação *web* de assistência remota desenvolvida num contexto de utilização real.

6. CONCLUSÕES

A assistência remota através de aplicações de voz e vídeo em tempo real é, cada vez mais, uma realidade para as empresas com canais de suporte e para o cidadão comum. O seu uso reduz o tempo consumido no processo de triagem e na resolução dos problemas.

Este projeto incidiu sobre a criação de uma solução de assistência remota com realidade aumentada via WebRTC. Inicialmente, foram identificados os requisitos do cliente e, depois, procedeu-se ao estudo das soluções existentes, académicas e proprietárias, analisados os protocolos de *streaming* para implementação de videochamadas e, por fim, investigadas bibliotecas de Realidade Aumentada (RA). A tecnologia WebRTC revolucionou este tipo de aplicações, tendo a sua adoção aumentado consideravelmente na última década. Neste trabalho, foram aliaram-se as potencialidades do WebRTC e da RA para a construção de uma aplicação *web* de assistência remota inovadora.

A especificação da solução proposta assentou na definição da arquitetura e casos práticos de uso alinhados com os requisitos do cliente e a análise de soluções existentes e tecnologias estudadas.

Quanto à implementação da solução, esta respeita os requisitos do cliente. Foi desenvolvida uma aplicação *web* multiplataforma, que pode ser utilizada via *browser*. Em termos de funcionalidades, providencia a realização da videochamada, através da criação da sala da videochamada e do convite de utilizadores. É possível ainda durante a videochamada, desligar e ligar o microfone e a câmara, partilhar o ecrã, etc.

O processo de implementação da aplicação contou ainda com o desenvolvimento de um protótipo. Este protótipo disponibiliza a funcionalidade inovadora de adição de anotações de realidade aumentada.

É de salientar que a solução, respeita premissas como a usabilidade e a responsividade, concedendo a possibilidade do uso da aplicação através de diversos dispositivos, desde computadores, a *tablets* e telemóveis. Desta forma, providencia uma melhor experiência ao utilizador.

A aplicação desenvolvida foi avaliada através dos testes de sistema realizados, onde foram testados os casos de uso através da simulação de um caso real, e da definição de métricas aplicadas num contexto de rede local. Os casos de uso implementados estão, totalmente, funcionais. Quanto à análise das métricas, permitiu definir os valores ideais e mínimos, para a largura de faixa e latência. Os resultados obtidos estão de acordo com os valores esperados para uma aplicação de tempo real, considerando os valores de *round trip time*.

A grande limitação da solução desenvolvida encontra-se na funcionalidade das anotações de RA, que utiliza o método Lucas-Kanade para o rastreio dos pontos associados às anotações. A limitação prende-se com o facto de só rastrear, eficazmente, vértices e pontos de contorno. Por exemplo, caso o especialista selecione um ponto para rastreio no centro de uma parede branca, o método não conseguirá rastrear esse ponto. Por consequência, o especialista deve selecionar pontos de zonas favoráveis e em conformidade com os requisitos do método, para assegurar um rastreio eficaz.

No que diz respeito a melhorias futuras, destacam-se o desenvolvimento da funcionalidade de *chat*, que poderá ser necessária em situações adversas, onde seja impossível a comunicação via áudio, a gravação da assistência remota efetuada e, por fim, a melhoria da funcionalidade das anotações de RA, através da inclusão de uma solução imersiva a três dimensões.

Referências Documentais

[1] Fang, Dikai & Xu, Huahu & Yang, Xiaoxian & Bian, Minjie. (2020). An Augmented Reality-Based Method for Remote Collaborative Real-Time Assistance: from a System Perspective. Mobile Networks and Applications. 25. 1-14. 10.1007/s11036-019-01244-4.

[2] Rasmussen, Troels & Grønbæk, Kaj. (2019). Tailorable Remote Assistance with RemoteAssistKit: A Study of and Design Response to Remote Assistance in the Manufacturing Industry. 10.1007/978-3-030-28011-6_6.

[3] Spitzer, Michael & Nanic, Ibrahim & Ebner, Martin. (2018). Distance Learning and Assistance Using Smart Glasses. Education Sciences. 8. 10.3390/educsci8010021.

[4] Aschauer, Andrea & Reisner-Kollmann, Irene & Wolfartsberger, Josef. (2021). Creating an Open-Source Augmented Reality Remote Support Tool for Industry: Challenges and Learnings. Procedia Computer Science. 180. 269-279. 10.1016/j.procs.2021.01.164.

[5] Obermair, F. & Althaler, J. & Seiler, U. & Zeilinger, P. & Lechner, A. & Pfaffeneder, L. & Richter, M. & Wolfartsberger, Josef. (2020). Maintenance with Augmented Reality Remote Support in Comparison to Paper-Based Instructions: Experiment and Analysis. 942-947. 10.1109/ICIEA49774.2020.9102078.

[6] "Cisco Webex Expert on Demand Solution Overview" [Online]. Disponível em: https://www.cisco.com/c/en/us/solutions/collateral/collaboration/webex-teams/solution-overview-c22-743893.html [Acedido a: 03-03-2021]

[7] "Embedding Webex Teams Meetings with the JavaScript SDK" [Online]. Disponível em: https://blog.webex.com/video-conferencing/embedding-%E2%80%AFwebex-%E2%80%AFteams-meetings-with-the-%E2%80%AFjavascript-%E2%80%AFsdk%E2%80%AF/ [Acedido a: 03-03-2021]

[8] "Cisco Webex Expert on Demand" [Online]. Disponível em: https://help.webex.com/en-us/kgk9he/Cisco-Webex-Expert-On-Demand#Cisco-Webex-Expert-on-Demand [Acedido a: 03-03-2021]

[9] "Overview of Dynamics 365 Remote Assist" [Online]. Disponível em: https://docs.microsoft.com/en-us/dynamics365/mixed-reality/remote-assist/ra-overview [Acedido a: 11-01-2021]

[10] "Descrição geral do Dynamics 365 Remote Assist para dispositivos móveis" [Online]. Disponível em: https://docs.microsoft.com/pt-pt/dynamics365/mixed-reality/remote-assist/mobile-app/remote-assist-mobile-overview [Acedido a: 11-01-2021]

[11] “Utilize Microsoft Teams no ambiente de trabalho virtual Azure” [Online]. Disponível em: https://docs.microsoft.com/pt-pt/azure/virtual-desktop/teams-on-avd [Acedido a: 16-10-2021]

[12] “IT troubleshooting at your fingertips with IBM Augmented Remote Assist” [Online]. Disponível em: https://www.ibm.com/downloads/cas/Y74R24OD [Acedido a: 10-03-2021]

[13] “IBM® Augmented Remote Assist Security White Paper” [Online]. Disponível em: https://www.ibm.com/downloads/cas/RDWO8LOX [Acedido a: 10-03-2021]

[14] “Vuforia Chalk: Assistência remota com tecnologia de realidade aumentada” [Online]. Disponível em: https://www.ptc.com/en/products/vuforia/vuforia-chalk [Acedido a: 11-03-2021]

[15] “Vuforia Chalk Technical architecture overview” [Online]. Disponível em: https://community.ptc.com/sejnu66972/attachments/sejnu66972/chalk/422/1/FAD_ChalkSecurityOverview%20Apr%202020.pdf [Acedido a: 11-03-2021]

[16] Hassan, N. A., & Hijazi, R. (2017). Future Trends. Data Hiding Techniques in Windows OS, 291–298. doi:10.1016/b978-0-12-804449-0.00008-7

[17] A. Dhamodaran, K. Gatimu and B. Lee, “Analysis of optimum substream lengths for dual TCP/UDP streaming of HD H.264 video over congested WLANs,” 2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, 2017, pp. 1-6, doi: 10.1109/CCNC.2017.8015366.

[18] “UDP vs. TCP and Which One to Use for Video Streaming” [Online]. Disponível em: https://www.wowza.com/blog/udp-vs-tcp [Acedido a: 14-01-2021]

[19] Divakaran, Dinil Mon & Murthy, Hema & Gonsalves, Timothy. (2006). Detection of Syn Flooding Attacks using Linear Prediction Analysis. 1. 1 - 6. 10.1109/ICON.2006.302563.

[20] “2019 Video Streaming Latency Report” [Online]. Disponível em: https://www.wowza.com/blog/2019-video-streaming-latency-report [Acedido a: 18-01-2021]

[21] X. Lei, X. Jiang and C. Wang, “Design and implementation of streaming media processing software based on RTMP,” 2012 5th International Congress on Image and Signal Processing, Chongqing, China, 2012, pp. 192-196, doi: 10.1109/CISP.2012.6469981.

[22] “RTMP Streaming: The Real-Time Messaging Protocol Explained” [Online]. Disponível em: https://www.wowza.com/blog/rtmp-streaming-real-time-messaging-protocol [Acedido a: 19-01-2021]

[23] A. Nurrohman and M. Abdurohman, "High Performance Streaming Based on H264 and Real Time Messaging Protocol (RTMP)," 2018 6th International Conference on Information and Communication Technology (ICoICT), 2018, pp. 174-177, doi: 10.1109/ICoICT.2018.8528770.

[24] “Adobe Flash Player EOL General Information Page” [Online]. Disponível em: https://www.adobe.com/pt/products/flashplayer/end-of-life.html [Acedido a: 19-01-2021]

[25] “What Is HLS (HTTP Live Streaming)?” [Online]. Disponível em: https://www.wowza.com/blog/hls-streaming-protocol [Acedido a: 22-01-2021]

[26] “HTTP Live Streaming” [Online]. Disponível em: https://developer.apple.com/documentation/http_live_streaming [Acedido a: 25-01-2021]

[27] “Understanding the HTTP Live Streaming Architecture” [Online]. Disponível em: https://developer.apple.com/documentation/http_live_streaming/understanding_the_http_live_streaming_architecture [Acedido a: 26-01-2021]

[28] Li Ling, Ma Xiaozhen and Huang Yulan, "CDN cloud: A novel scheme for combining CDN and cloud computing," Proceedings of 2013 2nd International Conference on Measurement, Information and Control, 2013, pp. 687-690, doi: 10.1109/MIC.2013.6758055.

[29] “MPEG-DASH: Dynamic Adaptive Streaming Over HTTP Explained” [Online]. Disponível em: https://www.wowza.com/blog/mpeg-dash-dynamic-adaptive-streaming-over-http [Acedido a: 08-02-2021]

[30] “Real-time communication for the web” [Online]. Disponível em: https://webrtc.org [Acedido a: 15-02-2021]

[31] G. Suciu, S. Stefanescu, C. Beceanu and M. Ceaparu, “WebRTC role in real-time communication and video conferencing,” 2020 Global Internet of Things Summit (GIoTS), Dublin, Ireland, 2020, pp. 1-6, doi: 10.1109/GIOTS49054.2020.9119656.

[32] “What Is WebRTC?” [Online]. Disponível em: https://www.wowza.com/blog/what-is-webrtc [Acedido a: 22-02-2021]

[33] “WebRTC Architecture” [Online]. Disponível em: https://webrtc.github.io/webrtc-org/architecture/ [Acedido a: 22-02-2021]

[34] N. M. Edan, A. Al-Sherbaz and S. Turner, "Design and evaluation of browser-to-browser video conferencing in WebRTC," 2017 Global Information Infrastructure and Networking Symposium (GIIS), 2017, pp. 75-78, doi: 10.1109/GIIS.2017.8169813.

[35] “Introduction to the Real-time Transport Protocol (RTP)” [Online]. Disponível em: https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/Intro_to_RTP [Acedido a: 14-09-2021]

[36] A. S. Ahson e M. Ilyas, “High Performance Browser Networking What every web developer should know about networking and web performance”, 1aed. O’Reilly Media, September 2013, Capítulo 18.

[37] “Can I use WebRTC?” [Online]. Disponível em: https://caniuse.com/?search=WebRTC [Acedido a: 25-02-2021]

[38] “WebRTC browser support on desktop and mobile” [Online]. Disponível em: https://bloggeek.me/webrtc-browser-support/ [Acedido a: 25-02-2021]

[39] International Telecommunication Union-Telecommunication, “End-user multimedia QoS categories”, SERIES G: TRANSMISSION SYSTEMS AND MEDIA, DIGITAL SYSTEMS AND NETWORKS; Quality of service and performance, 2001.

[40] “Why is WebRTC winning over its (non)competition?” [Online]. Disponível em: https://bloggeek.me/webrtc-winning-competition/ [Acedido a: 26-02-2021]

[41] O. Renius, “A Technical Evaluation of the WebXR Device API for Developing Augmented Reality Web Applications”, Dissertação, 2019.

[42] “WebXR Device API” [Online]. Disponível em: https://developer.mozilla.org/en-US/docs/Web/API/WebXR_Device_API [Acedido a: 10-05-2021]

[43] “WebXR Augmented Reality Module - Level 1” [Online]. Disponível em: https://immersive-web.github.io/webxr-ar-module/ [Acedido a: 11-05-2021]

[44] “three.js ar - paint” [Online]. Disponível em: https://threejs.org/examples/?q=paint#webxr_ar_paint [Acedido a: 12-05-2021]

[45] “WebXR Device API” [Online]. Disponível em: https://caniuse.com/?search=WebXR [Acedido a: 23-08-2021]

[46] “OpenCV Introduction” [Online]. Disponível em: https://docs.opencv.org/3.4/d1/dfb/intro.html [Acedido a: 15-05-2021]

[47] “Introduction to OpenCV.js and Tutorials” [Online]. Disponível em: https://docs.opencv.org/4.5.0/df/d0a/tutorial_js_intro.html [Acedido a: 16-05-2021]

[48] Procházka, David & Koubek, Tomas. (2011). Augmented Reality Implementation Methods in Mainstream Applications. Acta Universitatis Agriculturae et Silviculturae Mendelianae Brunensis. 59. 10.11118/actaun201159040257.

[49] “OpenCV Optical Flow” [Online]. Disponível em: https://docs.opencv.org/3.4/db/d7f/tutorial_js_lucas_kanade.html [Acedido a: 17-05-2021]

[50] “Express” [Online]. Disponível em: https://www.npmjs.com/package/express [Acedido a: 02-04-2021]

[51] “Socket.io” [Online]. Disponível em: https://www.npmjs.com/package/socket.io [Acedido a: 02-04-2021]

[52] “PeerJS: Simple peer-to-peer with WebRTC” [Online]. Disponível em: https://www.npmjs.com/package/peerjs [Acedido a: 03-04-2021]

[53] “UUID” [Online]. Disponível em: https://www.npmjs.com/package/uuid [Acedido a: 03-04-2021]

[54] “Embedded JavaScript templates” [Online]. Disponível em: https://www.npmjs.com/package/ejs [Acedido a: 03-04-2021]

[55] “MediaDevices.getUserMedia()” [Online]. Disponível em: https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/getUserMedia [Acedido a: 20-06-2021]