UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

DAVID PAVLAK PEREIRA

DESENVOLVIMENTO DE UM APLICATIVO PARA STREAMING DE CÂMERA DO DISPOSITIVO MÓVEL PARA USO COMO WEBCAM UTILIZANDO WEBRTC

PONTA GROSSA

2022

DAVID PAVLAK PEREIRA

DESENVOLVIMENTO DE UM APLICATIVO PARA STREAMING DE CÂMERA DO DISPOSITIVO MÓVEL PARA USO COMO WEBCAM

Development of an application for streaming camera from mobile device to use as a webcam using WebRTC

Trabalho de conclusão de curso de graduação apresentada como requisito para obtenção do título de Bacharel em Ciência da Computação da Universidade Tecnológica Federal do Paraná (UTFPR).
Orientador: Prof. Dr. Diego Roberto Antunes.

PONTA GROSSA

2022

DAVID PAVLAK PEREIRA

DESENVOLVIMENTO DE UM APLICATIVO PARA STREAMING DE CÂMERA DO DISPOSITIVO MÓVEL PARA USO COMO WEBCAM

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel da Universidade Tecnológica Federal do Paraná (UTFPR).

Data de aprovação: 8 de novembro de 2022

_____

Prof. Diego Roberto Antunes
Doutorado
Universidade Tecnológica Federal do Paraná – Campus Ponta Grossa

_____

Richard Duarte Ribeiro
Doutorado
Universidade Tecnológica Federal do Paraná – Campus Ponta Grossa

_____

Luiz Rafael Schmitke
Doutorado
Universidade Tecnológica Federal do Paraná – Campus Ponta Grossa

PONTA GROSSA

2022

Dedico este trabalho à minha família, pelos momentos de ausência.

AGRADECIMENTOS

Aos meus pais e irmãos, que sempre estiveram ao meu lado durante toda minha jornada, me incentivando nos momentos difíceis e compreendendo a minha ausência enquanto me dedica à realização deste trabalho.

A todos os professores do curso de Ciência da Computação pelas correções e ensinamentos que me permitiram apresentar um melhor desempenho no processo de formação profissional.

Ao meu orientador Prof. Dr. Diego Roberto Antunes, por ter desempenhado tal função com dedicação e amizade.

Aos amigos, que sempre estiveram ao meu lado, pela amizade incondicional e pelo apoio demonstrado ao longo de todo o período de tempo em que me dediquei a este trabalho.

A todos aqueles que contribuíram, de alguma forma, para a realização deste trabalho.

Muito obrigado.

RESUMO

A proposta de trabalho *home-office* (trabalho remoto) vem sendo cada vez mais ofertada pelas empresas, pois, beneficia não só os trabalhadores, mas também, os empregadores. Com a redução da taxa de desistência de trabalho, e consequentemente, o aumento da satisfação dos empregados, o aumento considerável na demanda é dado pelo grande número de trabalhadores que buscaram trocar o escritório de trabalho pelo conforto de casa, afirmando não ser mais necessário estar em um escritório presencialmente e em tempo integral para manter a produtividade. Além disso, a recente experiência epidêmica relacionada ao Covid-19 impulsionou ainda mais a procura pelo *home office*, que recentemente, segundo o IPEA (Instituto de Pesquisa Econômica Aplicada), alcançou sua alta de público. A quantidade de brasileiros que trabalharam de forma remota entre os meses de maio e novembro de 2020 chegou a 8,2 milhões, correspondendo a 11% dos 74 milhões de profissionais do país. Assim como no *home office,* várias instituições de ensino optaram pela educação remota, e, neste contexto, existe um crescimento e demanda considerável na tecnologia de *streaming* de vídeo e videoconferências, para que seja possível a realização de reuniões, aulas, encontros virtuais e demais atividades referentes à necessidade de contato remoto entre a população. O problema da pesquisa foi motivado pela baixa qualidade das *webcams* de *notebooks* e *webcams* externas, fazendo com que os usuários procurem outras formas de transmissão, por exemplo, os aplicativos e serviços para transmissão de câmera do *smartphone*. Porém, os aplicativos presentes no mercado geralmente são pagos, ou, a transmissão apresenta baixíssima qualidade, portanto, o objetivo deste trabalho foi desenvolver um sistema que faça a transmissão da câmera do *smartphone* assumindo o papel de *webcam* para o computador utilizando WebRTC.

Palavras-chave: streaming; transmissão em tempo real; Webrtc; Webcam.

ABSTRACT

The proposal for home-office(remote work) is increasingly being offered by the companies because it benefits not only the workers, but also the employers. With the reduction in the job dropout rate, and consequently, the increase in workers satisfaction, the considerable demand's increase is given by the large number of workers who sought to change the office for the comfort of home, affirming is no longer necessary to be in an office in person and full time to maintain productivity. In addiction, the recent epidemic experience related to Covid-19 has further boosted the demand for the home-office, wich recently, according to IPEA (Institute of Applied Economic Research) reached its highest number of people. Brazilians who worked remotely between the months of May and November 2020 reached 8.2 million people, corresponding to 11% of the 74 million professionals of the country. Just as the home office, several institutions of education choose for the remote education, and, in this context, there is a growth and considerable demand in streaming tecnology, so meetings can be held, classes, reunions, virtual meetings, and other activities referring to the remote contact needed between the population. The research problem was motivated because the quality of notebook webcams or even external webcams is low, causing users to look for other forms of transmission, for example, applications and services for smartphone camera transmission, however, applications and services for the transmission of camera from the Smartphone present in the market are usually paid, and the transmission has very low quality, therefore, the objective of the research was to develop a system to transmit the smartphone camera and assume the role of a webcam to the computer using WebRTC.

Keywords: Streaming; Real Time Streaming, WebRTC, WebCam.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
CPU	*Central Processing Units*
FPS	*Frames Per Second*
GB	*Gigabyte*
GHz	*Gigahertz*
GPU	*Graphics Processing Units*
ICE	*Interactive Connectivity Establishment*
IP	*Internet Protocol*
Mbps	*Megabits per Second*
MP	*Megapixel*
NAT	*Network Address Translation*
RTSP	*Real-Time Streaming Protocol*
SDP	*Session Description Protocol*
SO	Sistema Operacional
STUN	*Session Traversal Utilities for NAT*
TURN	*Traversal Using Relays around NAT*
WebRTC	*Web Real-Time Communication*

SUMÁRIO

1 INTRODUÇÃO

A crescente demanda pelo método de trabalho *home-office*(trabalho remoto) se dá pois, segundo a recente pesquisa "*Working from Home Around the World*" ("Trabalhando em casa ao redor do mundo", tradução própria), trabalhar de casa não só beneficia o empregado, mas também, melhora os resultados financeiros da empresa (AKSOY *et al.*, 2022).

Por parte dos empregados, os principais benefícios do *home-office* são, a não necessidade de deslocamento, o horário flexível, a economia de tempo ao se preparar para o trabalho e até mesmo o silêncio (AKSOY *et al.*, 2022). Por outro lado, por parte dos empregadores os principais motivos são a redução na taxa de desistência de trabalho e maior satisfação no trabalho (AKSOY *et al.*, 2022).

A adaptação e possível perduração deste método de trabalho é tão grande que, dentre os trabalhadores entrevistados em 27 países, cerca de 40% assumem desistir ou procurar outra vaga de trabalho caso seus respectivos empregadores exijam que eles voltem a trabalhar no escritório em tempo integral (AKSOY *et al.*, 2022).

Recentemente, a experiência epidêmica vivida pela polução mundial relacionada à COVID-19 (Coronavírus) acelerou consideravelmente a implantação de práticas de convivência à distância, partindo da necessidade de diminuir a propagação do vírus (SOUZA *et al.*, 2021). Estabelecimentos como instituições de ensino, comércios e áreas não essenciais buscaram manter-se fechados, sendo necessário o uso de práticas tais como o trabalho remoto (*home office*), ensino a distância, reuniões sociais, entre outras (SOUZA *et al.*, 2021).

Segundo o Instituto de Pesquisa Econômica Aplicada (IPEA), recentemente o Brasil alcançou o seu mais alto público trabalhando remotamente, chegando a 11% dos profissionais do país. Assim como no *home office*, várias instituições de ensino optaram pela educação remota e demais atividades que se comprometeram ao propósito de permanência obrigatória em casa (SOUZA *et al.*, 2021).

Além disso, existe um crescimento da demanda considerável na tecnologia de streaming de vídeo, para que seja possível a realização de reuniões, aulas, encontros virtuais e demais atividades remotas, tendo como base a demanda pela comunicação à distância (GONZÁLEZ *et al.*, 2017).

Porém, diversas pessoas não possuem *webcam* em casa, e para as que possuem, nem sempre apresentam uma boa qualidade. Além disso, a qualidade baixa de uma transmissão não só se dá pelo desempenho da câmera, mas também, pela troca de dados da rede utilizada, largura limitada de banda, eficiência energética, e diversos outros problemas (GONZÁLEZ *et al.*, 2017).

As transmissões entre aparelhos podem ser realizadas de diferentes formas e alguns fatores devem ser levados em consideração, tais como, o tipo de conteúdo transferido e as condições da rede (BEGEN; AKGUL; BAUGHER, 2011). Para transmissões em tempo real, o foco está na baixa latência, oscilação e transmissão eficiente dos dados que, por sua vez, tem a estrutura determinada pelos protocolos de transmissão (BEGEN; AKGUL; BAUGHER, 2011).

Apesar de existirem diversos protocolos de transmissão eles podem ser classificados em apenas dois grupos, "*push-based protocol*" ou "*pull-based protocol*" (BEGEN; AKGUL; BAUGHER, 2011).

Os protocolos baseados em empurrar dados (*push-based*) consistem em um servidor que transmite pacotes para um cliente até que o mesmo pare ou interrompa a sessão (GONZÁLEZ *et al.*, 2017).

Em contrapartida, protocolos baseados em puxar dados (*pull-based*) consistem em uma conexão entre cliente e servidor na qual o cliente envia uma requisição ao servidor e, dependendo da resposta, o servidor libera a transmissão para que o cliente efetue o *download* (BEGEN; AKGUL; BAUGHER, 2011).

Com o aumento da demanda de aplicações de videoconferência, e, aliado aos conceitos sobre transmissões em tempo real entre dispositivos, faz-se possível, com base nos estudos contidos neste documento, desenvolver um sistema para realizar a transmissão da câmera do *Smartphone* assumindo o papel de webcam para o computador utilizando um protocolo de transmissão em tempo real.

1.1 Problema e justificativa

Com o intuito de auxiliar neste problema de qualidade ou falta de *webcam*, surgiram alguns aplicativos para *smartphones* que usam a câmera e transmitem a imagem simulando uma *webcam*. Porém, os aplicativos e serviços para transmissão de vídeo via câmera de smartphone presentes no mercado geralmente são pagos, ou, a transmissão apresenta baixa qualidade. Como exemplo fez-se uma análise rápida

do aplicativo EpocCam[1], que possui diversas melhorias de desempenho na sua versão paga em comparação com a versão gratuita. Segundo dados retirados do próprio site[2] da desenvolvedora, como por exemplo, em sua versão gratuita, o aplicativo assume uma resolução de 640x480(480p) para transmissão, enquanto na sua versão paga passa a alcançar uma resolução de 1920x1080(1080p) assumindo resolução de alta definição. Não apenas em termos de resolução, na sua versão paga o aplicativo desbloqueia funções tais como o uso do microfone do *smartphone*, foco manual, ajuste na qualidade do vídeo, entre outras funcionalidades.

Portanto, partindo da hipótese de que a maioria dos *smartphones* atuais possuem uma câmera que entrega uma melhor qualidade do que uma *webcam* tradicional, a presente pesquisa visa desenvolver um aplicativo de código aberto e gratuito para auxiliar os usuários a melhorar a qualidade nas transmissões de vídeo em tempo real.

1.2 Objetivos

O objetivo geral deste trabalho é desenvolver um sistema web e móvel para uso da câmera do *smartphone* como *webcam* no computador, sendo dividido nos seguintes objetivos específicos:

1.2.1 Objetivos específicos

- Analisar a arquitetura dos softwares similares já existentes no mercado.
- Analisar protocolos de transmissão de vídeo em tempo real e suas aplicações.
- Desenvolver a aplicação cliente(*smartphone*)-servidor(*desktop*).
- Avaliar a aplicação proposta em diferentes sistemas de videoconferências e criar diferentes cenários de comparação.

1.3 Estrutura do trabalho

Este trabalho foi estruturado com o intuito de facilitar a compreensão da pesquisa, sendo dividido em 5 capítulos.

---

[1] https://www.elgato.com/en/epoccam
[2] https://help.elgato.com/hc/en-us/articles/360048912471-EpocCam-Difference-Between-Free-and-Pro-Versions-

No Capítulo 1 encontra-se a introdução, buscando introduzir o leitor ao contexto da pesquisa e aos objetivos.

O Capítulo 2 apresenta os trabalhos relacionados que condizem com a proposta do projeto, bem como estruturas e comparações entre eles.

No Capítulo 3 apresenta-se dados mais aprofundados sobre o mercado de videoconferência e *live streaming*, além de conceitos importantes sobre transmissões entre dispositivos em tempo real junto a apontamentos sobre câmeras virtuais e validação de aplicações.

O capítulo 4 discorre sobre a metodologia utilizada na solução, apresentando as ferramentas utilizadas e as metodologias de teste.

O Capítulo 5 engloba todo o desenvolvimento do sistema em questão, partindo do levantamento de requisitos, prosseguindo para a arquitetura da aplicação e por fim o desenvolvimento do sistema cliente-servidor.

No Capítulo 6 encontram-se os resultados, estes, sendo obtidos por meio de três diferentes metodologias. A primeira consiste em uma análise técnica da solução. A segunda, um experimento de desempenho da solução em uma videoconferência na qual diferentes *webcams* e outras soluções já existentes no mercado são apresentadas aos participantes da chamada junto ao projeto com o intuito de colher *feedbacks*. E a terceira, a aplicação de um teste as cegas, onde, diferentes cenários de transmissão transmitem o mesmo vídeo para que seja feita uma análise comparativa.

Por fim, no capítulo 7 apresentam-se as considerações finais e conclusões referente à pesquisa.

2 SISTEMAS SIMILARES

Já existem no mercado opções de aplicativos para usuários que desejam utilizar a câmera do *smartphone* como *webcam*. Contudo, um problema encontrado no decorrer da pesquisa é que, alguns desses aplicativos restringem funcionalidades e impõem limitações de uso para usuários da versão gratuita, havendo necessidade de pagar para utilizar o aplicativo sem nenhum tipo de limitação e desfrutar de uma experiência agradável.

Há limitações de funcionalidades como foco manual da câmera, uso de microfone, tipo de conexão utilizada e *flashlight*, contudo, o que mais chama a atenção é o fato de que grande parte dessas limitações são relacionadas a qualidade da transmissão, sendo um dos fatores principais pela busca do aplicativo.

A seguir são apresentados quatro aplicativos que têm o comportamento citado anteriormente, sendo introduzidos pelo seu funcionamento e características, seguindo de uma tabela demonstrando as diferenças entre suas versões pagas e gratuitas.

O EpocCam[1] é um aplicativo restrito para usuários de smartphones com o sistema operacional *iOS*, dando suporte para os sistemas operacionais *Windows* (versão 10 ou superior) e *MacOS* (versão 10.14 ou superior). Distribuído pela empresa *ElGato*, o aplicativo promete transformar os *iPhones* em verdadeiras *webcams* de altíssima qualidade e de maneira simples, bastando que o *smartphone* e o computador estejam conectados via cabo ou rede *Wi-Fi.* Com fácil utilização em plataformas de videoconferência como *Zoom*, *Microsoft Teams*, entre outras, o aplicativo ficou bastante conhecido por ser o primeiro a adicionar filtros na câmera controlados por gestos, sendo possível pintar o próprio cabelo ou interagir com personagens animados durante a transmissão (ELGATO, 2021).

Uma outra opção é o aplicativo Camo[3] distribuído pela empresa "*Reincubate*", sua versão para sistemas operacionais *Android* ainda está em desenvolvimento. O aplicativo tem suporte para os sistemas operacionais *iOS* (versão 10.13 ou superior) e o sistema operacional *Windows* (versão 7 de 64 bits ou superior). Obtendo suporte direto para aplicativos com foco em videoconferência como *Zoom*, *Google Meet*, *Microsoft Teams*, *Skype*, *Slack* e muitos outros. Um dos principais pontos do sistema é a utilização da função "*plug and play*", não sendo necessário nenhum tipo de

[3] https://reincubate.com/pt/camo/

software extra para o funcionamento do mesmo, além do aplicativo instalado no *smartphone* e no computador (CAMO, 2021).

O aplicativo DroidCam[4] distribuído por "*Dev47Apps*", inicialmente foi um projeto exclusivo para os sistemas operacionais *Android*, porém, expandiu seu desenvolvimento para sistemas iOS. Prestando suporte para a mesma gama de plataformas apresentadas nos aplicativos anteriores, os requisitos para utilização são *Android* (versão 5.0 ou superior) e *Windows* (versão 10 ou superior). Destaca-se nos dispositivos *Android* por permitir minimizar os aplicativos sem que a transmissão pause ou termine (DEV47APPS, 2021).

O aplicativo iVCam[5] distribuído pela empresa "*e2eSoft*" é desenvolvido para ambas as plataformas. Tem suporte para os sistemas operacionais iOS (versão 10.0 ou superior), sistema operacional Android (versão 9 ou superior), e, o sistema operacional Windows (versão 7 de 64 bits ou superior). Assim como as aplicações anteriores, o iVCam presta suporte para plataformas de videoconferência como Zoom, Google Meet, Skype, entre outras. Prometendo diversas funcionalidades, seus pontos fortes são a alta qualidade de vídeo com baixa latência, além de poder conectar diversos aparelhos em um único computador ao mesmo tempo. Outra funcionalidade bastante destacada é o fato de poder minimizar as aplicações sem interrupções na transmissão, assim como o DroidCam[4] (E2ESOFT, 2022).

A seguir é apresentado o Quadro 1, com um comparativo entre funcionalidades dos aplicativos supracitados em suas versões pagas e gratuitas.

[4] https://droidcam.softonic.com.br
[5] https://www.e2esoft.com/ivcam

Quadro 1 – Restrição de funcionalidades em aplicativos gratuitos

Funcionalidades	**EpocCam**		**Camo**		**DroidCam**		**iVCam**	
	Gratuito	Pago	Gratuito	Pago	Gratuito	Pago	Gratuito	Pago
Resolução máxima de vídeo	640x 480 (480p)	1920x 1080 (1080p)	1280x 720 (720p)	1440x 1080 (1080p)	640x 480 (480p)	1920x 1080 (1080p)	640x 480 (480p)	1920x 1080 (1080p)
Conexão USB	**Sim**	**Sim**	**Sim**	**Sim**	**Sim**	**Sim**	**Sim**	**Sim**
Diferentes tipos de conexão (USB, WiFi, NDI)	**Não**	**Sim**	**Não**	**Não**	**Sim**	**Sim**	**Não**	**Sim**
Utilização da Câmera frontal	**Sim**	**Sim**	**Sim**	**Sim**	**Sim**	**Sim**	**Sim**	**Sim**
Foco manual	**Não**	**Sim**	**Não**	**Sim**	**Não**	**Sim**	**Sim**	**Sim**
Zoom manual	**Não**	**Sim**	**Não**	**Sim**	**Não**	**Sim**	**Sim**	**Sim**
Marca d'água na transmissão	**Sim**	**Não**	**Sim**	**Não**	**Não**	**Não**	**Sim**	**Não**
Flashlight	**Não**	**Sim**	**Não**	**Sim**	**Não**	**Sim**	**Sim**	**Sim**
Utilização de múltiplas câmeras	**Não**	**Sim**	**Não**	**Sim**	**Não**	**Não**	**Não**	**Sim**
Utilização do microfone	**Não**	**Sim**	**Sim**	**Sim**	**Sim**	**Sim**	**Sim**	**Sim**

Fonte: Adaptado de Dev47apps (2021), Elgato (2021), Camo (2021), E2esoft (2022)

3 REFERENCIAL TEÓRICO

Este capítulo discute sobre o crescimento e aumento da demanda do mercado de videoconferência e *streaming*, os meios de transmissão, problemas e características importantes para uma boa qualidade de transmissão de vídeo, além de apontamentos sobre câmeras virtuais e métodos de validação de aplicações.

As seções deste capítulo, além de discutir sobre o crescimento do conceito, também abordam temas como: a qualidade de imagem, a necessidade de um investimento alto para utilizar produtos que apresentam boa qualidade e a possível solução do problema sendo utilizar a câmera do celular como *webcam*, posteriormente, são explorados os protocolos de transmissão de vídeo em tempo real, sendo essenciais para realizar uma transmissão entre aparelhos.

O capítulo segue explorando conceitos de câmeras virtuais, capazes de emular uma *webcam*, além de, explorar um método de avaliação de aplicações.

Por fim, finaliza-se o capítulo com a considerações necessárias.

3.1 Mercado de videoconferência e *streaming*

Em constante evolução, uma das formas de comunicação mais populares e utilizadas nos dias de hoje é o *streaming* de vídeo, uma ótima alternativa para reuniões presenciais, trabalho remoto, aulas *online*, entre outros (GONZÁLEZ *et al.*, 2017).

O desejo dos consumidores de acessar quantidades praticamente ilimitadas de conteúdo a qualquer hora do dia e independente do lugar em que eles estejam, acelera a implantação de serviços de *streaming* (BEGEN; AKGUL; BAUGHER, 2011).

As plataformas de *streaming* como Youtube[6], Twitch.tv[7] e FaceBookGaming[8], vem ganhando muito espaço no mercado utilizando a transmissão de jogos ao vivo. As transmissões ao vivo de pessoas jogando jogos começaram a crescer em popularidade por volta de 2009, tendo mais de 34 milhões de espectadores únicos mensais (HAMILTON; GARRETSON; KERNE, 2014). A Twitch[7] é um dos nomes mais populares quando se fala sobre serviço de *streaming* de vídeo, tal qual serve como local de encontro para as comunidades de jogadores (HAMILTON; GARRETSON; KERNE, 2014).

[6] https://www.youtube.com
[7] https://www.twitch.tv
[8] https://www.facebook.com/gaming/

Os *streamers* na maioria das vezes transmitem jogos digitais sozinhos ou em conjunto com amigos. Além de um canal de comunicação baseado em texto para interação do público, os *streamers* também incorporam as imagens de sua própria *webcam* na transmissão, facilitando o envolvimento do espectador para criar um meio social exclusivo, dando ênfase a construção da comunidade (HAMILTON; GARRETSON; KERNE, 2014).

De acordo com o Instituto de Pesquisa Econômica Aplicada (GÓES; MARTINS; NASCIMENTO, 2021), recentemente o Brasil alcançou sua alta de público trabalhando remotamente, na qual, entre os meses de maio e novembro de 2020 chegou a 8,2 milhões de pessoas, correspondendo a 11% dos 74 milhões de profissionais que continuaram a trabalhar durante a pandemia de COVID-19. Pesquisa feita com base nos dados do Instituto Brasileiro de Geografia e Estatística (IBGE).

Segundo a CNN Brasil, a busca por aplicativos de videoconferência disparou em 2020 por causa das medidas de distanciamento social.

> Sem a possibilidade de atender clientes presencialmente, empresas brasileiras se adaptaram às videoconferências. Se antes quase tudo era feito de forma presencial, agora boa parte do serviço oferecido está no ambiente virtual. Com medidas de isolamento social implementadas pelo mundo, os funcionários tiverem que se adaptar à rotina de trabalho em casa, e a procura por aplicativos de videoconferência disparou (RIBEIRO; WELLS, 2021).

Assim como no mercado de trabalho, a maioria de entidades relacionadas a educação após serem expostas ao cenário epidêmico recorreram ao ensino à distância (MAHFOODH; ALATAWI, 2021). Mesmo sendo uma solução à curto prazo para sustentar a sistema educacional, grande parte dos órgãos de educação planejam manter o ambiente de educação remota mesmo pós pandemia, alegando uma melhor gestão econômica (MAHFOODH; ALATAWI, 2021).

O alto crescimento na demanda das videoconferências expõe ainda mais um outro problema: a qualidade da tecnologia de vídeo. Se a experiência de usar uma videoconferência para reuniões for ruim e as pessoas não precisarem usá-la no futuro, elas não usarão (LOGITECH, 2021). Em contrapartida, se o vídeo e som apresentarem uma boa qualidade, a experiência tende a ser satisfatória e mais perto da realidade presencial, portanto, atrairá mais pessoas para usá-la (LOGITECH, 2021). E é por isso que uma *webcam* faz uma diferença tão grande no engajamento e interação pelo público (LOGITECH, 2021).

Com este cenário construído, câmeras de todos os tipos foram colocadas à prova, sendo elas: câmeras integradas, câmeras externas e até câmeras de dispositivos móveis (MARQUES, 2020).

As *Webcams* integradas não costumam variar em qualidade, a grosso modo, elas quase estão paradas no tempo (MARQUES, 2020). Mesmo nos modelos mais recentes de *notebooks*, é difícil encontrar um que possua *webcam* integrada com uma boa qualidade. A *Apple*, uma das maiores marcas do mercado quando se diz respeito a *notebooks*, em seus modelos "*MacBook Pro 16"* e "*MacBook Air"* lançados em 2020 contam com câmeras *FaceTime HD*, câmeras essas que fazem parte da linha de *notebooks* da empresa desde 2012 e atingem resoluções máximas de 720p (MARQUES, 2020).

Uma solução do problema seria a utilização de câmeras externas ou *webcams* profissionais, porém, ao analisar uma *webcam* como por exemplo a c270 distribuída pela *Logitech* que apresenta um valor relativamente baixo de mercado, nota-se que a mesma dispõe de uma resolução de 3 *megapixels* (com resolução de 1280 x 720) (BRINGIT, 2020). É possível encontrar opções ainda mais baratas, porém, com qualidade ainda inferior, como é o caso do modelo "WC045" distribuído pela *Multilaser,* utilizando uma resolução de 1.5 *megapixels* (BRINGIT, 2020). Por outro lado, uma *webcam* profissional como a *StreamCam* distribuída pela *Logitech,* promete entregar um serviço de qualidade *Full HD,* mas, em contrapartida, tem seu valor de mercado relativamente alto (LOGITECH, 2021).

Com um grande ponto positivo no contexto, tem-se os *smartphones*, já que, atualmente a maioria desses dispositivos contam com ótimas câmeras, com imagens bonitas e com boa resolução (MARQUES, 2020).

Para efeitos de comparação, a mesma empresa citada a cima (*Apple*) antes do lançamento dos modelos de *notebook* "*MacBook Pro 16"* e "*MacBook Air"* em 2020, realizou o lançamento do *smartphone "iPhone 11"* em 2019. Atenta-se ao fato de que, os *notebooks* lançados em 2020 contam com a câmera *FaceTime HD* de 720p que é que é uma webcam de 1.2 *megapixels* (com resolução de 1.280x720 *pixels*), enquanto que o *smartphone* lançado em 2019 possui uma câmera de 12 *megapixels* (resolução de 4.000x3.000 *pixels*), notando a constante evolução nas tecnologias das câmeras dos smartphones, e por outro lado, a falta de evolução nas câmeras dos *notebooks* (HINER, 2020).

Com o baixo desempenho das *webcams* integradas e para poupar o dinheiro gasto ao comprar uma *webcam* dedicada com o intuito de melhorar a experiência das videoconferencias, é possível utilizar a câmera do *smartphone* como *webcam* no computador (VYAS, 2021).

3.2 Protocolos de transmissão de vídeo

As transmissões de conteúdos entre diferentes aparelhos em uma rede podem ser realizadas de diferentes formas, porém, alguns fatores devem ser levados em consideração para que sejam determinados métodos capazes de realizar a comunicação da maneira correta. O tipo de conteúdo transferido e as condições da rede são aspectos relevantes (BEGEN; AKGUL; BAUGHER, 2011).

Para uma transferência simples de arquivos em uma rede que possui perdas, adicionar redundância nos pacotes funciona como uma proteção extra para garantir a entrega do conteúdo (BEGEN; AKGUL; BAUGHER, 2011).

Por outro lado, para transmissões em tempo real o foco está na baixa latência, oscilação e transmissão eficiente dos dados, de modo que, para esse conjunto de tratativas é necessário e mais vantajoso utilizar um protocolo de transmissão específico (BEGEN; AKGUL; BAUGHER, 2011).

Um protocolo de transmissão consiste em diversos algoritmos, ou regras, capazes de manipular os dados transmitidos pela rede, dividindo-os em diferentes partes, como cabeçalhos, dados, autenticação e tratamento de erros, proporcionando uma transmissão de informações mais robusta e confiável (GONZÁLEZ *et al.*, 2017).

Mesmo que seja difícil escolher um protocolo de transmissão ideal, todo e qualquer protocolo de transmissão existente pode ser classificado em apenas dois grupos, “*push-based protocol*” ou “*pull-based protocol*”, sendo eles, protocolos baseados em empurrar dados ou protocolos baseados em puxar dados, respectivamente (BEGEN; AKGUL; BAUGHER, 2011).

3.2.1 Arquitetura de protocolos *Pull-Based*

Os protocolos baseados em puxar dados (*pull-based*) são dependentes da largura de banda, visto que, consistem em um uma conexão entre cliente e servidor, na qual o cliente envia uma requisição ao servidor e, dependendo da resposta, o

servidor libera a transmissão para que o cliente efetue o *download* (GONZÁLEZ *et al.*, 2017).

Esta arquitetura de transmissão divide o conteúdo de mídia em diversos pacotes e aloca-os em segmentos(nós) para que cada nó selecione, de maneira independente, seus vizinhos e periodicamente notifique-os sobre quais pacotes ele possui, para que posteriormente cada nó solicite apenas os pacotes de seu interesse com base nas notificações dos seus respectivos vizinhos, formando uma rede não estruturada (ZHANG *et al.*, 2007). Com as vantagens de simplicidade e robustez, protocolos *pull-based* são comumente utilizados em sistemas P2P(ponto-a-ponto), onde cada ponto é um participante da transmissão e também um servidor, para manter o sistema mais robusto e funcional (ZHANG *et al.*, 2007).

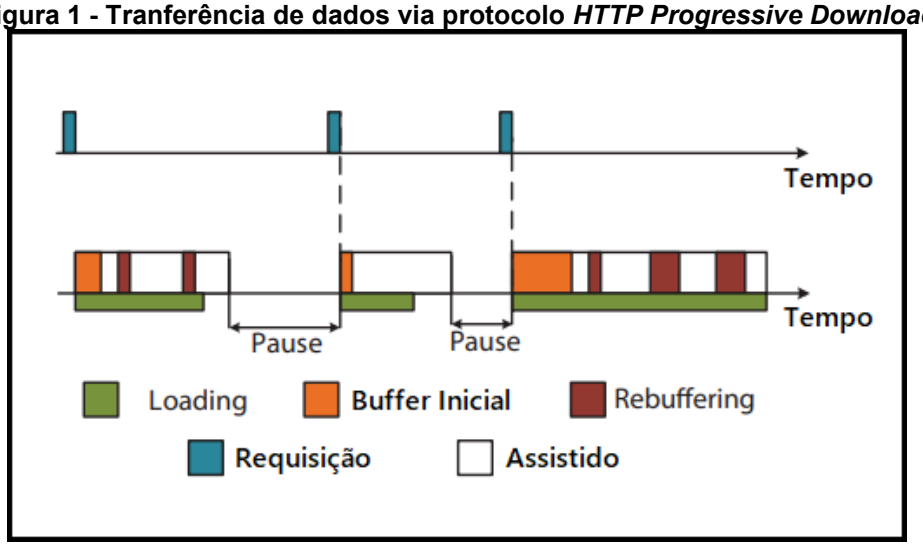
O *HTTP Progressive Download* é um protocolo bastante popular desta arquitetura, estando presente em aplicações mundialmente conhecidas e utilizadas como o *YouTube* durante muitos anos (PEREIRA; PEREIRA, 2014).

Uma transmissão utilizando esse protocolo é iniciada quando o cliente seleciona um vídeo e uma resolução, enviando uma solicitação para o servidor (PASTUSHOK; TURLIKOV, 2016). Então o servidor reconhece essa solicitação, encontra o vídeo e a resolução requisitada, uma vez que, cada diferente resolução de um mesmo vídeo possui uma taxa de bits exclusiva (PASTUSHOK; TURLIKOV, 2016). Após esse processo, o servidor começa a enviar dados para o equipamento do usuário, que por sua vez preenche um *buffer* para que, quando os dados armazenados em *buffer* forem suficientes, a reprodução do vídeo seja iniciada, esse intervalo entre solicitação e início da reprodução do vídeo é chamado de *Buffer* Inicial (PASTUSHOK; TURLIKOV, 2016).

Se uma taxa de transferência for menor do que uma taxa de bits de reprodução, ou seja, se o usuário selecionar uma resolução de vídeo que demanda uma taxa de bits maior do que a sua respectiva largura de banda, o cliente irá enfrentar um fenômeno conhecido como *Rebuffering*, onde acontecerá uma paralisação na representação do vídeo já que o equipamento do usuário está drenando o *buffer* mais rápido do que preenchendo, portanto quando vazio, o cliente precisa enchê-lo outra vez (PASTUSHOK; TURLIKOV, 2016).

A figura 1 apresenta as etapas de transferência deste protocolo.

Figura 1 - Tranferência de dados via protocolo *HTTP Progressive Download*



Fonte: Adaptado de Pastushok e Turlikov (2016)

Mesmo que bastante usado e consistente, o protocolo apresenta problemas, como o fato de um arquivo ser baixado completamente após o início da transferência (PEREIRA; PEREIRA, 2014). Como exemplo, suponha que um vídeo contenha 10 minutos de reprodução e o usuário assistiu apenas os primeiros 2 minutos, mesmo que o usuário feche o vídeo, o arquivo inteiro era entregue pelo servidor. Os servidores de vídeo entregavam quantidades enormes de dados, que quando não assistidos, estavam sendo entregues em vão e sendo um grande problema para os usuários de *smartphones*, onde o *streaming* de vídeo tornou-se um dos aplicativos móveis mais utilizados consumindo grande parte do tráfego nas redes móveis que possui uma largura de banda limitada (PAN; CHENG, 2018).

Em 2013, apresentado pela *Google*, O protocolo *Adaptive Streaming* tem se tornado a melhor opção para servidores de vídeo principalmente para distribuição em redes móveis (PAN; CHENG, 2018). Utilizado por companhias como *Netflix*, o *YouTube* também passou a utilizar a tecnologia (PAN; CHENG, 2018). Com total influência do *Progressive Download*, a diferença entre as tecnologias é que o *Adaptive Streaming* trabalha com requisições por blocos enquanto o usuário estiver conectado ao vídeo, reduzindo o consumo de dados do servidor e trabalhando de maneira mais eficiente em redes com baixa largura de banda (BEGEN; AKGUL; BAUGHER, 2011). Outro ponto importante é o ajuste dinâmico na taxa de *bits*, buscando adaptar-se as condições da rede do cliente, a tecnologia monitora o consumo de dados e se adapta de acordo com as mudanças na rede, diminuindo a taxa de bits e abaixando a qualidade da reprodução do vídeo se necessário, evitando problemas como

congelamento da transmissão e, posteriormente, um *Rebuffering* (PAN; CHENG, 2018).

3.2.2 Arquitetura de protocolos *Push-Based*

Os protocolos baseados em empurrar dados (*push-based*) consistem em um servidor que transmite pacotes para um cliente até que o mesmo pare ou interrompa a sessão, portanto, o servidor mantém uma conexão com o cliente recebendo comandos e esperando alterações na sessão (GONZÁLEZ *et al.*, 2017). Protocolos do tipo "*push*" geralmente utilizam transporte de dados em tempo real (GONZÁLEZ *et al.*, 2017).

Para que este tipo de transmissão ocorra, é importante ressaltar que, a taxa de *bits* com que o servidor entrega os pacotes para o cliente, precisa corresponder a taxa de *bits* de consumo do cliente, garantindo que os níveis de *buffer* do cliente permaneçam estáveis, ou seja, é necessário otimizar o uso da rede para que o cliente não consuma uma taxa de *bits* maior do que a quantidade disponibilizada, e por outro lado, entregar mais *bits* do que o necessário irá sobrecarregar a rede (BEGEN; AKGUL; BAUGHER, 2011).

Um dos mais famosos e utilizados protocolos da categoria *push* é o RTSP, ou, protocolo de transmissão em tempo real (BEGEN; AKGUL; BAUGHER, 2011). Esse tipo de protocolo permite que o servidor empurre para o cliente os pacotes necessários, sendo uma ótima opção para transmissões de baixa latência (BEGEN; AKGUL; BAUGHER, 2011).

A seguir são apresentadas as funcionalidades de cada método utilizado pelo RTSP com base em (SCHULZRINNE; RAO; LANPHIER, 2016).

- Descrição: O primeiro método utilizado pelo RTSP é o "*describe*", tal qual o cliente envia uma requisição ao servidor e o mesmo responde com uma descrição do conteúdo requisitado.
- Configurar: Após o "*describe*", o método "*setup*" é chamado para estabelecer como o fluxo será transportado, ou seja, estabelecer a porta para qual os dados de vídeo e áudio vão ser recebidos e outra para os metadados. A resposta do servidor é quem gera essa configuração e ela deve existir antes do método "*play*".
- Iniciar: Por sua vez, inicia o envio dos dados.

- Pausar: O método “*pause*” irá pausar todas as transmissões atuais, e pode ser revertido posteriormente utilizando um método “*play*”.
- Destruir: Por fim é chamado o método “*teardown*”, que interrompe toda e qualquer transmissão de dados atualmente utilizada.

3.2.3 Protocolo SDP

O SDP é encarregado de formalizar uma oferta ou resposta para uma sessão multimídia, que é enviado para um ou mais pontos de uma rede em um formato compreendido pelos participantes presentes na rede (TUTORIALSPOINT, 2022). Os pontos da rede que recebem essa informação, decidem se desejam participar da sessão ou quando e como ingressar na sessão desejada (TUTORIALSPOINT, 2022).

Por mais que o SDP não incorpore de fato um protocolo de transporte, ele é destinado a usar diferentes protocolos de transporte conforme a necessidade, como por exemplo o protocolo UDP (BEGEN *et al.*, 2021). Tendo uma entrega mais rápida e menos confiável, o UDP é interessante em comunicações sensíveis ao tempo, incluindo voz sobre IP (VoIP) e reprodução de vídeo ou áudio (ROSENCRANCE; MOOZAKIS; LATON, 2021).

A oferta SDP contém informações como o nome e finalidade da sessão, hora(s) em que a sessão está ativa, informações sobre a rede que compõe a sessão como endereços IP e portas utilizadas, informações sobre a mídia sendo transferida como tipo, protocolo de transporte, formato, entre outras informações relevantes (BEGEN *et al.*, 2021).

O protocolo utiliza uma formação de <tipo> = <valor>, com o intuito de descrever sessões completamente distintas e especificas para os participantes da rede, o <tipo> define um parâmetro de sessão e o <valor> fornece um valor específico para esse parâmetro (TUTORIALSPOINT, 2022).

Os parâmetros de um protocolo SDP e suas definições, a sua totalidade, são demonstrados no Quadro 2.

Quadro 2 - Parâmetros do protocolo SDP

Parâmetros	Definições
v	Versão do protocolo
o	Criador/Proprietário da sessão
s	Nome da sessão
i	Informações da sessão
u	URI da descrição
e	Endereço de e-mail
p	Número de celular
c	Informações da conexão
b	Informações da rede
z	Informações sobre tempo
k	Chave de criptografia
a	Zero ou mais atributos da sessão

Fonte: Adaptado de Tutorialspoint (2022)

Um projeto bastante popular que utiliza o protocolo SDP para transmissão de mídia é o WebRTC, tendo seu foco em aplicações via navegadores utilizando comunicação em tempo real, aplicações de vídeo conferências e sessões de mídia (MARAšEVIć; GAVROVSKA, 2020).

3.3 WebRTC

A tecnologia WebRTC é um projeto gratuito e de código aberto que oferece uma conexão ponto a ponto entre navegadores e aplicativos móveis. Diferentemente dos protocolos apresentados anteriormente, o WebRTC é disponibilizado em formato de API, permitindo a ligação direta entre navegadores ou aplicações móveis para troca de informações, arquivos ou qualquer tipo de dados (MARAšEVIć; GAVROVSKA, 2020).

Sendo muito utilizado para aplicações de vídeo conferência, bate papo por voz e famosa pela sua simplicidade de implementação, a tecnologia é composta pela junção de outras três APIs, onde cada uma conta com suas funcionalidades específicas e ao utilizá-las em conjunto é possível estabelecer uma comunicação em tempo real (LI *et al.*, 2019), sendo elas:

- GetUserMedia: Concede ao usuário o acesso a câmera e/ou microfone do dispositivo (ÖZTÜRK *et al.*, 2021).

- RTCPeerConnection: Utilizado para criar uma conexão ponto a ponto eficiente. É importante ressaltar que, esta conexão só é estabelecida e a troca de dados iniciada após as informações dos pontos serem muito bem definidas com SDP (LI *et al.*, 2019).
- RTCDataChannel: Seu papel na arquitetura é transferir os dados de um ponto ao outro, estabelecendo um canal de alta taxa de transferência e baixa latência para a transmissão dos dados (LI *et al.*, 2019).

Uma comunicação pela tecnologia WebRTC é estabelecida via protocolo SDP, e após entender os conceitos da API, é preciso entender o fluxo para estabelecer uma conexão utilizando-a.

3.3.1 Reconhecimento de endereços

Um dos problemas encontrados na formalização não só de um SDP, mas na comunicação ponto a ponto em geral, é que nos dias de hoje a maioria dos dispositivos ligados a rede estão atrás de um *firewall* ou roteadores domésticos (MARAšEVIć; GAVROVSKA, 2020). Este tipo de segurança procura mudar constantemente seus endereços IP graças ao NAT, não sendo uma tradução padronizada, e, este processo gera um problema bastante comum no mundo das comunicações pela rede, sendo necessário encontrar uma maneira de passar por esta segurança para que seja possível que dois pontos se comuniquem (LI *et al.*, 2019).

Como medida tratativa para que cada cliente crie um SDP corretamente antes de enviá-lo ao servidor de sinal, o WebRTC utiliza a tecnologia ICE, que ajuda os clientes a coordenar a descoberta de seus endereços IP públicos, onde ambos os pontos que buscam iniciar uma comunicação gerarão uma lista de candidatos a "*ice candidates*" que contêm um endereço IP que um cliente "A" pode usar para se conectar com o cliente "B" (TEMASYS, 2022).

Em segundo plano, o WebRTC fará uma série de requisições para um servidor STUN de maneira que cada cliente salve seus candidatos ICE (que podem ser lidos pelo outro cliente), contudo, o algoritmo determina qual o melhor candidato e em qual ponto a mídia em tempo real pode começar a ser transmitida entre os dois pontos (TEMASYS, 2022). Servidores STUN são disponibilizados gratuitamente na Internet por fontes confiáveis como *Google*.

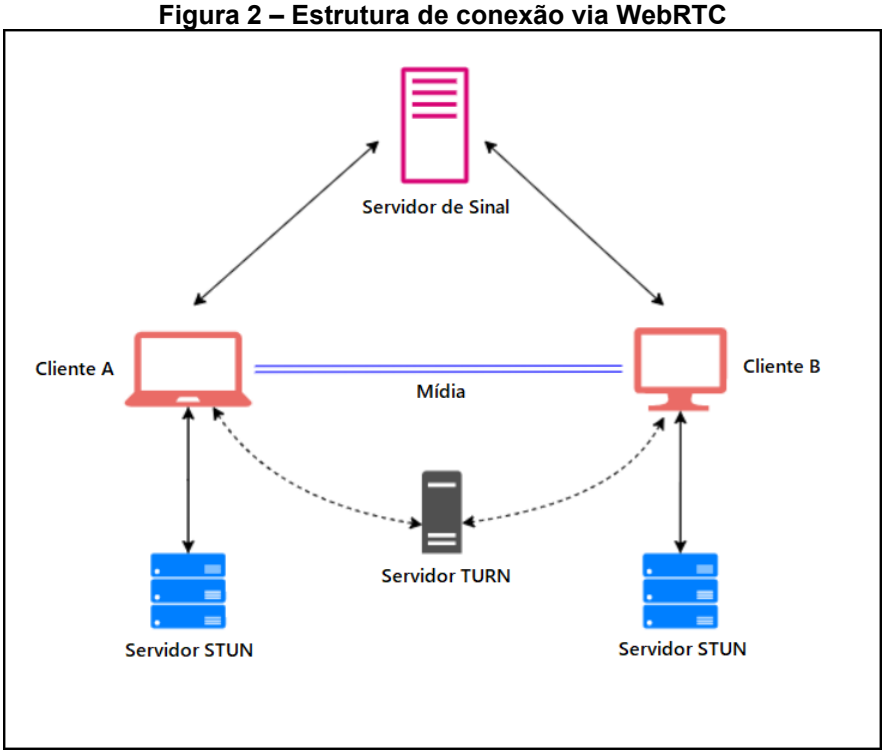
Em algumas vezes, um servidor STUN não é capaz de identificar o IP público de ambos os clientes e estabelecer uma conexão entre eles, então entra o servidor TURN. Semelhante ao STUN, o TURN busca encontrar um endereço público dos pontos, porém, um servidor TURN é capaz de estabelecer uma conexão entre dois pontos, onde ambos conseguem realizar uma troca de dados de mídia (LI *et al.*, 2019). Mesmo não sendo uma conexão direta, ao utilizar a porta de retransmissão do servidor, o TURN garante que os dados não passarão pelo firewall ou roteador, assumindo um papel intermediário entre os pontos para que a conexão aconteça (LI *et al.*, 2019).

3.3.2 *Signaling Server* (Servidor de Sinal)

Um dos processos mais importantes dessa tecnologia é de responsabilidade do servidor de sinal, que é indispensável para que isso aconteça. Para que se estabeleça uma conexão entre um cliente "A" e um cliente "B", é necessário que o cliente "A" envie uma oferta SDP para o servidor de sinal solicitando que o cliente "B" se conecte a ele (TRUECONF, 2022). Após este processo, o servidor então sinaliza para o destinatário (cliente "B") dizendo que o cliente "A" deseja estabelecer uma conexão (TRUECONF, 2022). Caso o cliente "B" aceite esta conexão, ele devolverá para o servidor uma resposta SDP para que por fim, o servidor entregue-a ao cliente "A" e os dois pontos estabeleçam uma conexão direta (TRUECONF, 2022). O protocolo de comunicação utilizado pelo Servidor de Sinal não é específico, deixando-o bastante versátil para diferentes implementações (LI *et al.*, 2019).

A figura 2 ilustra uma estrutura de conexão via WebRTC.

Figura 2 – Estrutura de conexão via WebRTC



Fonte: Adaptado de Kumar et al. (2021)

3.4 Câmeras virtuais

Uma transmissão entre dispositivos não é suficiente para que uma mídia seja disponibilizada via *webcam*, portanto, apresenta-se o conceito de câmeras virtuais.

Como o nome sugere, câmera virtual é uma ferramenta capaz de emular uma *webcam* tradicional e pode ser utilizada por grande parte das aplicações que tem por finalidade o uso de vídeo conferências (E2ESOFT, 2021). A grosso modo, uma câmera virtual passa a ser reconhecida pelo sistema operacional mesmo que não exista fisicamente, e, em sua grande maioria, é utilizada para demonstrar animações, filtros, filmes, imagens ou até mesmo manipular a imagem de uma webcam real (E2ESOFT, 2021).

Em 29 de julho de 2017 o software Open Broadcaster Software, bastante estabelecido no ramo *de streaming,* passou a fornecer gratuitamente um *plugin* para manipulação de vídeo via câmera virtual elevando ainda mais a utilização do conceito (OBS, 2021).

O plugin OBS-VirtualCam utiliza a tecnologia DirectShow[9] que por sua vez é uma tecnologia desenvolvida para resolver desafios quando trata-se de dados multimídia, tem como foco simplificar a tarefa de criar aplicativos de mídia para *desktop* isolando a complexidade de transporte de dados (WHITE *et al.*, 2022).

3.5 Benchmarks

Em cenários empresariais e aplicações do cotidiano, desenvolver um produto de sucesso e mantê-lo de forma sustentável, vai além de uma busca incessante por excelência nos processos, de maneira que, são necessários parâmetros e referências do que se busca alcançar (VOLPATO, 2020).

Em diversas situações é comum encontrar dificuldades na validação de produtos com base na concorrência, e, uma forma simples de resolver esse problema é através de uma prática chamada de *benchmark* (VOLPATO, 2020).

O *benchmark* é uma técnica desenvolvida originalmente para o mundo corporativo que busca melhorar a eficiência de processos comparando o desempenho de resultados, utilizando métricas como pesquisa de campo, analise de produtos, práticas de outras empresas de sucesso e até mesmo engenharia reversa em seus produtos (VOLPATO, 2020).

Aos poucos a técnica foi migrando para o meio da informática, assumindo os mesmos padrões de comparar dois ou mais elementos com padrões e testes similares (MARTINEZ, 2022).

Os testes de *benchmark* no meio da informática têm como objetivo analisar o desempenho de aplicações com base em *hardware*, levando em conta o processador, memória RAM e placa de vídeo, onde os sistemas capazes de realizar esse tipo de teste geram diversas informações relevantes em questão de desempenho, demonstrando um índice correspondente à performance geral ou específica da aplicação analisada (MARTINEZ, 2022).

Uma das grandes vantagens de usar este tipo de teste é que eles são excelentes para usuários comuns demonstrando os pontos fortes de fracos de uma aplicação ou *hardware* com muita eficiência, medindo situações reais e com aplicações reais (MARTINEZ, 2022).

[9] https://learn.microsoft.com/pt-br/windows/win32/directshow/directshow

3.6 Considerações

Com base nos estudos supracitados, aliado a alta considerável na demanda por videoconferências e o baixo rendimento de aplicativos gratuitos para uso de *smartphones* como *webcam*, um dos maiores benefícios da tecnologia WebRTC é que os clientes podem se comunicar entre si sem necessidade de instalação de *plug-ins* ou softwares terceiros, ignorando o servidor e diminuindo consideravelmente a latência na troca de dados, ligeiramente diferente de outras tecnologias nas quais os clientes precisam esperar o servidor processar os dados pra depois recebê-los. Outro ponto importante é a facilidade disponibilizada pela API, pela qual tarefas difíceis como decodificação de vídeo e estruturação de conexão são realizadas por funções formalizadas e distribuídas pela biblioteca. Portanto, o WebRTC em conjunto com a manipulação de uma câmera virtual, viabiliza o desenvolvimento de um aplicativo multiplataforma capaz de estabelecer uma conexão em tempo real entre cliente(*smartphone*) e servidor(*desktop*) para utilização da câmera do smartphone como *webcam*, buscando baixa latência e qualidade estável de imagem.

4 METODOLOGIA

A metodologia aplicada no trabalho segue em torno do estabelecimento de uma conexão entre dispositivos até a demonstração da imagem da transmissão via *webcam*.

A arquitetura da aplicação sugeriu uma aplicação *desktop*(servidor) que buscou construir um cenário onde uma aplicação *mobile*(cliente) consiga fornecedor uma transmissão de vídeo sendo capturada pela câmera do *smartphone*, e posteriormente, disponibilizar essa transmissão via *webcam*.

A primeira etapa foi realizar estudos sobre protocolos de transmissões em tempo real e suas aplicações. Consequentemente, a conexão entre aplicações no projeto é realizada com base no funcionamento do WebRTC, assumindo uma abordagem multiplataforma com o Flutter[10].

Com a proposta de uma aplicação de código aberto, o projeto seguiu com uma busca por ferramentas *open-source* capazes de manipular as funcionalidades da API (WebRTC). O estabelecimento de uma conexão entre duas aplicações rodando em sistemas distintos foi possível ao manipular os códigos base disponibilizados pelo projeto flutter-webrtc[11], realizando etapas como, validação de informações, troca de requisições entre as aplicações e os servidores, além da transmissão de vídeo.

Sendo assim, com a transmissão de vídeo entre dispositivos sendo realizada pelas aplicações, o conceito de câmeras virtuais foi escolhido para manipular a mídia, uma vez que utiliza-se um *plugin* externo para inicializar uma câmera virtual e posteriormente manipula-la, de maneira que seja possível manipular os dados provindos da aplicação mobile e disponibiliza-los como uma webcam.

Por fim, buscou-se validar a solução proposta, portanto, foram realizados testes como, *benchmark*, validações por parte dos usuários e testes de qualidade de imagem. As ferramentas utilizadas no desenvolvimento bem como a metodologia proposta nos testes estão descritas nas sessões 4.1 e 4.2.

4.1 Ferramentas

Assim como um dos pilares do projeto, as ferramentas e dependências utilizadas nas aplicações são *open-source*, não exigindo nenhuma licença ou

[10] https://flutter.dev/
[11] https://github.com/flutter-webrtc

pagamento para utilização/implementação. As 3 linguagens utilizadas na solução foram:

- Flutter: A tecnologia escolhida para o desenvolvimento capaz de suprir as necessidades apontadas foi o Flutter. Disponibilizado pela *Google,* é um *framework* de código aberto bastante procurado quando se diz respeito a aplicações multi-plataforma. Utilizada em ambas as aplicações cliente e servidor, é responsável pela transmissão de vídeo em tempo real entre aparelhos via API, além de toda a interface das aplicações.
- Go: A linguagem GoLang[12] é utilizada no desenvolvimento dos servidores TURN e Sinal utilizados para comunicação entre aplicações.
- Python: Por sua vez, a linguagem Python[13] é utilizada no desenvolvimento de um *script* para manipulação da mídia, disponibilizando-a como uma webcam.

As dependências utilizadas no projeto bem como bibliotecas e *plugins* externos foram:

- Flutter-webrtc: Outro ponto importante na escolha do Flutter é que sendo produzido pela mesma empresa, o *framework* contém a biblioteca Flutter-webrtc, disponibilizando todo o sistema de conexão e troca de informações necessárias entre o projeto e a API WebRTC, por sua vez, sendo responsável pela transmissão de vídeo entre aplicações.
- Flutter-webrtc-server: Para o reconhecimento de endereços e sinalização entre clientes em uma rede local, sendo composto pela implementação de um servidor TURN e um servidor de sinal, o projeto flutter-webrtc-server[14] foi desenvolvido utilizando a linguagem GoLang e *websockets* para a troca de mensagens e estabelecimento de uma conexão entre as aplicações.
- OBS-VirtualCam: O OBS-VirtualCam[15] é um *plugin* utilizado para instalar uma câmera virtual, disponibilizado pelo OBS[16]. Seu papel no projeto foi emular uma *webcam* e disponibilizar a transmissão de vídeo para sistemas de videoconferência.

[12] https://go.dev/
[13] https://www.python.org/
[14] https://github.com/flutter-webrtc/flutter-webrtc-server
[15] https://obsproject.com/forum/resources/obs-virtualcam.539/
[16] https://obsproject.com

- PyWin32: Este módulo disponibilizado para linguagem Python, utiliza os recursos nativos do sistema operacional *Windows*, no projeto, o PyWin32[17] foi utilizado para captura de imagem da transmissão.
- PyVirtualCam: Em conjunto com o módulo anterior, o módulo PyVirtualCam[18] é capaz de iniciar a câmera virtual e manipula-la, de modo a disponibilizar a imagem da transmissão como uma webcam.

4.2 Experimentos

Para avaliação da solução apresentada foram usadas três diferentes metodologias.

O primeiro experimento foi realizado em torno de um cenário de *Benchmark* utilizando o software MSI-Afterburner[19], onde analisou-se o desempenho da solução proposta em termos técnicos, buscando avaliar propriedades como desempenho de CPU, memória RAM, GPU e FPS. Deste modo, a solução foi exposta a uma comparação direta com o aplicativo Camo. Não foram feitos testes de *Benchmark* envolvendo outras aplicações por problemas na captura das informações.

A segunda metodologia de validação foi um teste de avaliação dos usuários, onde, foi realizada uma videoconferência de aproximadamente 10 minutos buscando validar a solução em termos de qualidade de transmissão em comparação a outros cenários de transmissão. Na videoconferência em questão, foram selecionadas 16 pessoas convidadas por meio de um canal da rede social Discord voltado para ciência da computação, onde o foco da transmissão foi ensinar aos participantes a montar um cubo mágico. Essa transmissão teve 3 etapas, onde em cada uma delas, foram utilizados métodos de transmissão de vídeo diferentes, sendo uma delas a solução proposta. Em sequência, foi disponibilizado aos voluntários um questionário buscando validar a solução.

Por fim, na última metodologia de teste foi proposto um teste direto de comparação de qualidade de vídeo. Baseando-se em um “teste as cegas”, a proposta é validar a qualidade de imagem e preferência de transmissão dos voluntários sem nenhuma informação sobre qual dos cenários demonstrados é a solução proposta. Portanto, foi realizada uma gravação de uma videoconferência onde foi montado um

[17] https://pypi.org/project/pywin32/
[18] https://pypi.org/project/pyvirtualcam/
[19] https://www.msi.com/Landing/afterburner/graphics-cards

cenário com 4 diferentes tipos de transmissão, capturando o mesmo vídeo simultaneamente. Posteriormente, este vídeo foi disponibilizado em uma comunidade de desenvolvedores e solicitado para que os voluntários respondessem um questionário com questões referentes a qualidade de cada transmissão individualmente, e, por qual cenário eles tem mais interesse.

Todas as metodologias de testes foram realizadas utilizando as versões gratuitas das aplicações, já que, um dos objetivos deste trabalho é desenvolver uma aplicação de código aberto.

5 DESENVOLVIMENTO

O projeto em questão se divide em uma aplicação cliente-servidor, na qual o lado cliente é inteiramente utilizado pelo *smartphone*, e, a aplicação do lado servidor é utilizada pelo computador a qual é responsável tanto pelo servidor quanto pela disponibilidade da mídia para utilização como *webcam*. À sua totalidade, o conjunto das aplicações foi nomeado *TCCam*. Ambas as aplicações cliente e servidor encontra-se disponíveis no repositório TCCam[20].

5.1 Requisitos

Com base nos estudos da tecnologia WebRTC presentes neste documento, uma vez que a tecnologia foi estruturada e desenvolvida para prestar serviço entre navegadores de redes distintas, o projeto apresenta uma estrutura focada em transmissão de mídia entre dispositivos em uma mesma rede local.

Aliado a estrutura da tecnologia junto ao objetivo do projeto, para que seja estabelecida uma conexão entre dois pares e uma possível transmissão de mídia entre ambos, capaz de ser disponibilizada pelo computador como uma *webcam* tem-se os seguintes requisitos:

- Ponto 1 - Servidor (computador)
- Ponto 2 - Cliente (*smartphone*).
- Servidor para reconhecimento dos endereços.
- Servidor de sinal.
- Transmissão via API.
- Manipulação da mídia para uma saída webcam.

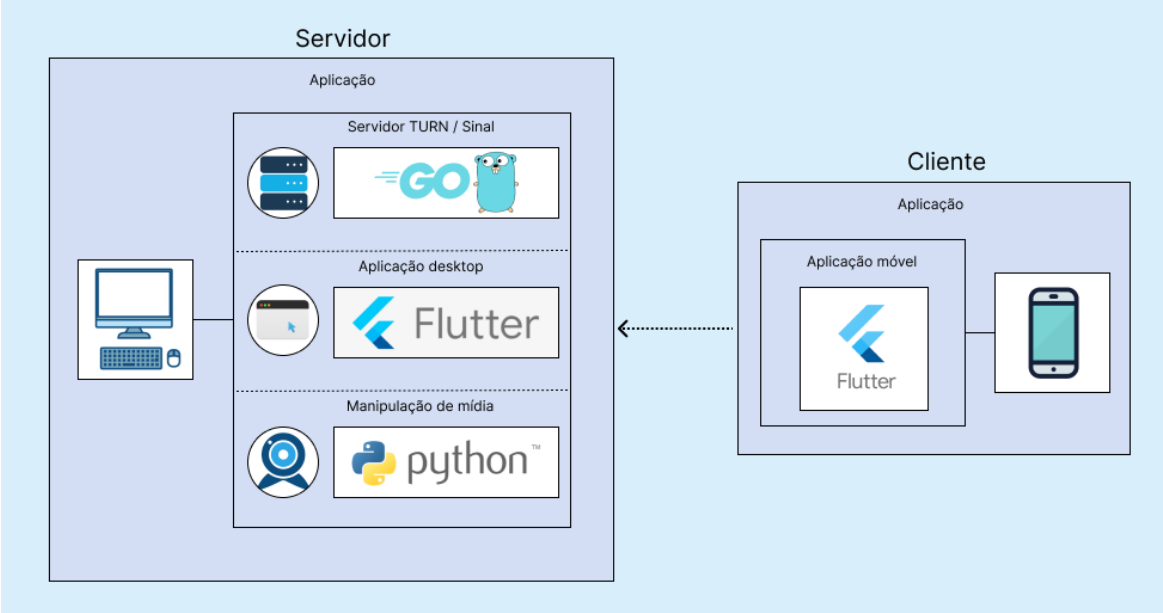
Tendo conhecimento dos requisitos supracitados, a arquitetura da aplicação junto as tecnologias utilizadas são discutidas na seguinte seção.

5.2 Arquitetura da aplicação

Para o desenvolvimento da solução, foi proposta a arquitetura ilustrada na Figura 3, demonstrando a arquitetura da aplicação em conjunto com as tecnologias utilizadas para cada processo.

[20] Código disponível em https://github.com/dpavlak/TCCam

Figura 3 - Arquitetura geral da aplicação



Fonte: Autoria própria (2022)

O projeto foca em uma transmissão de vídeo entre dois dispositivos conectados em uma mesma rede utilizando WebRTC. Nota-se que ao utilizar uma troca de informações local reduz ainda mais a latência da transmissão.

Tendo como base outros aplicativos que atuam no mercado e ressaltando a ideia de uma aplicação gratuita, o projeto em questão não apresenta diferentes tipos de usuários ou conexões com diferentes opções de utilização, descartando a necessidade de um banco de dados para controle de permissões.

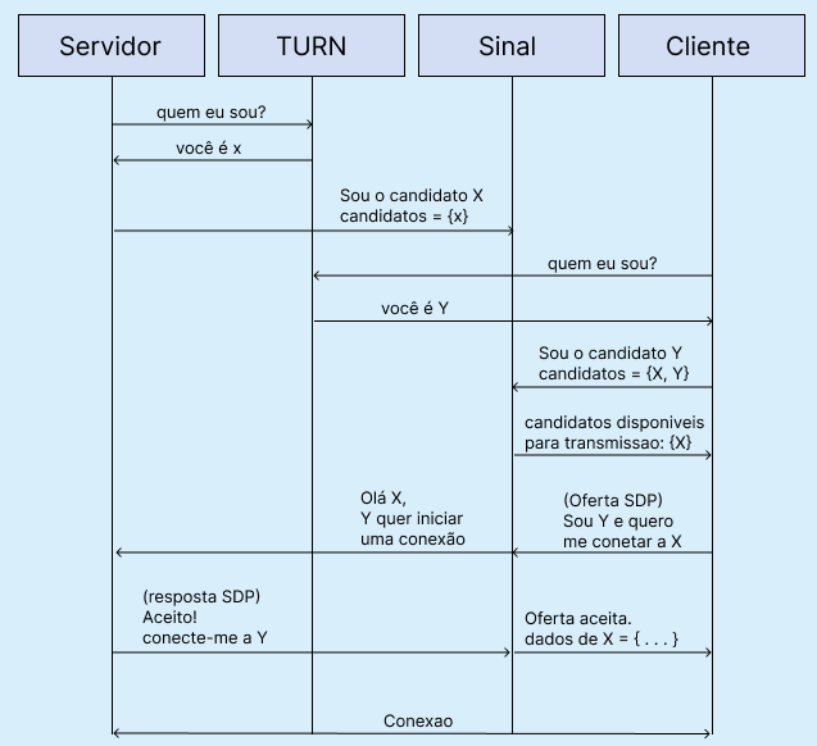
Para uma utilização simples e rápida, ao iniciar a aplicação servidor no computador é instalada a câmera virtual do aplicativo OBS-Studio para que, posteriormente, seja possível disponibilizar a imagem do aplicativo em formato de *webcam*. Juntamente a este processo, os servidores TURN e Sinal são iniciados automaticamente, e, uma mensagem de reconhecimento de endereço pelo aplicativo é enviada instantaneamente para o servidor TURN. Tendo esse processo realizado, a aplicação *desktop* já está apta a receber uma oferta SDP para começar uma transmissão.

Já pelo lado do cliente, ao iniciar a aplicação no dispositivo móvel o usuário é apresentado a uma tela inicial contendo um botão para iniciar uma conexão. Ao clicar neste botão é preciso informar ao sistema o endereço do servidor TURN, previamente disponibilizado pelo computador, para que o mesmo faça o reconhecimento do seu respectivo endereço na rede.

Após ambas as aplicações terem conhecimento do seu respectivo endereço, a aplicação móvel é redirecionada para uma tela onde pode enviar uma oferta SDP para o servidor de Sinal, que faz a comunicação necessária entre ambos os clientes (*desktop* e dispositivo móvel), dando início a conexão ponto a ponto e transmissão da mídia.

A Figura 4 ilustra o fluxo de requisições utilizadas entre cliente e servidor para iniciar uma transmissão de mídia, de modo que a aplicação desktop é representada pela coluna “Servidor” e a aplicação móvel pela coluna “Cliente”.

Figura 4 – Fluxo de conexão entre os dispositivos



Fonte: Autoria própria (2022)

Ao executar este processo e ter a transmissão de vídeo acontecendo entre as aplicações, o computador dá início a manipulação da mídia para que a mesma seja disponibilizada pela câmera virtual OBS e utilizada como webcam pelo usuário.

Com base nas informações descritas, as seções 3.3 e 3.4 são destinadas para entender de maneira mais detalhada cada componente da aplicação e sua respectiva funcionalidade no projeto.

5.3 Cliente

O cliente é responsável por servir dados de vídeo para o servidor. Representado por um aplicativo móvel no projeto, conta com suporte para sistemas operacionais IOS e Android, de maneira que respeitem as seguintes exigências:

Quadro 3 - Versões necessárias de sistemas operacionais mobile

Sistema Operacional	Versão mínima	Versão recomendada
iOS	8	10.1
Android	5.0	9

Fonte: Autoria própria (2022)

Para fins de explicação e organização, as telas do aplicativo móvel serão tratadas neste tópico como (1, 2, 3, 4 e 5), sendo demonstradas em conjunto na Figura 5.

Figura 5 - Fluxo de tela do aplicativo móvel

Fonte: Autoria própria (2022)

Como citado anteriormente, assumindo o fato de não precisar de um controle de diferentes tipos de usuários, o sistema não dispõe de uma tela para cadastro de usuários. Ao iniciar o aplicativo móvel o usuário é apresentado a tela inicial (Figura 5.1) contendo apenas um botão para iniciar uma transmissão.

Ao clicar sobre o botão é demonstrado um diálogo (Figura 5.2) onde é requisitado ao usuário o endereço do servidor, informação essa, demonstrada na tela inicial da aplicação *desktop*, demonstrada na seção 5.4.1 deste trabalho.

Após inserir o IP do servidor no seu respectivo campo, ao clicar sobre o botão "conectar" o usuário inicia o fluxo de conexão WebRTC ilustrado na Figura 4. A função *getTurnCredential* envia uma mensagem para o servidor TURN requisitando seu endereço na rede, tipo de conexão disponível e demais informações necessárias para posteriormente gerar um SDP. Esta troca de mensagens entre cliente e servidor é demonstrada no Quadro 4.

Quadro 4 - Requisição e resposta sobre reconhecimento de informações

```
getTurnCredential send: Allow self-signed certificate => 172.22.192.1:8086
getTurnCredential received data: response => {
    "username": "1662929619:flutter-webrtc",
    "password": "IzkLHP0ePL/OGfDXizCIvt0Fkhw",
    "ttl": 86400,
    "uris": ["turn:127.0.0.1:19302?transport=udp"]
}
```

Fonte: Autoria própria (2022)

Para dar continuidade no processo de estabelecimento de conexão, o cliente automaticamente enviará uma mensagem para o servidor de sinal demonstrando sua disponibilidade para iniciar uma conexão de transmissão de mídia, consequentemente, o mesmo responderá com o candidato disponível, neste caso, o aplicativo *desktop*. O Quadro 5 demonstra a troca de mensagens em questão. Nota-se que a resposta entregue pelo servidor(*received data*) já demonstra ambos os dispositivos disponíveis para conexão (dispositivo *mobile* e *desktop*). Posteriormente o cliente é redirecionado para uma tela onde é demonstrado o nome e id do candidato disponível junto a um botão para iniciar a transmissão (Figura 5.3).

Quadro 5 - Requisição e resposta sobre disponibilidade de candidatos

```
send: {
    "type": "new",
    "data": {
        "name": "Fluttrer android(localhost)",
        "id": "545084",
        "user_agent": "flutter-webrtc/android-plugin 0.0.1"
    }
}

received data: {
    "type": "peers",
    "data": [
        {
            "id": "585450",
            "name": "Flutter windows(DESKTOP-BSMHG2H)”,
            "user_agent”: "flutter-webrtc/windows-plugin 0.0.1"
        },
        {
            "id": "545084",
            "name": "Flutter android(localhost)”,
            "user_agent”: "flutter-webrtc/android-plugin 0.0.1"
        }
    ]
}
```

Fonte: Autoria própria (2022)

Neste momento, possuindo as informações necessárias de ambos os pontos sobre os quais deseja-se iniciar a conexão, o próximo passo é enviar uma oferta SDP do cliente para o servidor de sinal para que o mesmo envie a oferta para a aplicação *desktop*. Ao clicar sobre o ícone de câmera, uma sequência de passos é utilizada para criar a oferta SDP e convidar o aplicativo *desktop* para uma transmissão de maneira que, primeiramente cria-se a sessão contendo dados como identificação do dispositivo e tipo de mídia transmitida. Posteriormente utiliza-se as funções getUserMedia[21] em conjunto com createPeerConnection[22], ambas disponibilizadas pela biblioteca para modelar a oferta SDP e enviá-la para o servidor. Um exemplo de SDP criado após este processo é demonstrado na estrutura JSON do Quadro 6.

[21] https://api.flutter.dev/flutter/dart-html/Navigator/getUserMedia.html
[22] https://pub.dev/documentation/flutter_webrtc/latest/flutter_webrtc/createPeerConnection.html

Quadro 6 - Exemplo de oferta SDP

```
data: {
    "type": "offer",
    "data": {
        "to": "082764",
        "from": "202625",
        "description": {
            "sdp": "v=0\r\no=- 6171545722842729842 2 IN IP4 127.0.0.1\r\ns=-\r\nt=0
0\r\na=group: BUNDLE 0\r\na=extmap-allow-mixed\r\ na=msid-semantic: WMS 0c7c0893-4749-
443e-86e0-4a5509bbb510\r\nm=video 9 UDP/TLS/RTP/SAVPF 96 97 98 99 35 36 125 124
127\r\nC=IN IP4 0.0.0.0\r\na=rtcp:9 IN IP4 0.0.0.0\r\na=ice-ufrag:X995\r\na=ice-
pwd:moacZkAD2HFeE90ZhDS5W9GC\r\na=ice-options: trickle renomination\r\na=fingerprint:sha-
256
CD:12:12:42:40:47:50:E0:2A:61:8F:6B:FF:BE:C6:7B:DD:43:D0:EF:AB:8D:75:89:29:FD:FC:96:F2:0F:
3D:30\r\na=setup:actpass\r\na=mid:0\r\na-extmap:1 urn:ietf:params:rtp-hdrext:toffset\r\na-
extmap:2 http://www.webrtc.org/experiments/rtp-hdrext/abs-send-time\r\na=extmap:3
urn:3gpp:video-orientation\r\na-extmap:4 http://www.ietf.org/id/draft-holmer-rmcat-
transport-wide-cc-extensions-01\r\na=extmap:5 http://www.webrtc.org/experiments/rtp-
hdrext/playout-delay\r\na-extmap:6 http://www.webrtc.org/experiments/rtp-hdrext/video-
content-type\r\na-extmap:7 http://www.webrtc.org/experiments/rtp-hdrext/Video-timing\r\na-
extmap:8 http://www.webrtc.org/experiments/rtp-hdrext/color-space\r\na=extmap:9
urn:ietf:params:rtp-hdrext:sdes:mid\r\na-extmap:10 urn:ietf:params:rtp-hdrext:sdes:rtp-
stream-id\r\na=extmap:11 urn:ietf:params:rtp-hdrext:sdes:repaired-rtp-stream-
id\r\na=sendrecv\r\na=msid:0c7c0893-4749-443e-86e0-4a5509bbb510 a4406cb6-01bc-4cab-ab81-
b9ee41e9264c\r\na=rtcp-mux\r\na=rtcp-rsize\r\na=rtpmap:96 VP8/90000\r\na=rtcp-fb:96 goog-
remb \r\na=rtcp-fb:96 transport-cc\r\na=rtcp-fb:96 ccm fir\r\na=rtcp-fb:96 nack\r\na=rtcp-
fb:96 nack pli\r\na=rtpmap:97 rtx/90000\r\na=fmtp:97 apt=96\r\na=rtpmap:98
VP9/90000\r\nartcp-fb:98 goog-remb\r\na=rtcp-fb:98 transport-cc\r\na=rtcp-fb:98 ccm
fir\r\na=rtcp-fb:98 nack\r\na=rtcp-fb:98 nack pli\r\na=rtpmap:99 rtx/90000\r\na=fmtp:99
apt=98\r\na=rtpmap:35 AV1/90000\r\na=rtcp-fb:35 goog-remb\r\na=rtcp-fb:35 transport-
cc\r\na=rtcp-fb:35 ccm fir\r\na=rtcp-fb:35 nack\r\na=rtcp-fb:35 nack pli\r\na=rtpmap:36
rtx/90000\r\na=fmtp:36 apt=35\r\na=rtpm ap:125 red/90000\r\na=rtpmap: 124
rtx/90000\r\na=fmtp:124 apt=125\r\na=rtpmap:127 ulpfec/90000\r\na-ssrc-group:FID 389649963
1 2240307127\r\na=ssrc:3896499631 cname: LlFewMGGPgppDCig\r\na=ssrc:3896499631
msid:0c7c0893-4749-443e-86e0-4a5509bbb510 a44 06cb6-01bc-4cab-ab81-
b9ee41e9264c\r\na=ssrc:2240307127 cname:llFeWMGGPgppDCig\r\na=ssrc:2240307127
msid:0c7c0893-4749-443e-86e0-4a5509bbb510 a44e6cb6-01bc-4cab-ab81-b9ee41e9264c\r\n",
            "type":"offer"
        },
        "session_id": "202625-682764",
        "media": "video"
    }
}
```

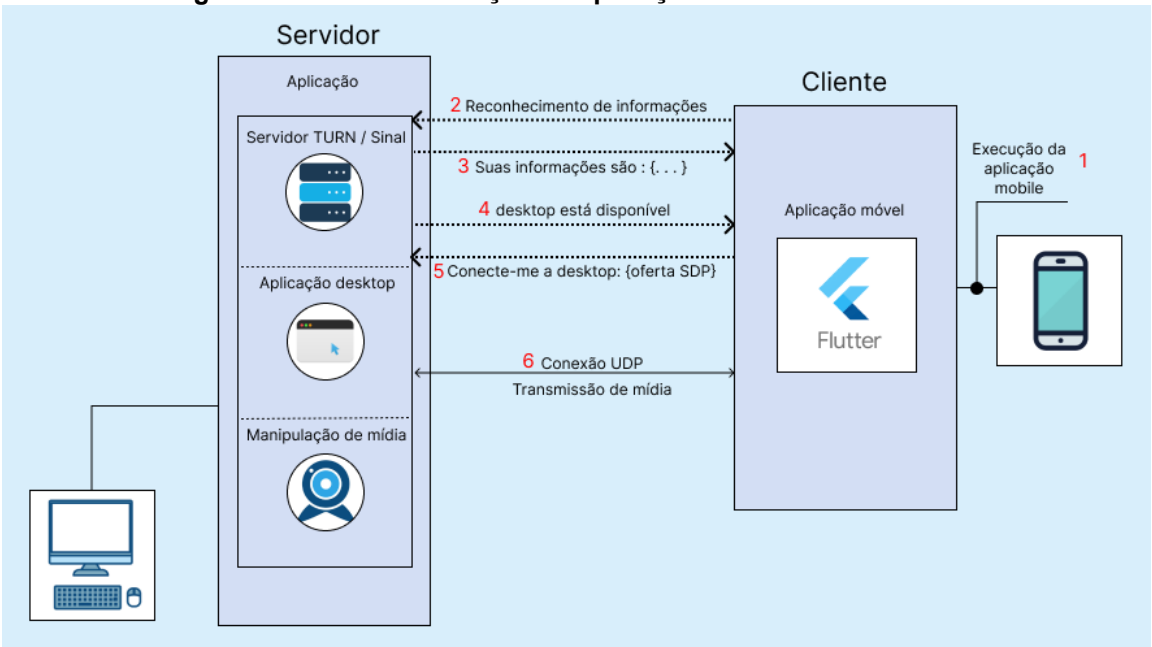
Fonte: Autoria própria (2022)

Após a oferta ser enviada, o aplicativo apresenta um diálogo informando ao usuário que está aguardando a confirmação (Figura 5.4). Quando o convite é aceito pelo aplicativo *desktop*, a conexão por fim é iniciada juntamente com a transmissão do vídeo de modo que o usuário dispõe de dois botões, um destinado a encerrar conexão e consequentemente a transmissão, e, outro com a finalidade de transicionar entre as câmeras frontal e traseira do dispositivo móvel (Figura 5.5).

Para fins de compreensão geral do projeto, a Figura 6 ilustra a maneira que o aplicativo móvel se conecta com a aplicação *desktop*, enumerando o fluxo de

interação entre a aplicação e o servidor de 1 a 6, respeitando-a de maneira ascendente e assumindo uma conexão que foi aceita pela aplicação *desktop*.

Figura 6 - Fluxo de interação da aplicação móvel com o servidor



Fonte: Autoria propria (2022)

5.4 Servidor

O lado servidor da aplicação dispõe de uma manipulação maior de tecnologias de modo que realize as seguintes operações:

1) Inicialização dos servidores TURN e Sinal.
2) Estabelecer uma conexão com o cliente.
3) Manipular a mídia recebida de modo a ser exibida como *webcam*.

Esta seção conta com outras três subseções para que as operações sejam explicadas separadamente, devido a quantidade de operações e suas respectivas particularidades.

5.4.1 Servidores TURN e Sinal

Uma vez que a tecnologia já está consolidada em seu respectivo ramo de atuação, para projetos que utilizam conexão local é disponibilizado pela empresa

proprietária o flutter-webrtc-server, uma aplicação com finalidade de inicializar e gerenciar um servidor local capaz de estabelecer uma conexão entre dois clientes.

Um cenário onde um cliente “X” e um cliente “Y” querem estabelecer uma conexão, compreende-se que eles precisam transitar dados entre si e para eles se comunicarem, em primeiro lugar, é necessário verificar se ambos os clientes conseguem se comunicar e de que maneira isso deve ser feito. Em segundo lugar, é preciso conseguir autenticar essa conexão, ou seja, deixar essa conexão segura. E por fim, estabelecer uma conexão de maneira que consiga efetuar esse trafego de dados.

A aplicação em questão dispõe de um servidor TCP/IP, onde o servidor TURN é responsável pela validação dos clientes dando ao cliente sua respectiva autenticação na rede. Já o servidor de Sinal, é a interface de comunicação entre cliente e servidor, servindo como um “tradutor” de mensagens HTTP utilizando as credenciais dos clientes para conseguir autenticar uma conexão entre as duas pontas. Depois que a autenticação é feita, o serviço estabelece esse meio de comunicação via *websocket* utilizando protocolo UDP.

Acresce que o *websocket* não é um servidor, sendo apenas um meio de comunicação entre os clientes. O único servidor de fato existente nesta aplicação é o TURN, responsável receber as mensagens sendo elas uma requisição de validação direta vinda do cliente, ou, mensagens do cliente traduzidas pela interface de comunicação (servidor de Sinal) como por exemplo uma oferta/resposta SDP.

O Quadro 7 mostra a configuração do servidor e suas respectivas rotas.

Quadro 7 - Configuração do servidor

	Rotas
Host	“0.0.0.0” (*localhost*)
Porta	8086
WebSocket	“/ws”
Servidor Turn	“/api/turn”

Fonte: Flutter-webrtc-server

Além das rotas, para que posteriormente exista a troca de mensagens de maneira correta, na interface de comunicação (servidor de sinal) foram implementados

métodos codificados e decodificados com o formato de comunicação JSON[23]. A estrutura dos métodos é apresentada no Quadro 8.

Quadro 8 - Definição dos métodos para troca de mensagens

Nome	Estrutura
New	‘id’: _selfId, (String) ‘name’: DeviceInfo.label, (String) ‘user_agent’: DeviceInfo.userAgent (String)
Bye	‘session_id’: sessionId, (String) ‘from’: _selfId (String)
Offer	‘to’: session.pid, (String) ‘from’: _selfId, (String) Description: { ‘sdp’: RTCSessionDescription,sdp, (String) ‘type’: RTCSessionDescription.type, (String) }, ‘session_id’: session.sid, (String) ‘media’: media, (String)
Answer	‘to’: session.pid, (String) ‘from’: _selfId, (String) Description: { ‘sdp’: RTCSessionDescription,sdp, (String) ‘type’: RTCSessionDescription.type, (String) }, ‘session_id’: session.sid (String)
Candidate	'to': peerId, (String) 'from': _selfId, (String) 'candidate': { 'sdpMLineIndex': candidate.sdpMLineIndex (Int), 'sdpMid': candidate.sdpMid (String), 'candidate': candidate.candidate (String) }, ‘sessionId’: ‘sessionId’ (String)
Leave	‘peerId’: ‘peerId’ (String)

Fonte: Flutter-webrtc-server

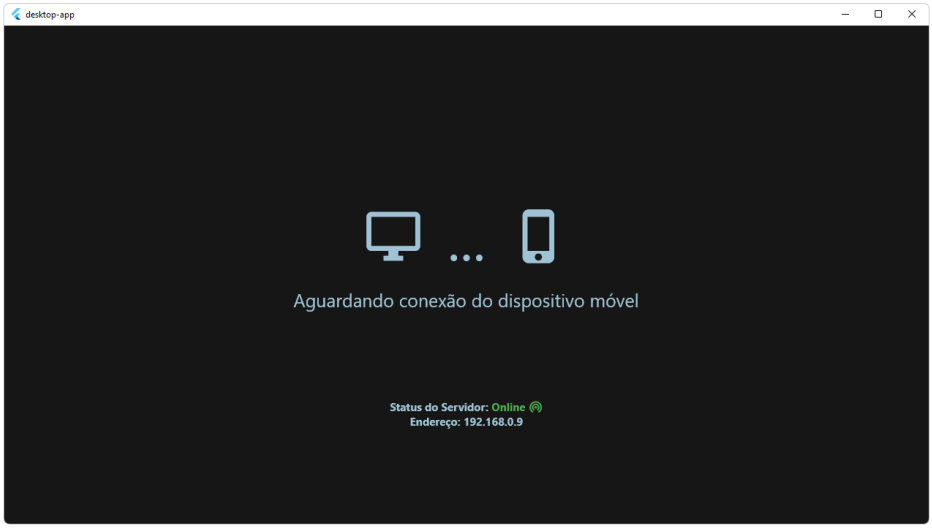
Para que esteja disponível para ambos os clientes, o servidor é iniciado automaticamente ao executar a aplicação *desktop*. A linguagem Dart[24] utilizada pelo

[23] https://www.json.org/json-en.html
[24] https://dart.dev

framework Flutter disponibiliza métodos como Process.run[25], capaz de manipular o terminal do *desktop* e disparar comandos para iniciar processos.

A Figura 7 demonstra a tela inicial da aplicação *desktop*.

Figura 7 – Tela inicial da aplicação *desktop*



Fonte: Autoria própria (2022)

5.4.2 Estabelecimento de conexão com o cliente

De modo a funcionar como um receptor de dados, após executar o aplicativo *desktop* e o servidor ser inicializado, a aplicação envia uma requisição de autenticação para o servidor TURN pelo endereço “https://localhost:8086/api/turn”.

Com suas credenciais autenticadas, o aplicativo *desktop* segue seu fluxo inicializando uma conexão de sí mesmo com o *websocket,* previamente liberado pelo servidor utilizando o método *“new”* e disponibilizando suas informações para o servidor com a finalidade de esperar uma oferta de conexão. Uma vez que o servidor está sendo executado pelo *desktop*, a conexão com o *websocket* é realizada pelo endereço “https://localhost:8086/ws”.

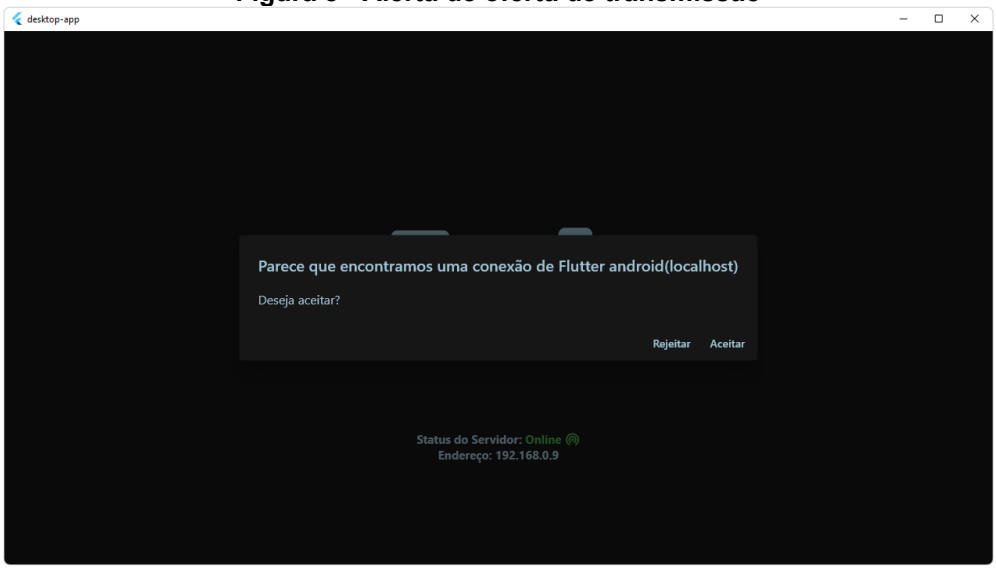
Após iniciar uma conexão com o *websocket,* diferentemente da aplicação móvel que é redirecionada para uma tela onde mostra-se o candidato disponível para receber uma transmissão (Figura 5.3), após enviar as informações para o servidor, a aplicação *desktop* mantém-se na tela inicial (Figura 7) até que um cliente *mobile*

[25] https://api.dart.dev/stable/2.18.2/dart-io/Process-class.html

conecte-se ao servidor e envie uma requisição de transmissão para o *desktop* utilizando o método *"Offer"* descrito no Quadro 8.

Após receber uma oferta, é apresentado para o usuário uma caixa de diálogo contendo o nome do ofertante e solicitando a aceitação ou rejeição da conexão, como demonstrado na Figura 8.

Figura 8 - Alerta de oferta de transmissão



Fonte: Autoria própria (2022)

O botão "Rejeitar" tem por finalidade informar ao servidor a rejeição da solicitação de transmissão utilizando o método *"Bye",* que por sua vez receberá o id do respectivo cliente (*desktop*) e o id da sessão em que o mesmo não deseja participar, anteriormente oferecida pela aplicação *mobile*. O trecho de código ilustrado no Quadro 9, demonstra a resposta em formato JSON para rejeição de transmissão.

Quadro 9 - Resposta de rejeição de transmissão

```
flutter: send: {
    "type": "bye",
    "data": {
        "session_id": "202625-082764",
        "from": "082764"
    }
}
```

Fonte: Autoria própria (2022)

Por outro lado, o botão “Aceitar” tem por finalidade informar ao servidor o desejo de aceitar a proposta de transmissão, consequentemente uma resposta SDP é criada imediatamente por parte do *desktop* e enviada para o servidor utilizando o método “*Answer*” apresentado no Quadro 8. O Quadro 10 demonstra o conteúdo do objeto de resposta SDP.

Quadro 10 - Resposta SDP

```
data: {
    "type": "answer",
    "data": {
        "to": "202625",
        "from": "082764",
        "description": {
            "sdp": "v=@\r\no=- 6419967442140687491 2 IN IP4 127.0.0.1\r\ns=-\r\nt=0
0\r\na=group:BUNDLE 0\r\na=extmap-allow-mixed\r\na=msid-semantic: WMS 3ED81C1B-C2BD-40EA-
9F74-9E59B9A726CA\r\nm=video 9 UDP/TLS/RTP/SAVPF 96 97 98 99 35 36 125 124 127\r\nc=IN IP4
0.0.0.0 \r\na=rtcp:9 IN IP4 0.0.0.0\r\na=ice-ufrag:SwSb\r\na=ice-
pwd:y52h6vDkUhIgd3iLbGDihGzB\r\na=ice-options: trickle\r\na=fingerprint:sha-256
5E:91:BC:E7:69:51:7A:C5:66:A5:AB:CO:B1:E7:8A:D5:EO:A7:BA:B7:D4:87:67:74:4C:2A:94:06:0A:E6:OD
:F2\r\na=setup:active\r\na=mid:0\r\na=extmap:1 urn:ietf:params:rtp-
hdrext:toffset\r\na=extmap:2 http://www.webrtc.org/experiments/rtp-hdrext/ abs-send-
time\r\na=extmap:3 urn:3gpp:video-orientation\r\na=extmap:4 http://www.ietf.org/id/draft-
holmer-rmcat-transport-wide-cc-extensions-01\r\na=extmap:5
http://www.webrtc.org/experiments/rtp-hdrext/playout-delay\r\na=extmap:6
http://www.webrtc.org/experiments/rtp-hdrext/video-content-type\r\na=extmap:7
http://www.webrtc.org/experiments/rtp-hdrext/video-timing\r\na=extmap:8
http://www.webrtc.org/experiments/rtp-hdrext/color-space\r\na=sendrecv\r\na=rtcp-
mux\r\na=rtcp-rsize\r\na=rtpmap:96 VP8/90000\r\na=rtcp-fb:96 goog-remb\r\na=rtcp-fb:96
transport-cc\r\na=rtcp-fb:96 ccm fir\r\na=rtcp-fb:96 nack\r\na=rtcp-fb:96 nack
pli\r\na=rtpmap:97 rtx/90000\r\na=fmtp:97 apt=96\r\na=rtpmap:98 VP9/90000\r\na=rtcp-fb:98
goog-remb\r\na=rtcp-fb:98 transport-cc\r\na=rtcp-fb:98 ccm fir\r\na=rtcp-fb:98
nack\r\na=rtcp-fb:98 nack pli\r\na=fmtp:98 profile-id=0\r\na=rtpmap:99
rtx/90000\r\na=fmtp:99 apt=98\r\na=rtpmap:35 AV1/90000\r\na=rtcp-fb:35 goog-remb\r\na=rtcp-
fb:35 transport-cc\r\na=rtcp-fb:35 ccm fir\r\na=rtcp-fb:35 nack\r\na=rtcp-fb:35 nack
pli\r\na=rtpmap:36 rtx/90000\r\na=fmtp:36 apt=35\r\na=rtpmap:125 red/90000\r\na=rtpmap:124
rtx/90000\r\na=fmtp:124 apt=125\r\na=rtpmap:127 ulpfec/90000\r\na-ssrc-group:FID 3908574709
1200 169243\r\na=ssrc:3908574709 cname: RnSw7atWfjszi7H4\r\na-ssrc:3908574709 msid:3ED81C1B-
C2BD-40EA-9F74-9E59B9A726CA C943576335BC-495C-AAE3-5EAFBD0530F9\r\na-ssrc: 1200169243
cname:RnSw7atwfjszi7H4\r\na=ssrc:1200169243 msid:3ED81C1B-C2BD-40EA-9F74-9E59B9A726CA
C9435763-35BC-495C-AAE3-5EAFBD0530F9\r\n",
            "type": "answer"
            },
        "session_id": "202625-682764"
    }
}
```

Fonte: Autoria própria (2022)

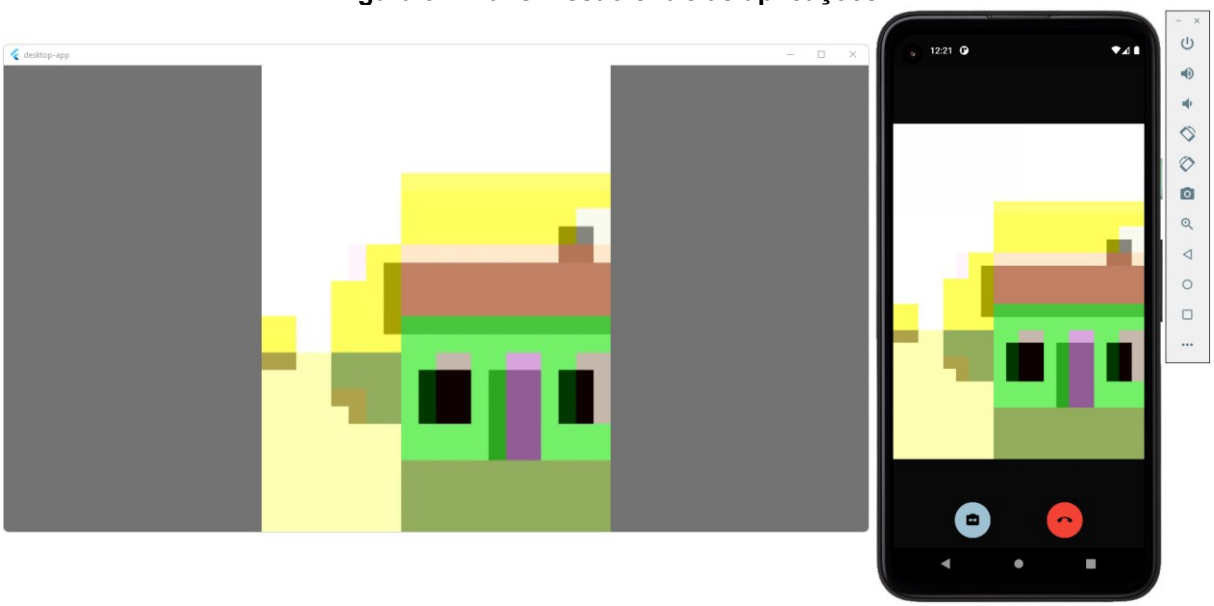
Deste modo, agora que os dispositivos se conhecem na rede e aceitaram transmitir dados entre si, assumindo uma comunicação segura entre ambas as partes, a biblioteca Flutter-WebRTC encarrega-se de estabelecer uma conexão UDP entre ambos os clientes. Mesmo que não sendo considerado um protocolo de transmissão muito confiável, o UDP é usado para estabelecer conexões de baixa latência contendo

tolerância a perdas de pacotes entre aplicações. Ou seja, o protocolo não oferece garantia de que os dados serão entregues aos destinos nem mesmo oferece recursos para retransmitir mensagens perdidas ou corrompidas.

Na grande maioria dos casos, conexões via WebRTC são utilizadas para transitar mídia entre dois(ou mais) pontos da sessão, ou seja, o mesmo cliente que envia sua transmissão de mídia para os participantes da sessão também recebe transmissões. Neste projeto, não. Assim como descrito anteriormente, o aplicativo *desktop* não envia nenhuma transmissão para a aplicação *mobile*, eliminando possíveis sobrecargas de dados na transmissão. Portanto, a função RTCVideoRenderer[26] é utilizada para reconhecer os dados de mídia sendo enviados pelo *smartphone* e a função RTCVideoView[27] é utilizada para demonstrar a mídia na tela da aplicação *desktop*.

A Figura 9 ilustra uma transmissão de mídia sendo executada entre as aplicações.

Figura 9 - Transmissão entre as aplicações



Fonte: Autoria propria (2022)

Por mais que não tenha um botão ou uma ferramenta de predefinição da qualidade de imagem na aplicação, por padrão, a aplicação foi desenvolvida utilizando

[26] https://pub.dev/documentation/flutter_webrtc/latest/flutter_webrtc/RTCVideoRenderer-class.html
[27] https://pub.dev/documentation/flutter_webrtc/latest/flutter_webrtc/RTCVideoView-class.html

a resolução 1920x1080(*Full HD*). Porém, a tecnologia WebRTC e sua implementação disponibilizada para o Flutter ajusta a resolução de uma transmissão de vídeo com base no desempenho da rede, ou seja, uma transmissão com condições de rede instáveis terá uma imagem com menor qualidade de imagem para garantir a entrega da transmissão.

Mesmo tendo a transmissão acontecendo entre cliente e servidor, um dos objetivos do projeto é fazer com o que *desktop* reconheça essa transmissão como uma imagem de *webcam* para que seja possível utilizar tal aplicação em cenários de videoconferência. Portanto, a seção a seguir discorre sobre tecnologias e métodos utilizados na solução encontrada para essa questão.

5.4.3 Manipulação da mídia de modo a ser exibida como *webcam*

Com a troca de informações estabelecida entre cliente e servidor e parte do problema resolvido, tem-se a utilização da aplicação como *webcam*. A abordagem utilizada para solucionar esse problema foi utilizar uma câmera virtual.

Uma vez que o *plugin* OBS-VirtualCam é capaz de disponibilizar uma *webcam* virtual para o usuário, ao iniciar a aplicação, juntamente com a inicialização do servidor comandos de terminal são disparados para que o *plugin* seja instalado no *desktop* que executa a aplicação.

Em busca de manter a qualidade local da imagem e partindo da ideia de que um vídeo é formado por uma série de imagens(*frames*) dando-lhe sensação de movimento (U; CH; K, 2021), com a transmissão de mídia acontecendo entre cliente e servidor e sendo demonstrada na tela da aplicação *desktop*, a abordagem utilizada no projeto foi desenvolver um *script* em linguagem Python com o propósito de executar diversas capturas de tela sobre a mídia que está sendo transmitida pela aplicação e enviá-las para a câmera virtual.

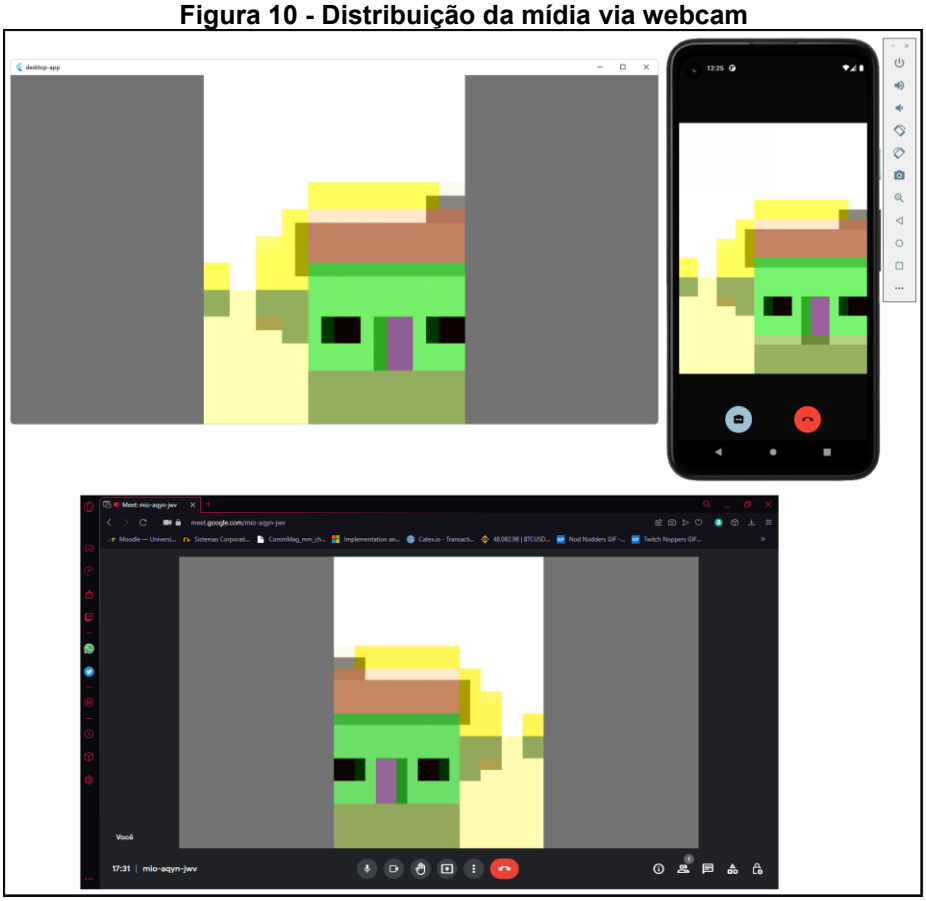
Esta abordagem foi possível ao utilizar a API PyWin32. Composta por diversos módulos, esta API presta suporte direto para as ferramentas do sistema operacional Windows. Bastante relevante no projeto, a escolha dessa ferramenta auxilia na performance, aumentando a velocidade em que as capturas de imagem são feitas influencia diretamente no *delay*(atraso) da transmissão além de capturar imagens de qualquer aplicação específica em execução pelo *desktop*.

Além do módulo em questão, foi necessário a utilização do mesmo em conjunto com a API PyVirtualCam. A API por sua vez tem por finalidade iniciar a

manipulação da câmera virtual (OBS-VirtualCam) com a função pyvirtualcam.Camera[28], e, enquanto a transmissão de mídia estiver acontecendo o *script* mantem-se em loop fazendo capturas de tela da mídia, ou seja, utiliza-se os recursos disponibilizados pelos módulos Win32Gui[29], Win32Ui[30], Win32Con[31] (alguns dos módulos que compõem a API PyWin32) para criar frames da transmissão e enviá-los para a câmera virtual pelo método send(frame)[28], de modo que a transmissão de mídia em tempo real da aplicação cliente-servidor seja demonstrada pela câmera virtual.

A Figura 10 esboça uma transmissão sendo executada pela aplicação, e a disponibilização da mesma via *webcam* em uma videoconferência via Google Meet[32].

Figura 10 - Distribuição da mídia via webcam

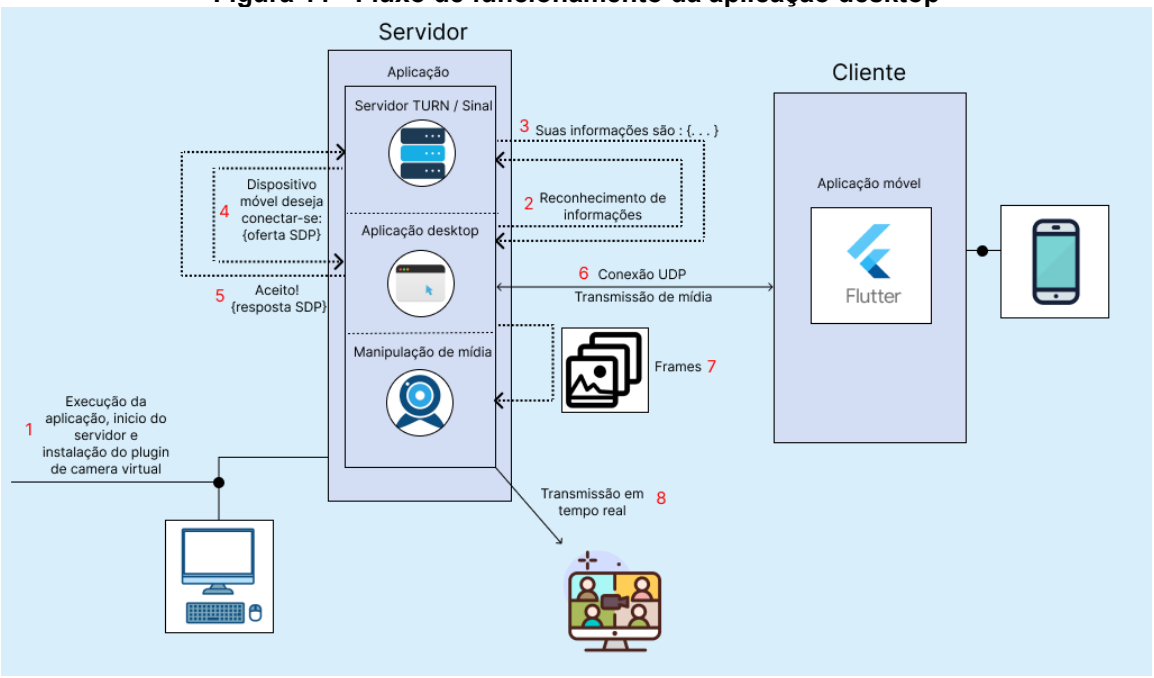


Fonte: Autoria própria (2022)

[28] https://letmaik.github.io/pyvirtualcam/
[29] http://timgolden.me.uk/pywin32-docs/win32gui.html
[30] https://mhammond.github.io/pywin32/win32ui.html
[31] https://www.programcreek.com/python/index/475/win32con.
[32] https://meet.google.com

A Figura 11 tem por finalidade demonstrar o fluxo de funcionamento da aplicação *desktop* e sua iteração com o servidor ilustrando o conteúdo presente nesta seção. Enumerando de 1 a 6 e acontecendo de maneira ascendente, na figura assume-se uma conexão requisitada previamente pela aplicação móvel.

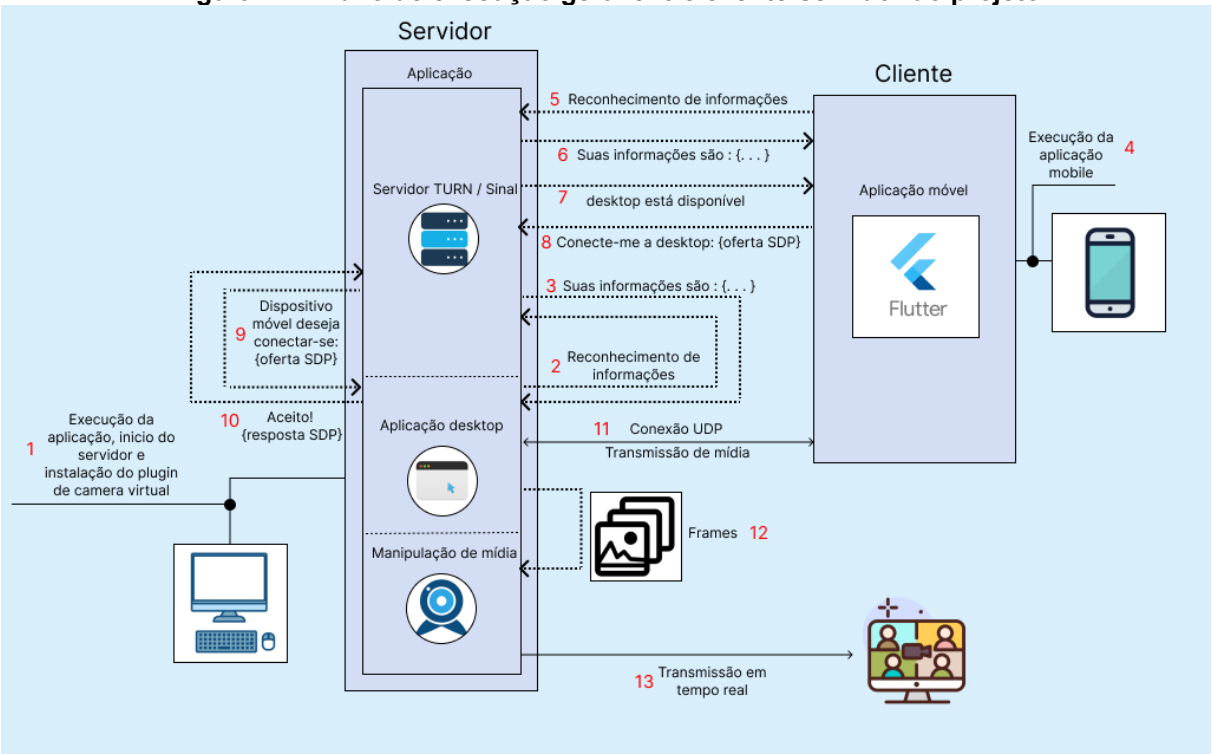
Figura 11 - Fluxo de funcionamento da aplicação desktop



Fonte: Autoria própria (2022)

Além de expressar de maneira ilustrativa o funcionamento da aplicação móvel e *desktop* individualmente, a Figura 12 esboça o cenário completo de funcionamento do projeto. Enumerado de 1 a 13, o fluxo de execução geral do projeto segue em ordem ascendente, desde a inicialização de ambas as aplicações, o estabelecimento de uma conexão, e a disponibilização da transmissão em tempo real como uma *webcam*.

Figura 12 - Fluxo de execução geral entre cliente-servidor do projeto



Fonte: Autoria própria (2022)

6 TESTES E RESULTADOS

Para avaliação da solução apresentada foram usadas três abordagens. Na primeira foi realizada uma análise técnica da solução. Na segunda, foi aplicado um questionário de percepção dos usuários. E por fim, foi realizado um teste as cegas.

6.1 Especificações de teste

É fato que diversos fatores externos podem afetar no desempenho de transmissões em tempo real, portanto, os testes foram expostos as mesmas condições, buscando evitar diferenças relevantes no desempenho das transmissões. Portanto, para apresentar os resultados, mesmo que trabalhando em diferentes cenários os testes foram realizados sobre as mesmas condições (*smartphones*, *desktop*, *Internet*), sendo comentadas a seguir.

A *Internet* utilizada para a transmissão demonstrou ter velocidade de *download* de cerca de 320 Mbps, velocidade de *upload* de cerca de 27 Mbps e uma latência de cerca de 14 ms. Para todos os cinco clientes (dispositivos que enviam a imagem para a transmissão, 3 *smartphones* e 2 *notebooks*) nos três diferentes testes, foi utilizada uma conexão *Wi-fi* 5 GHz, sendo uma conexão que possibilita uma alta velocidade na transmissão de dados.

Por parte dos cenários que utilizam a câmera do *smartphone* como webcam, o *desktop* utilizado para receber essas informações possui as seguintes especificações:

- CPU: Ryzen 5 5600X Six-Core.
- Memória RAM: 16GB (*dual-channel*).
- GPU: Geforce GTX 1060 6GB.
- Armazenamento: SSD 600GB (1x 120GB, 1x 480GB).
- SO: Windows 11 Pro versão 21h2.

Os smartphones utilizados nos experimentos foram do modelo Samsung *Galaxy S10*, *Iphone 7* e *Iphone 8*. As respectivas especificações dos aparelhos estão descritas no Quadro 11.

Quadro 11 - Smartphones utilizados para teste

	CPU	Memória RAM	**GPU**	**Armazenamento**	SO	**Câmeras**
Samsung Galaxy S10	Octa-core (2x2.73 GHz Mongoose M4 & 2x2.31 GHz Cortex-A75 & 4x1.95 GHz Cortex-A55)	6GB	Mali-G76 MP12	128GB	Android 12	Frontal: 10 MP, traseira: 12MP.
Iphone 7	Quad-core 2.34 GHz (2x Hurricane + 2x Zephyr)	2GB	PowerVR Series7XT Plus (six-core graphics)	32GB	iOS 15.7	Frontal: 7 MP, traseira: 12MP.
Iphone 8	Hexa-core (2x Monsoon + 4x Mistral)	2GB	Apple GPU (three-core graphics)	64GB	iOS 16	Frontal: 7 MP, traseira: 12MP.

Fonte: Autoria própria (2022)

Assim como demonstrado para o *desktop* e para os *smartphones*, segue as especificações do *notebook* utilizado no experimento.

Quadro 12 - Especificações dos *notebooks*

	CPU	Memória RAM	**GPU**	**Armazenamento**	SO	**Câmera**
Notebook “1”	Core i5-1035G1 Quad-Core	8GB	Intel UHD Graphics	SSD 256GB	Windows 10 Pro versão 21h2	0.3MP
Notebook “2”	Core i5-10210U Quad-core	8GB	Intel UHD Graphics	SSD 256GB	Windows 10 Pro versão 21h2	1MP

Fonte: Autoria própria (2022)

Além das especificações supracitadas, considera-se que testes foram feitos utilizando as versões gratuitas dos aplicativos em questão. Todavia, não inviabiliza a solução, uma vez que as versões gratuitas são bastante usadas pelo público e o TCCam é uma aplicação de código aberto.

6.2 Benchmark

Neste cenário aqui presente, analisou-se o desempenho do *desktop* enquanto realizava-se transmissões via aplicativos. O teste demonstrou uma comparação que acontece diretamente entre o aplicativo Camo e o TCCam enquanto em uma transmissão, onde ambos os monitoramentos de desempenho ocorreram durante 5 minutos.

Não foram feitas comparações com as outras aplicações por impossibilidade de captura de dados. Ao utilizar as demais aplicações, as mesmas não foram reconhecidas pela ferramenta de teste, dificultando o reconhecimento das informações necessárias.

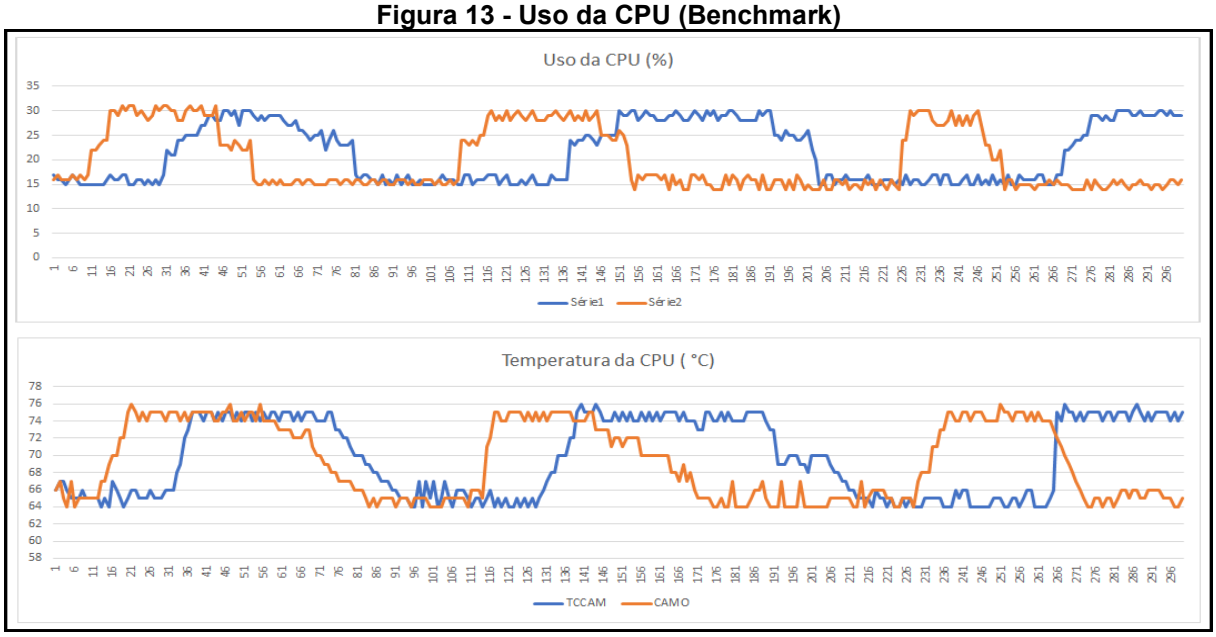
O monitoramento acontece enquanto o *desktop* trabalha sobre os processos padrões para execução do sistema operacional em conjunto com o navegador OperaGx[33] e o sistema de videoconferência Zoom[34]. O *desktop* e *smartphone* (Samsung Galaxy s10) utilizados no experimento estão descritos na seção 6.1.

Os gráficos relacionados ao experimento são demonstrados a seguir, separados em CPU, GPU, RAM e FPS, de modo que o TCCam é represetando pela cor azul enquanto que o Camo é representado pela cor alaranjada.

A Figura 13, demonstra dois gráficos relacionados ao uso da CPU durante o teste, assim como uma captura de tela referente ao experimento é demonstrada no Apêndice A, Figura 28.

[33] https://www.opera.com/pt-br/gx
[34] https://zoom.us

Figura 13 - Uso da CPU (Benchmark)



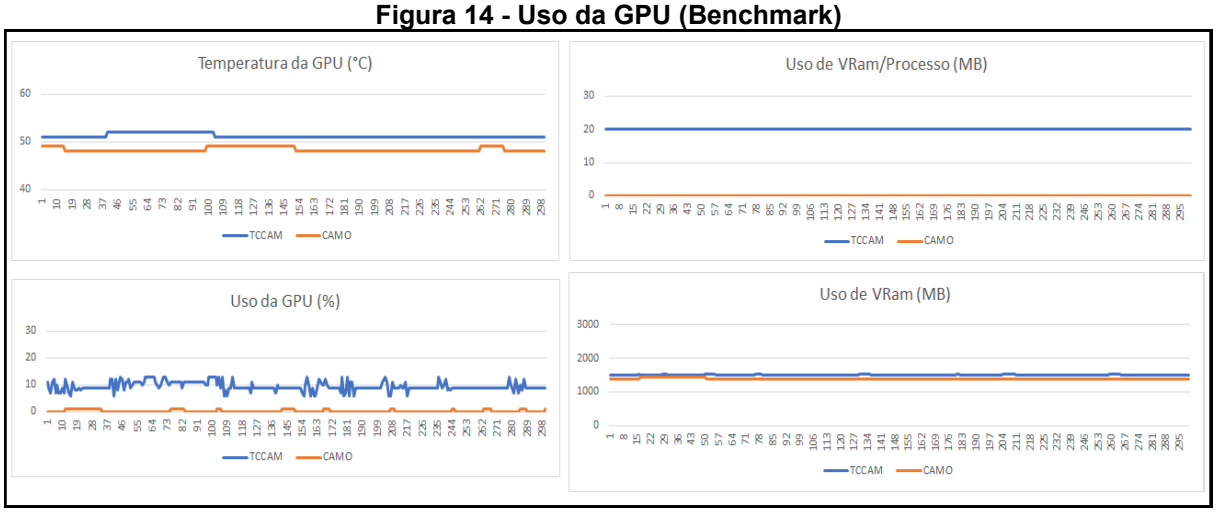
Fonte: Autoria própria (2022)

Com base no gráfico de “Uso da CPU” nota-se um padrão de desempenho por parte da CPU, atingindo um uso mínimo de 15% e máximo de 31%. Apesar de não apresentar valores preocupantes, nota-se no gráfico “Temperatura da CPU” a influência direta que a % de uso tem sobre a temperatura.

Mesmo que em poucas vezes, a temperatura da CPU atingiu o valor máximo de 76 °C, apresentando uma temperatura mínima de 64 °C. Enquanto com o TCCam a CPU apresentou uma temperatura média de 69.7 °C, com o Camo a CPU apresentou uma média de 69.2 °C.

A Figura 14 demonstra os gráficos referentes ao uso da GPU durante o experimento.

Figura 14 - Uso da GPU (Benchmark)



Fonte: Autoria própria (2022)

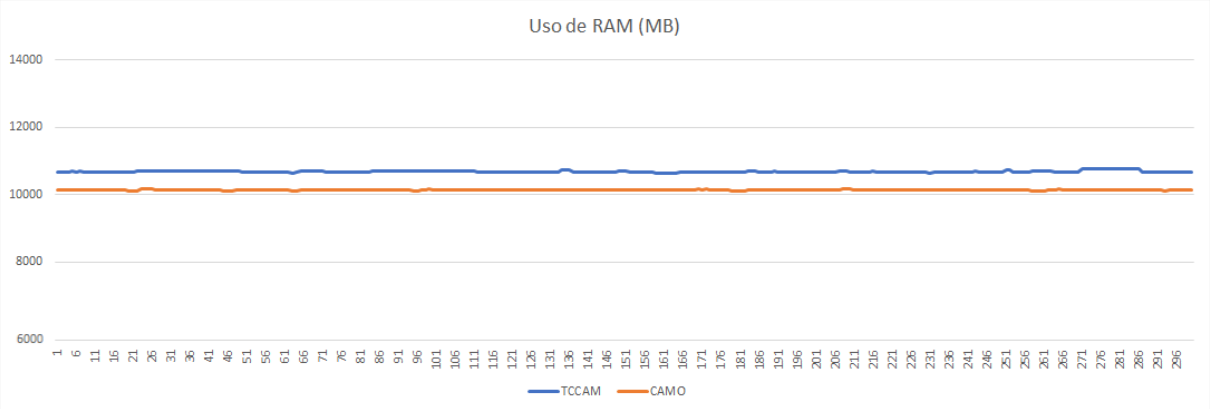
Ao analisar o desempenho da GPU no experimento, nota-se que em ambos os casos a utilização deste recurso é bem baixa. Nota-se no gráfico “Uso da GPU” que no cenário utilizando o TCCam é utilizando em média 9.5% do recurso durante o experimento. A leve utilização do recurso é demonstrada no gráfico “Uso de VRam/Processo”, onde mostra que a solução utiliza 20MB da VRAM, enquanto que no cenário utilizando o Camo, o uso do recurso é nulo.

Mesmo que com um uso nulo do recurso, o gráfico “Uso de VRam” demonstrou que em média 1390.9 MB da VRam foi utilizada durante o monitoramento do Camo, sendo um recurso provavelmente utilizado por outros processos em execução no sistema.

Com uma utilização da GPU levemente maior por parte do TCCam, a temperatura em ambos os casos se manteve bastante estável atingindo um valor máximo de 52 °C.

A seguir, a Figura 15 tem por finalidade demonstrar o gráfico referente ao uso de memória RAM durante o monitoramento.

Figura 15 - Uso de memória RAM (Benchmark)

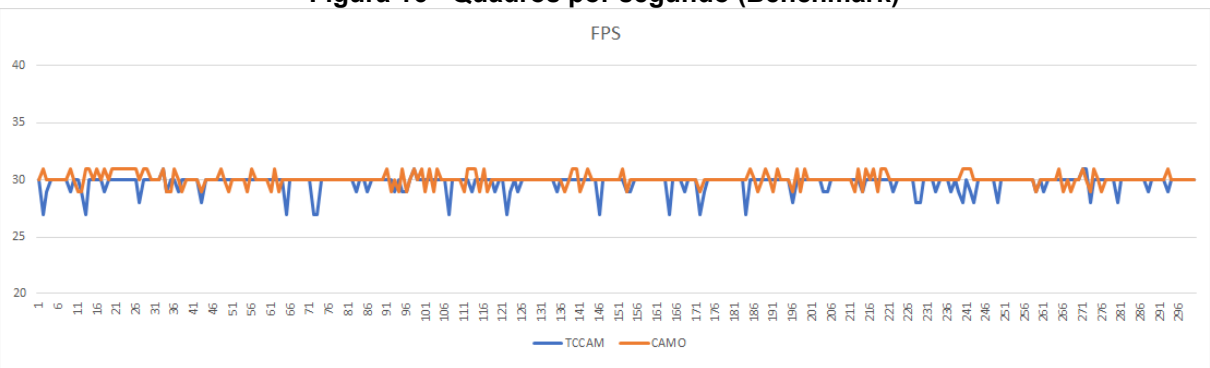


Fonte: Autoria própria (2022)

Ao analisar o gráfico é importante ressaltar que pela estrutura da solução, mais memória RAM é consumida durante sua utilização. Enquanto o experimento utilizando o Camo utilizou uma média de 10137.9 MB de memória RAM, o experimento utilizando o TCCam utilizou em média 10685.4 MB, demonstrando um consumo médio cerca de 547,5 MB superior, tratando-se de um valor considerável.

Por fim, a Figura 16 demonstra o gráfico de quadros por segundo de ambos os cenários.

Figura 16 - Quadros por segundo (Benchmark)



Fonte: Autoria própria (2022)

Ao analisar o gráfico, entende-se que em ambos os cenários a quantidade de quadros por segundo entregues foi bastante estável. Enquanto o TCCam teve uma média de 29.7 FPS enquanto o Camo teve uma média de 29.8 FPS. É importante ressaltar a performance da solução nessa questão, tendo em vista a abordagem utilizada no desenvolvimento (Seção 5.4.3).

Com base nos dados demonstrados, não notou-se baixo desempenho por parte do *desktop* utilizado no teste. Se mantendo estável, por mais que consumindo uma quantidade considerável a mais de memória RAM ao utilizar o TCCam, o cenário analisado não apresentou travamentos ou desempenho que influenciasse negativamente no comportamento da máquina.

6.3 Questionário de avaliação dos usuários

Para a aplicação desta metodologia de avaliação do projeto, foram criados três cenários de transmissão diferentes: Um cenário utilizando uma transmissão via *webcam* do *notebook* e os outros dois cenários são utilizando aplicativos para transmissão da câmera do *smartphone* como *webcam*, um deles sendo o projeto aqui proposto.

Com as especificações de *notebook*, *smartphone* e *desktop* presentes na seção 6.1, nesta abordagem foram utilizados três cenários que tem por definição:

- Cenário “A”: A utilização do *notebook “1”* como transmissor da imagem.
- Cenário “B”: A utilização de um *smartphone (iphone 8)* como transmissor da imagem em conjunto com o *desktop*, utilizando o aplicativo iVCam[35].
- Cenário “C”: A utilização de um *smartphone* (*Samsung Galaxy s10*) como transmissor da imagem em conjunto com o *desktop*, utilizando o TCCam.

Com isso, foi realizada uma videoconferência via Google Meet com aproximadamente 16 participantes de diferentes regiões do país incluindo participantes residindo no país de Portugal. A transmissão teve por foco ensinar os participantes a resolver um cubo mágico e sua duração foi de aproximadamente 10(dez) minutos, onde, cada cenário teve cerca de 3 minutos de duração.

Foi solicitado aos participantes que analisassem o desempenho da transmissão, onde, o experimento iniciou com o cenário “A” e cerca de três minutos depois passou-se a utilizar o cenário “B”, por fim cerca de três minutos depois de iniciar o cenário “B” passou-se a utilizar o cenário “C”.

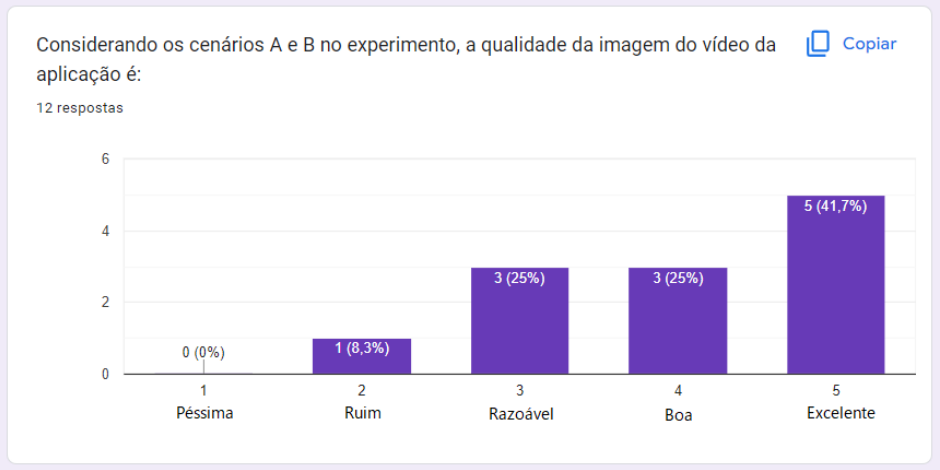
Ao final do experimento, foi elaborado um questionário com questões referentes aos cenários apresentados na transmissão e foi solicitado aos participantes fosse respondido coletando *feedbacks*.

[35] https://www.e2esoft.com/ivcam/

O questionário apresenta 6 perguntas e para validação utilizam a escala Likert assumindo valores de 1 a 5. Respondido por 12 pessoas, os resultados são demonstrados nas Figuras 17, 18, 19, 20, 21 e 22, enquanto que uma captura de tela do experimento é ilustrada no Apêndice A, Figura 29.

A Figura 17 demonstra que 41,7% dos voluntários consideraram a imagem do vídeo da aplicação excelente. 91,7% dos voluntários consideraram a imagem razoável ou superior, enquanto que 8,3% (1 voluntário) considerou a imagem ruim.

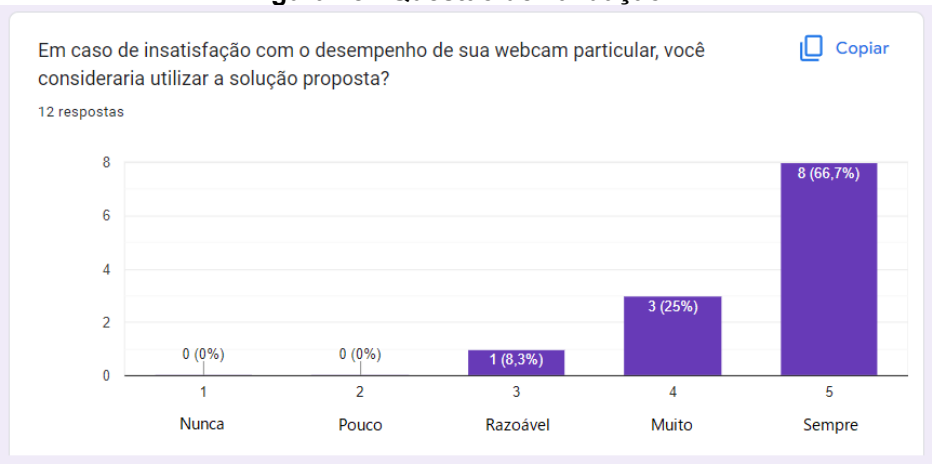
Figura 17 - Questão de validação 1



Fonte: Autoria própria (2022)

Com base nas informações demonstradas na Figura 18, nota-se a aceitação considerável da solução por parte dos voluntários, tendo em vista que 8 dos 12 participantes optariam por sempre utilizar a solução proposta em caso de insatisfação com o desempenho da *webcam* particular, enquanto que entre os outros 4 participantes restantes, 3 considerariam muito utilizar a solução.

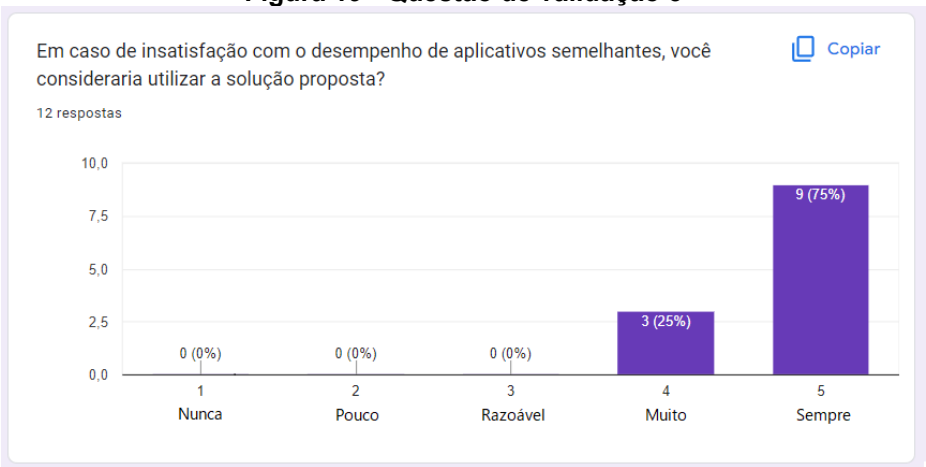
Figura 18 - Questão de validação 2



Fonte: Autoria própria (2022)

A Figura 19 demonstra que 100% dos usuários contentaram-se com o desempenho da solução, de modo que todos consideraram muito, ou, sempre a utilização da solução em relação a outros aplicativos semelhantes.

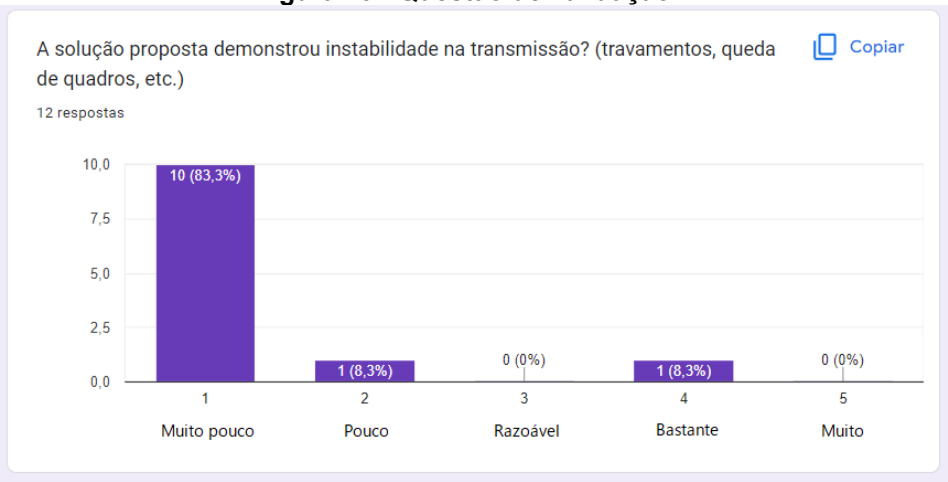
Figura 19 - Questão de validação 3



Fonte: Autoria própria (2022)

Com base no gráfico demonstrado na Figura 20, nota-se um bom desempenho apresentado por parte da solução, onde 91,7% dos voluntários disseram que o TCCam apresentou pouca ou muito pouca instabilidade na transmissão, enquanto que um caso isolado demonstrou presenciar bastante instabilidade.

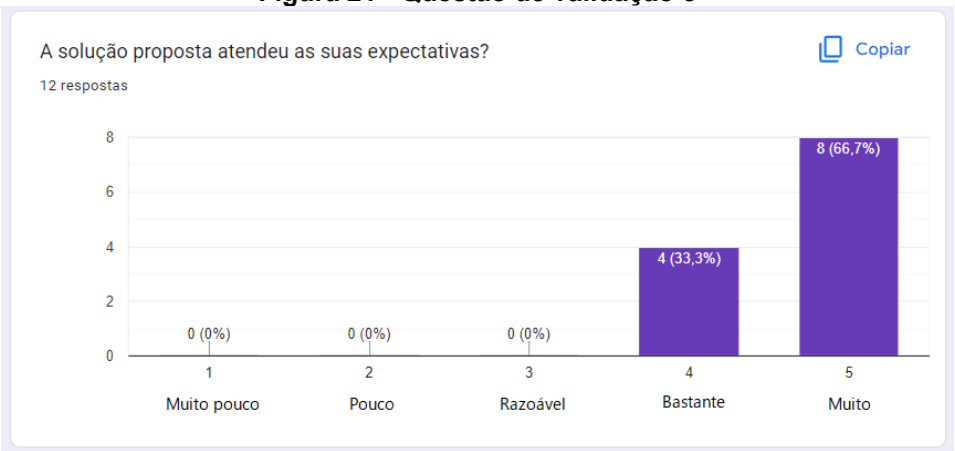
Figura 20 - Questão de validação 4



Fonte: Autoria própria (2022)

A Figura 21 ilustra uma questão referente as expectativas dos voluntários, onde 66,7% dos assumiram que a solução atendeu muito suas expectativas, enquanto que 33% disseram atender bastante.

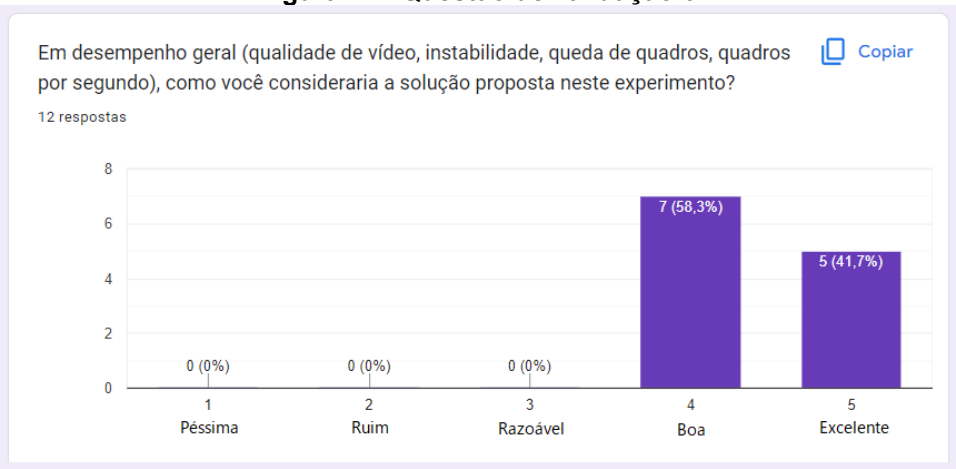
Figura 21 - Questão de validação 5



Fonte: Autoria própria (2022)

Por fim, a Figura 22 demonstra uma questão referente ao desempenho geral da solução no experimento. Nota-se que, 100% dos voluntários presenciaram um bom desempenho, ou, excelente.

Figura 22 - Questão de validação 6



Fonte: Autoria própria (2022)

O resultado desta metodologia demonstra boa parte de contentamento com a aplicação. Também nota-se nas questões 1 e 4 os casos isolados de baixo rendimento. Infelizmente, assim como citado anteriormente, transmissões de vídeo em tempo real envolvem muitas variáveis. Falhas e baixo rendimento podem ser ocorridos tanto pelo lado transmissor quanto pelo lado dos receptores da transmissão, neste caso podendo ser causado até pela distância entre as trocas de informações ou possíveis instabilidades na Internet do participante.

Mesmo que com casos isolados de baixo rendimento, as questões 1 e 4 apresentam dados importantes como por exemplo, 66,7% dos voluntários entendem que a aplicação teve uma qualidade de transmissão boa ou excelente e 83,3% disseram que a transmissão apresentou uma instabilidade muito baixa.

O questionário foi respondido de forma anônima e não foram solicitados dados pessoais dos voluntários, evitando qualquer tipo de influência nas respostas.

É importante ressaltar que de acordo com as questões 2, 3 e 6 o TCCam demonstrou boa aprovação por parte dos interrogados, e que segundo a questão 5, 100% dos interrogados estão satisfeitos em relação as suas expectativas.

Porém, mesmo que sendo um teste bem sucedido, ao aplicar esta metodologia notou-se que os voluntários tiveram dificuldades para conseguir comparar os cenários de maneira correta ou com coerência já que foram apresentados de maneira isolada. Portanto, apresenta-se outra proposta de validação.

6.4 Comparação de qualidade de vídeo

Este teste tem como base um “teste as cegas” e é uma terceira metodologia de validação para a solução, tendo em vista que a primeira proposta apresentou bons resultados.

Para a aplicação desta metodologia de avaliação do projeto, assim como na primeira, foram criados quatro diferentes cenários de transmissão. Seguindo o mesmo conceito, dividem-se em 1 cenário utilizando uma transmissão via *webcam* do *notebook* e os outros 3 cenários utilizando aplicativos para transmissão da câmera do smartphone.

As especificações de *notebook*, *smartphones* e *desktop* estão presentes na seção 6.1, e, nesta abordagem foram utilizados os cenários:

- Cenário “A”: A utilização do *notebook “2”* como transmissor da imagem.
- Cenário “B”: A utilização do *iPhone 7* como transmissor da imagem utilizando o aplicativo DroidCam.
- Cenário “C”: A utilização do um *iPhone 8* como transmissor da imagem utilizando o aplicativo EpoCcam.
- Cenário “D”: A utilização do *Samsung Galaxy S10* como transmissor da imagem utilizando o TCCam.

Com os cenários estabelecidos, este teste teve por finalidade avaliar o desempenho dos cenários demonstrados de maneira mais visível e de fácil comparação.

Para inicializar o teste foi criado uma videoconferência via Google Meet, onde os quatro cenários transmitiram a mesma imagem ao mesmo tempo de ângulos pouco diferentes, desta vez, o vídeo consistiu em ficar rotacionando o cubo mágico de maneira aleatória, fazendo com o que o voluntário não prestasse atenção nas dicas de como resolvê-lo mas sim no desempenho das transmissões.

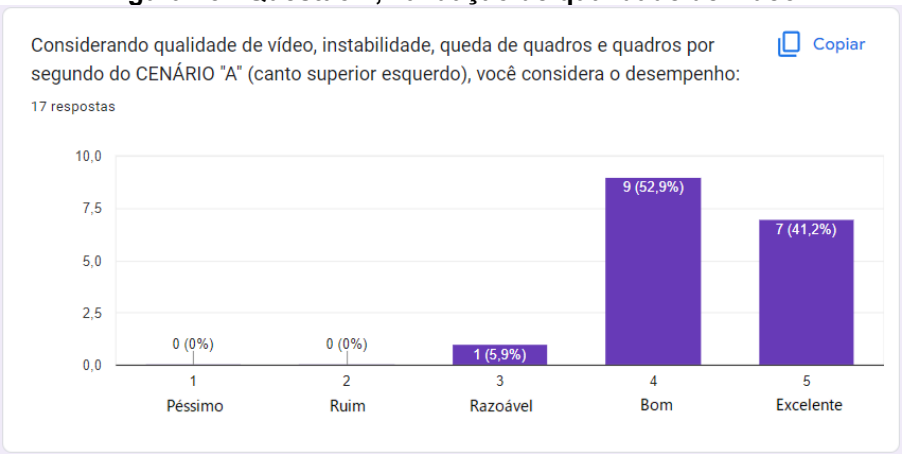
A transmissão foi gravada e adicionada ao site *youtube*[36], posteriormente o vídeo foi disponibilizado em uma comunidade de desenvolvedores para 17 voluntários de maneira com que nenhum soubesse sobre as especificações dos cenários ou a ordem de cada um deles.

[36] https://www.youtube.com/watch?v=s9jbwatTsUA

Sem nenhum tipo de auxílio exterior ou dica sobre o teste sendo executado, outro formulário foi realizado e enviado para os voluntários de maneira que nas perguntas 1, 2, 3, 4 utiliza-se a escala Likert para que os quatro cenários sejam avaliados individualmente de uma maneira geral, e por fim, a pergunta 5 apresenta uma questão direta sobre qual dos quatro cenários o voluntário escolheria utilizar. Uma captura de tela do experimento é ilustrada no Apêndice A, Figura 30.

A Figura 23 demonstra a conclusão dos voluntários com base no cenário “A”. Nota-se um bom desempenho entregue pelo cenário descrito, onde 94,1% dos 16 voluntários consideraram a qualidade da transmissão boa ou excelente.

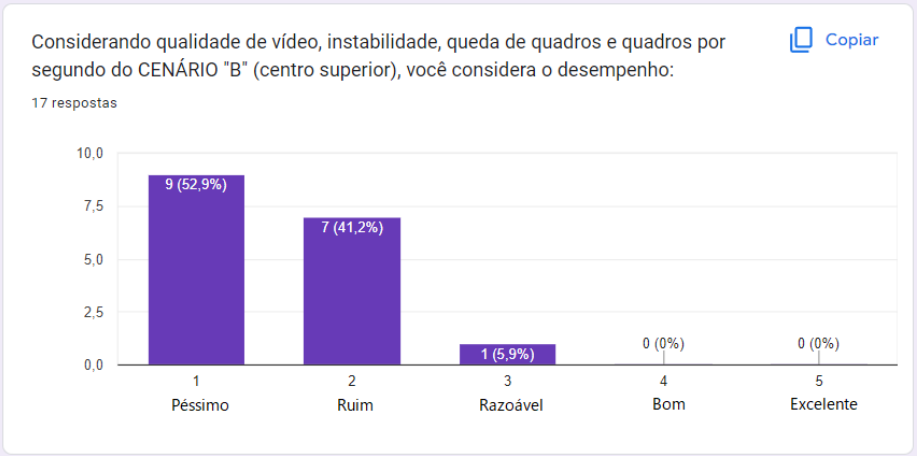
Figura 23 - Questão 1, validação de qualidade de vídeo



Fonte: Autoria própria (2022)

É fato que, demonstrando um desempenho consideravelmente pior, o cenário “B” não recebeu nenhum voto assumindo uma transmissão boa ou excelente, e, 52,9% dos voluntários consideraram uma qualidade de transmissão péssima. A Figura 24 demonstra os resultados do cenário “B”.

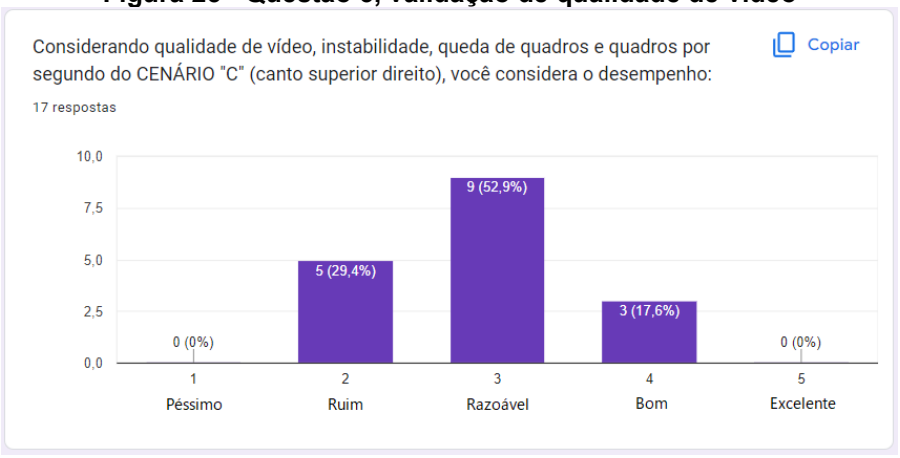
Figura 24 - Questão 2, validação de qualidade de vídeo



Fonte: Autoria própria (2022)

O cenário “C” demonstrou um desempenho razoável, onde 52,9% dos voluntários marcaram a opção “razoável” e os outros 47,1% distribuíram-se entre “bom” e “ruim”, ainda sim a maior parte optou por uma transmissão “ruim”. A Figura 25 demonstra os respectivos resultados do cenário “C”.

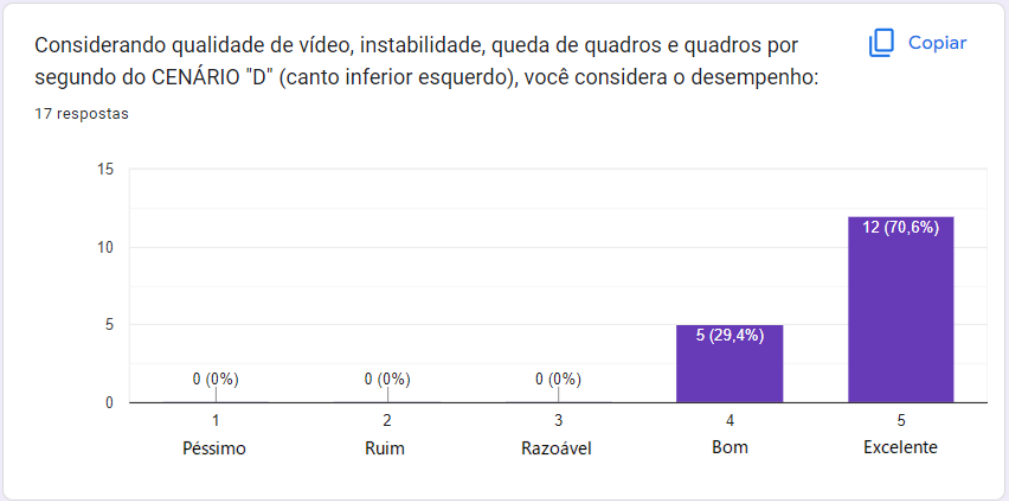
Figura 25 - Questão 3, validação de qualidade de vídeo



Fonte: Autoria própria (2022)

O cenário “D” demonstrou grande parte de aprovação e satisfação por parte dos voluntários, de modo que 70,6% assumiram classificar a transmissão como “excelente”, enquanto que os outros 29,4% classificaram como uma boa transmissão. A Figura 26 demonstra os resultados da questão 4.

Figura 26 - Questão 4, validação de qualidade de vídeo



Fonte: Autoria própria (2022)

Por fim a questão 5 foi formulada de modo que os voluntários apontassem o cenário que utilizariam em caso de necessidade, demonstrado na Figura 27.

Figura 27 - Questão 5, validação de qualidade de video

Fonte: Autoria própria (2022)

Os resultados deste teste foram expressivamente mais satisfatórios que os da seção 6.2.

Atenta-se ao fato de analisar as questões 2 e 3, onde demonstra-se a insatisfação dos voluntários com os cenários “B” e “C” assumindo um dos propósitos da pesquisa que é o baixo desempenho de aplicações cuja a função é a mesma deste projeto. Outro ponto importante é o fato de que a câmera do notebook 2 é de 1mp, ou seja, resolução HD (720p), e mesmo que as câmeras de todos os smartphones sejam

superiores, segundo a avaliação dos voluntários, a câmera do *notebook "2"* teve um desempenho consideravelmente superior ao das aplicações similares, ficando atrás apenas da solução proposta.

A questão 5, alinhando-se aos objetivos do projeto, demonstra que 70,6% dos voluntários (12 entre 17) escolheriam utilizar o TCCam dentre as opções demonstradas, assim sendo, entende-se o fato de que o aplicativo cumpriu seu objetivo.

7 CONCLUSÃO

Este trabalho propôs o desenvolvimento de uma solução que auxilia pessoas que não possuem uma *webcam* ou estão insatisfeitas com a qualidade de aplicativos externos para usar a câmera do *smartphone* como uma, visando um aplicativo de código aberto capaz de transmitir a imagem da câmera do *smartphone* como *webcam*.

O aplicativo permite ao usuário a praticidade de ter um *smartphone* por perto fazendo com que ele utilize dos recursos dispostos pelo mesmo para uma transmissão em tempo real.

As aplicações implementaram, com êxito, todos os módulos apresentados, partindo do estabelecimento de uma conexão até a demonstração da imagem como uma *webcam*. A aplicação *desktop*(servidor) construiu uma estrutura de modo que a aplicação mobile(cliente) consigue fornecedor todos os dados necessários para que a imagem da câmera do smartphone seja disponibilizada via *webcam*.

Utilizando uma abordagem de desenvolvimento multiplataforma com o Flutter, a sequência de passos utilizada para iniciar uma transmissão de dados entre as aplicações foi baseada na tecnologia WebRTC. Mesmo que a tecnologia seja comumente usada para transmissões entre navegadores, aplicando-se ao cenário aqui demonstrado, a tecnologia demonstrou todos os componentes necessários para que o projeto apresente uma boa qualidade de transmissão.

O projeto teve início com estudos sobre protocolos de transmissões em tempo real e suas aplicações. Posteriormente, foi realizado um mapeamento de ferramentas *open-source* voltadas para a tecnologia WebRTC. Sendo assim, o projeto seguiu com o estudo e manipulação dos códigos base disponibilizados pelo projeto *flutter-webrtc,* sendo possível o estabelecimento de uma conexão entre duas aplicações rodando em sistemas distintos.

Após o cenário descrito, seguiu-se com estudos referentes a manipulação da imagem da *webcam* que por fim teve sua construção utilizando o contexto de câmeras virtuais com dependência do plugin OBS-VirtualCam. Com todos os requisitos para o aplicativo realizados, sucedeu o desenvolvimento da aplicação de maneira que fosse possível a manipulação de dados provindos da aplicação *mobile* como uma *webcam*.

Assim como um dos pilares do projeto, as ferramentas e dependências utilizadas nas aplicações são *open-source*, não exigindo nenhuma licença ou

pagamento para utilização/implementação, logo não houveram gastos durante o projeto.

O projeto tinha por objetivos específicos: realizar um mapeamento da literatura sobre protocolos de transmissão de vídeo em tempo real; analisar a arquitetura dos softwares similares já existentes no mercado; desenvolver a aplicação cliente(*smartphone*)-servidor(*desktop*); realizar uma rotina de testes, tal como analisar o desempenho do sistema em diferentes videoconferências e criar diferentes cenários de comparação. Com base nas informações supracitas dentre os tópicos aqui presentes, pode-se afirmar que todos os objetivos do trabalho foram atingidos, uma vez que com base nos estudos de transmissões e arquiteturas de trabalhos similares as aplicações foram implementadas e testadas com êxito.

7.1 Trabalhos futuros

Embora o trabalho tenha cumprido todos os seus objetivos, há implementações adicionais que poderiam ser feitas.

Para o futuro, pode-se pensar na transmissão do microfone. Embora não seja alinhada com a proposta do trabalho que foca apenas na transmissão de vídeo, alguns dos aplicativos semelhantes ao TCCam disponibilizam o uso do microfone na aplicação. Outro ponto importante seria desenvolver um botão para que o usuário escolha a resolução da transmissão, de modo a adaptar-se a usuários com infraestrutura (internet, computador e afins) melhores ou piores. Assim como uma das propostas da utilização de câmeras virtuais, outra ideia interessante seria disponibilizar filtros para os usuários, para fins de manipulação da imagem.

Por falta de suporte da própria tecnologia flutter-webrtc, o projeto não dispõe de suporte para sistemas operacionais Linux. Por outro lado, por falta de estrutura e limitações financeiras, a aplicação precisa ser testada e analisada corretamente para a plataforma *iOS*. Por mais que a tecnologia e sua documentação garanta a funcionalidade nestes cenários, nenhum teste foi executado.

Além dos testes utilizando a plataforma iOS, há a necessidade de aplicar os testes com as versões pagas das aplicações demonstradas, mesmo que os testes com as versões gratuitas não inviabilizem a solução.

Uma questão interessante para se analisar é a disponibilização da imagem via *webcam* utilizando DLL, ou o desenvolvimento de um driver próprio utilizando o DirectShow.

Por fim, disponibilizar o arquivo instalador para *desktop* em uma página web voltada para a aplicação além de publicar o aplicativo móvel na *PlayStore* e *AppStore*, para que mais usuários consigam desfrutar da solução aqui apresentada.

REFERÊNCIAS

AKSOY, Cevat Giray *et al.* Working from Home Around the World. **NATIONAL BUREAU OF ECONOMIC RESEARCH**, Cambridge, Setembro 2022.

ANDERSON, Roy M *et al.* How will country-based mitigation measures influence the course of the COVID-19 epidemic? **The Lancet**, 9 Março 2020. 931-934.

BEGEN, Ali *et al.* SDP: Session Description Protocol. **Internet Engineering Task Force (IETF)** , Janeiro 2021.

BEGEN, Ali C; AKGUL, Tankut ; BAUGHER, Mark. Watching Video over the Web: Part 1: Streaming Protocols. **IEEE Internet Computing**, 15 Maio 2011. 54-63.

BRINGIT. Webcam para PC: tudo o que você precisa saber antes de comprar. **BringIt**, 2020. Disponivel em: https://www.bringit.com.br/blog/manutencao-de-notebook/webcam-para-pc-tudo-o-que-voce-precisa-saber-antes-de-comprar/. Acesso em: 7 nov. 2021.

CAMO. Use your phone as a webcam to look amazing on video calls. **Reincubate**, 2021. Disponivel em: https://reincubate.com/camo/. Acesso em: 10 Novembro 2021.
DEV47APPS. DroidCam - Use Your Phone as a Webcam. **Dev47Apps**, 2021. Disponivel em: https://www.dev47apps.com. Acesso em: 15 nov. 2021.

E2ESOFT. Virtual Camera. **e2esoft**, 2021. Disponivel em: https://www.e2esoft.com/vcam/. Acesso em: 28 ago. 2022.

E2ESOFT. iVCam. **E2ESOFT**, 2022. Disponivel em: https://www.e2esoft.com/ivcam/. Acesso em: 11 nov. 2022.

ELGATO. EpocCam - Difference Between Free and Pro Versions. **ElGato**, 2021. Disponivel em: https://help.elgato.com/hc/en-us/articles/360048912471-EpocCam-Difference-Between-Free-and-Pro-Versions. Acesso em: 26 out. 2021.

ELGATO. EpocCam – Turn Your Phone Into a Webcam.. **ElGato**, 2021. Disponivel em: https://www.elgato.com/en/epoccam. Acesso em: 1 nov. 2021.

GÓES, Geraldo S.; MARTINS, Felipe dos S.; NASCIMENTO, José Antônio S. Trabalho remoto no Brasil em 2020 sob a pandemia do Covid-19: quem, quantos e onde estão? **ipea**, 2021. Disponivel em: https://www.ipea.gov.br/cartadeconjuntura/index.php/2021/07/trabalho-remoto-no-brasil-em-2020-sob-a-pandemia-do-covid-19-quem-quantos-e-onde-estao/. Acesso em: 9 set. 2022.

GONZÁLEZ, Iván Santos *et al.* Implementation and Analysis of Real-Time Protocols. **Sensors**, 12 Abril 2017.

HAMILTON, William Alexander; GARRETSON, Oliver ; KERNE, Andruid. Streaming on twitch: fostering participatory communities of play within live mixed media. **CHI '14: Conference on Human Factors in Computing Systems**, Toronto, Abril 2014. 1315-1324.

HINER, Jason. Apple's new 2020 MacBook Air left out a key upgrade for people working from home. **CNET**, 2020. Disponivel em: https://www.cnet.com/tech/computing/apples-new-2020-macbook-air-left-out-a-key-upgrade-for-people-working-from-home/. Acesso em: 23 mar. 2022.

KUMAR, Prashant *et al.* fybrrStream: A WebRTC based Efficient and Scalable P2P Live Streaming Platform. **International Conference on Computer Communications and Networks**, Julho 2021.

LI, Gaohe *et al.* Development and Research Based on WebRTC Mobile Phone Video Communication. **2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)**, Chengdu, 15 Março 2019.

LOGITECH. WEBCAMS VS. CÂMERAS DE LAPTOP: A VERDADEIRA DIFERENÇA.Como uma webcam específica para essa finalidade melhora a experiência da reunião e aumenta a adoção de vídeo. **Logitech**, 2021. Disponivel em: https://www.logitech.com/pt-br/video-collaboration/resources/think-tank/articles/webcams-versus-laptop-cameras.html. Acesso em: 29 ago. 2022.

MAHFOODH, Hajar ; ALATAWI, Hessa. Higher Education in COVID-19: From Emergency to Sustainable Remote Education. **Sustainable Leadership and Academic Excellence International Conference (SLAE)**, Manama, 09 Novembro 2021.

MARAšEVIć, Jovana ; GAVROVSKA, Ana. Virtual Reality and WebRTC implementation for Web educational application development. **28th Telecommunications forum TELFOR 2020**, Belgrade, 24 Novembro 2020.

MARQUES, Eduardo. Análise mostra que webcams de laptops pararam no tempo. **MacMagazine**, 2020. Disponivel em: https://macmagazine.com.br/post/2020/04/13/analise-mostra-que-webcams-de-laptops-pararam-no-tempo/. Acesso em: 16 fev. 2022.

MARTINEZ, Bruno. Os 5 melhores benchmarks para testar a performance do seu PC. **Show me tech**, 2022. Disponivel em: https://www.showmetech.com.br/top-5-benchmark-medir-performance-pc/. Acesso em: 21 nov. 2022.

OBS. OBS-VirtualCam. **OBS Open Broadcaster Software**, 2021. Disponivel em: https://obsproject.com/forum/resources/obs-virtualcam.539/. Acesso em: 12 set. 2022.

ÖZTÜRK, Savaş *et al.* Functionality, Performance and Usability Tests of WebRTC Based Video Conferencing Products. **2021 15th Turkish National Software Engineering Symposium (UYMS)**, Izmir, 17 Novembro 2021.

PAN, Wubin ; CHENG, Guang. QoE Assessment of Encrypted YouTube Adaptive Streaming for Energy Saving in Smart Cities. **IEEE Access**, 9 Março 2018. 25142-25156.

PASTUSHOK, Igor ; TURLIKOV, Andrey. Lower bound and optimal scheduling for mean user rebuffering percentage of HTTP progressive download traffic in cellular networks. **2016 XV International Symposium Problems of Redundancy in Information and Control Systems (REDUNDANCY)**, St. Petersburg, 26 Setembro 2016.

PEREIRA, Ella ; PEREIRA, Rubem. Dynamic Adaptive Streaming over HTTP and Progressive Download: Comparative Considerations. **2014 28th International Conference on Advanced Information Networking and Applications Workshops**, Victoria, 13 Maio 2014. 905-909.

RIBEIRO, Denise ; WELLS, Anthony. Com pandemia, demanda por videoconferências dispara em empresas brasileiras. **CNN Brasil**, 2021. Disponivel em: https://www.cnnbrasil.com.br/business/com-pandemia-demanda-por-videoconferencias-dispara-em-empresas-brasileiras/./. Acesso em: 4 jul. 2022.

ROSENCRANCE, Linda ; MOOZAKIS, Chuck ; LATON, George. User Datagram Protocol (UDP). **TechTarget**, 2021. Disponivel em: https://www.techtarget.com/searchnetworking/definition/UDP-User-Datagram-Protocol. Acesso em: 28 set. 2022.

SCHULZRINNE, Henning ; RAO, Anup ; LANPHIER, Rob. Real-Time Streaming Protocol Version 2.0. **RFC 7826**, 2016. Disponivel em: https://www.rfc-editor.org/rfc/rfc7826.html#section-13.2. Acesso em: 13 jul. 2022.

SOUZA, Gustavo Henrique Silva *et al.* Educação Remota Emergencial (ERE): Um estudo empírico sobre Capacidades. **Research, Society and Development**, 18 Janeiro 2021.

TEMASYS. Understanding ICE in WebRTC. **Temasys**, 2022. Disponivel em: https://temasys.io/guides/developers/webrtc-ice-sorcery/. Acesso em: 11 set. 2022.

TRUECONF. O que é WebRTC? **TrueConf**, 2022. Disponivel em: https://trueconf.com/br/webrtc.html#:~:text=Como%20Funciona%20o%20WebRTC,-Vamos%20ver%20como&text=Um%20navegador%20pode%20solicitar%20acesso,a%20uma%20transmissão%20ao%20vivo. Acesso em: 11 Setembro 2022.

TUTORIALSPOINT. SDP - Session Description Protocol. **TutorialsPoint**, 2022. Disponivel em: https://www.tutorialspoint.com/session_initiation_protocol/session_initiation_protocol_sdp.htm#. Acesso em: 21 nov. 2022.

U, Pavan Kumar; CH, Venugopal Reddy; K, Suresh Babu. A REVIEW ON VIDEO PROCESSING. **International Journal of Advanced Technology in Engineering and Science**, 6 Junho 2021.

VOLPATO, Bruno. Benchmarking: o que é, como fazer, dicas e material gratuito! **Resultados Digitais**, 2020. Disponivel em: https://resultadosdigitais.com.br/marketing/benchmarking/. Acesso em: 21 nov. 2022.

VYAS, Kishan. How to use your Android phone as a webcam for your PC or laptop. **xda**, 2021. Disponivel em: https://www.xda-developers.com/how-to-use-android-phone-as-webcam-for-pc/. Acesso em: 12 nov. 2021.

WHITE, Steven *et al.* DirectShow. **Microsoft**, 2022. Disponivel em: learn.microsoft.com/pt-br/windows/win32/directshow. Acesso em: 18 set. 2022.

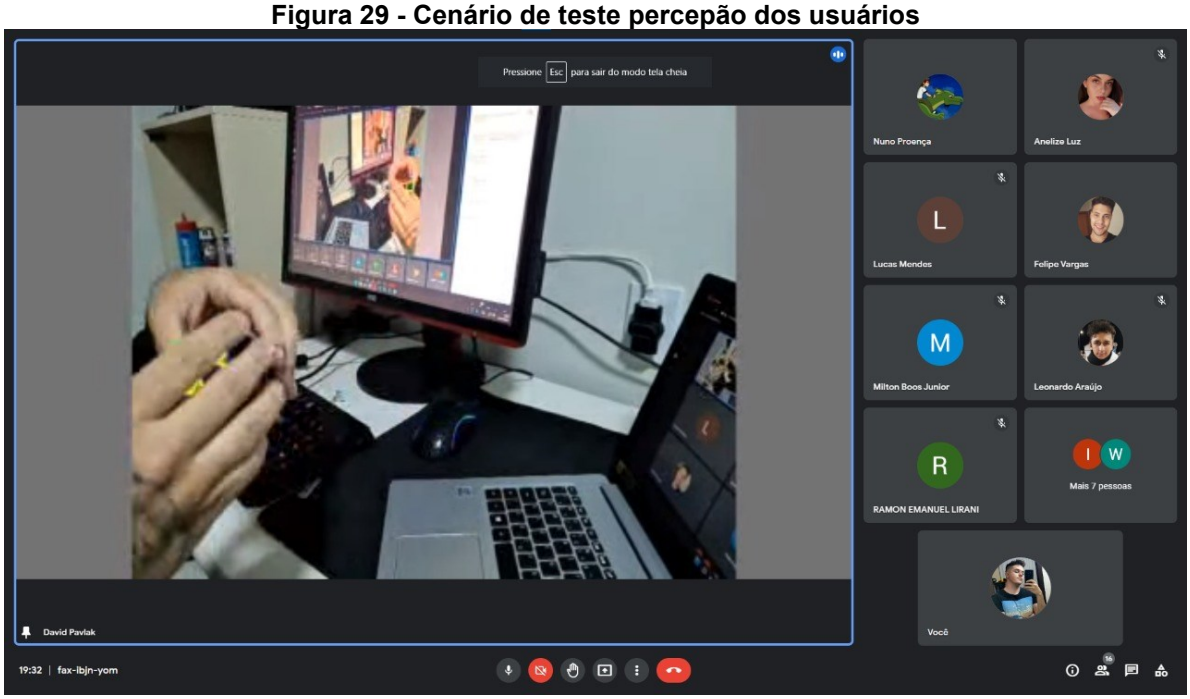
ZHANG, Meng *et al.* Understanding the Power of Pull-Based Streaming Protocol: Can We Do Better? **IEEE Journal on Selected Areas in Communications** , Dezembro 2007. 1678-1694.

APÊNDICE A - Cenários de Teste

Figura 28 - Cenário de teste Benchmark

Fonte: Autoria própria (2022)

Figura 29 - Cenário de teste percepão dos usuários



Fonte: Autoria própria (2022)

Figura 30 - Cenário de teste qualidade às cegas

Fonte: Autoria própria (2022)