Gaming On The Edge: Uma arquitetura de computação na borda para jogos em dispositivos móveis

Gabriel Pimenta Robaina[1], **Adriano Fiorese**[1]

[1]Programa de Pós-Graduação em Computação Aplicada (PPGCAP)
Universidade do Estado de Santa Catarina (UDESC)
Joinville, Santa Catarina, Brasil 89219-710

gabriel.robaina@edu.udesc.br, adriano.fiorese@udesc.br

Abstract. *The standard software architecture for mobile games requires local processing of game logic and graphics. On the other hand, cloud-based remote gaming architectures for mobile devices present high latency for game commands at a high cost for service providers. This work proposes the Gaming On The Edge (GOTE) architecture, which aims to allow complex games to be played on mobile devices by leveraging edge computing infrastructure for graphics processing and multimedia content distribution. Experimental results show that GOTE architecture is a viable alternative to cloud based remote gaming on mobile devices at the advantage of lowering latency of video and game input. An open source implementation of the architecture is provided in order to assist further research in this area.*

Resumo. *A arquitetura de software padrão para jogos para dispositivos móveis requer o processamento local da lógica do jogo e gráficos. Por sua vez, as arquiteturas de remote gaming em nuvem para dispositivos móveis apresentam alta latência para os comandos de jogo e um alto custo para o provedor do serviço. Esse trabalho propõe a arquitetura Gaming On The Edge (GOTE), que visa permitir que jogos complexos sejam jogados em dispositivos móveis usando computação na borda da rede para processamento de gráficos e distribuição de conteúdo multimídia. Resultados experimentais da avaliação da prova de conceito dessa arquitetura indicam ser esta uma alternativa viável para remote gaming baseado em nuvem para jogos em dispositivos móveis, com menor latência de vídeo e de comandos de jogo. Além disso, uma implementação da arquitetura foi disponibilizada com código aberto para auxiliar futuras pesquisas na área.*

1. Introdução

O mercado de jogos para dispositivos móveis tornou-se o maior quando comparado a plataformas tradicionais como PC e consoles. Em 2021, jogos para *tablets* e *smartphones* somaram uma receita de 96 bilhões de dólares, 52% da fatia de mercado, e uma tendência de crescimento para os anos seguintes [Newzoo 2021]. A arquitetura padrão para esses jogos requer o processamento local da lógica de jogo e gráficos usando os recursos de *hardware* do próprio dispositivo. Isso apresenta uma limitação para a variedade e complexidade dos jogos que visam essas plataformas, uma vez que os dispositivos móveis tem menos capacidade de *hardware* quando comparados a PCs e consoles.

Uma possível solução para esse problema se encontra na utilização da infraestrutura de nuvem para processamento remoto da lógica de jogo e gráficos. Empresas como

a Sony, NVIDIA e Paperspace tem fornecido serviços de *cloud gaming* [Lin et al. 2019]. Na arquitetura de *cloud gaming* as interações do jogador são enviadas para um servidor de nuvem, e uma cena já renderizada é devolvida em uma transmissão de vídeo [Messaoudi et al. 2017]. Apesar de escalável, essa abordagem de jogabilidade remota requer que o código de jogo seja distribuído em múltiplos servidores de nuvem, fazendo com que a arquitetura seja suscetível a uma alta latência nos comandos de jogo em más condições de conexão, levando a uma baixa Qualidade de Experiência (QoE - *Quality of Experience*). Além disso, esse modelo implica em altos custos de infraestrutura para o provedor de *cloud gaming* uma vez que a maior parte do custo computacional está localizado na nuvem [Cai et al. 2016].

Computação de borda (EC - *edge computing*) é um paradigma que leva o processamento de dados para as bordas da rede ao invés de na nuvem computacional centralizada. Componentes de infraestrutura dedicados à borda, como *cloudlets* [Lin et al. 2019], ou dispositivos como roteadores e *smartphones* podem trocar cargas de trabalho com comunicação de baixa latência dentro da rede local. Ainda, esses dispositivos podem delegar o pós-processamento de dados para a nuvem se necessário [Liu et al. 2019]. Esses nós de borda podem fazer uso de virtualização para hospedar a execução de código enviado a partir de aplicações externas por meio de *code offloading* [Zhang et al. 2019]. Essa estratégia torna possível a disponibilização de aplicações sofisticadas, como jogos originalmente desenvolvidos para PC ou consoles, em dispositivos móveis. Além disso, o *code offloading* estende a duração da bateria uma vez que a maior parte da carga computacional seria transferida do dispositivo móvel do usuário para um nó da borda da rede. Dessa forma, esse trabalho se dedica a encontrar uma oportunidade para que usuários experimentem jogos sofisticados em dispositivos móveis em uma estratégia de jogos remotos utilizando recursos computacionais oferecidos na borda da rede.

Para isso, este trabalho propõe uma arquitetura de EC para jogos remotos em dispositivos móveis denominada Gaming On The Edge (GOTE), aproveitando a proximidade entre nós de borda e dispositivos móveis para obter uma baixa latência de comunicação. Semelhante à arquitetura de jogos em nuvem, essa alternativa evita que a lógica e a renderização do jogo sejam realizadas pelo dispositivo móvel do jogador, delegando a execução do código relacionado para um nó de borda próximo, permitindo que jogos sofisticados sejam jogados em *smartphones* e aumentando a QoE do usuário.

Nesse sentido, este trabalho fornece as seguintes contribuições: 1) uma arquitetura de jogos remotos baseada em borda que visa permitir que jogos intensivos em recursos sejam jogados em dispositivos móveis; 2) um *pipeline* de *streaming* que atinge um feedback de vídeo de jogos com baixa latência no contexto de EC sem instrumentação do código do jogo; 3) uma implementação de prova de conceito com código aberto[1] dos principais componentes da arquitetura para auxiliar pesquisas futuras nesta área.

Este trabalho está organizado da seguinte forma. A Seção 2 denota conceitos básicos para a compreensão do trabalho proposto. A Seção 3 apresenta e discute trabalhos anteriores que levaram a abordagem de jogos remotos para o contexto de EC. A Seção 4 define a arquitetura GOTE. Além disso, a Seção 5 descreve os experimentos realizados com essa arquitetura e seus resultados, que são discutidos na Seção 6. A Seção 7 conclui

[1]https://github.com/gpr-indevelopment/gote-game-server-2

este artigo e propõe trabalhos futuros.

2. Fundamentação técnica

Esta seção apresenta conceitos e ferramentas utilizadas na arquitetura e implementação do sistema. O padrão *Web Real-Time Communication* (WebRTC) é usado na arquitetura GOTE para estabelecer a comunicação entre a aplicação cliente do jogador e o servidor de renderização, enquanto o componente GStreamer é responsável pelo *pipeline* de mídia que permite o *streaming* em tempo real das cenas do jogo. Além disso, o *pipeline* construído aproveita o codificador de vídeo Nvidia NVENC, que utiliza aceleração por *hardware* para obter *feedback* de vídeo com baixa latência.

2.1. WebRTC

WebRTC é um padrão que fornece interfaces de programação de aplicativos (APIs - *Application Programming Interface*) que permitem comunicação de ponto a ponto entre pares (P2P - *Peer to Peer*) em tempo real com navegadores HTML5 [Hardie and Turner 2021], e é comumente usado para *webconferências*. Ele também permite que transmissões com protocolo de transporte em tempo real (RTP - *Real-time Transport Protocol*) sejam exibidas em *tags* de vídeo em páginas HTML5 [Loreto and Romano 2014]. O RTP aproveita o *User Datagram Protocol* (UDP) em vez do *Transmission Control Protocol* (TCP) na camada de transporte para obter comunicação de baixa latência e alta taxa de transferência de dados. Para que os pares se conectem, primeiro eles devem passar pelo processo de sinalização, no qual cada par compartilha informações sobre os tipos de mídia suportados, *codecs* e configurações relacionadas por meio do Protocolo de Descrição de Sessão (SDP - *Session Description Protocol*). Além disso, as informações de acessibilidade de cada par, como endereços IP públicos, são coletadas de um servidor de *Session Traversal Utilities for Network Address Translation* (STUN) e compartilhadas por meio da técnica de *Interactive Connectivity Establishment* (ICE) [Rosenberg 2010]. Tradicionalmente, toda a comunicação e troca de informações neste processo é mediada por um servidor de sinalização dedicado [Loreto and Romano 2014]. No GOTE, o servidor de *streaming* também fornece funcionalidades de sinalização e atua como mediador desse processo. O WebRTC é vantajoso para a arquitetura GOTE, pois fornece um padrão para estabelecer uma sessão de *streaming* entre o servidor de renderização e a aplicação cliente do *smartphone*. Ele também permite que a transmissão do jogo seja facilmente exibida em *smartphones*, aproveitando a API WebRTC disponível em navegadores modernos.

2.2. STUN

As diferentes topologias de rede dos pares que se comunicam via WebRTC podem aumentar a complexidade do processo de estabelecimento da conexão. Por exemplo, os pares podem estar em diferentes redes privadas, usando o protocolo *Network Address Translation* (NAT) para serem acessados pela internet por meio de endereços IP públicos. O GOTE aproveita o protocolo STUN para permitir a conexão entre o servidor de renderização e o cliente do jogador em uma ampla variedade de topologias de rede.

STUN é um protocolo usado no processo de sinalização do WebRTC para coletar informações e endereços IP públicos dos pares envolvidos na conexão P2P [Loreto and Romano 2014, Rosenberg 2010]. Os pares podem solicitar seus endereços IP públicos por meio de um servidor STUN centralizado em nuvem, criando uma ligação

NAT nos roteadores. Essa associação mapeia um IP público e uma porta para um IP na rede privada, permitindo que os pares sejam acessados pela Internet. O STUN também pode ser usado para manter ligações NAT por meio de verificações periódicas de conectividade [Rosenberg 2008]. Por fim, os pares podem trocar seus endereços IP públicos por meio do ICE como parte do processo de sinalização do WebRTC.

2.3. GStreamer

GStreamer é um *framework* para aplicativos de *streaming* de mídia. Ele permite que pipelines multimídia sejam construídos com uma ampla variedade de fontes e formatos de entrada e saída. Um *pipeline* do GStreamer consiste em elementos que são interconectados para levar dados multimídia de uma fonte para uma saída [GStreamer 2021]. A codificação de vídeo deve ser executada por um elemento do *pipeline* de mídia de acordo com os formatos de mídia permitidos pelo nó de destino no contrato de sessão. No caso da arquitetura GOTE, o *streaming* de vídeo RTP atua como a saída do *pipeline* de mídia.

3. Trabalhos relacionados

O projeto Games@Large [Nave et al. 2008] teve como objetivo pesquisar, desenvolver e implementar uma arquitetura para execução remota de jogos utilizando *code offloading* para servidores locais. Os casos de uso dessa arquitetura incluem hotéis, navios de cruzeiro e cibercafés. Em vez de transmitir as cenas do jogo como vídeo de volta para o dispositivo do jogador, essa abordagem pretendia que as cenas fossem renderizadas localmente usando os recursos de hardware do dispositivo móvel [Eisert and Fechteler 2007]. Isso foi possível capturando os comandos enviados pela lógica do jogo para a API gráfica relacionada e redirecionando-os para o dispositivo móvel do jogador para renderização. Os testes mostraram baixas taxas de quadros para dispositivos móveis, variando de 7 *frames* por segundo (FPS) em um jogo de estratégia para 18 FPS em um jogo casual com uma média de aproximadamente 0,34 Mb/s enviados pela rede local. Um dos jogos testados não pôde ser executado no dispositivo móvel, pois não possuía os recursos de aceleração de hardware requeridos para a renderização do jogo.

O projeto EdgeGame [Zhang et al. 2019] propôs uma arquitetura baseada em EC para jogos móveis e construiu um protótipo que transfere o processamento da lógica do jogo e a renderização para nós de borda usando virtualização. As cenas do jogo são enviadas de volta como um *stream* de vídeo para o dispositivo móvel usando o padrão WebRTC. Um algoritmo de controle de congestionamento é usado para ajustar dinamicamente a taxa na qual os dados são transferidos com base nas condições da rede [Jansen et al. 2018]. Esse padrão se encaixa no caso de uso de jogos móveis, pois fornece adaptabilidade em redes instáveis e comunicação em tempo real da entrada do jogador e do *streaming* de vídeo do jogo. No EdgeGame, o jogador pode localizar os nós de borda disponíveis enviando solicitações para um servidor centralizado em nuvem, que também é responsável por gerenciar contas de usuários e fornecer serviços de autenticação. Os atrasos de rede experimentados na abordagem de EC foram significativamente menores (16,2ms) quando comparados a uma estratégia baseada em nuvem (44,2ms). Além disso, a QoE do usuário no EdgeGame foi 20% maior quando comparada a uma alternativa baseada em nuvem.

O projeto RenderLink [Oros and Bâcu 2020] também adota a abordagem de *code offloading* para a borda enquanto envia um *stream* de vídeo de volta para o cliente usando

WebRTC. Em vez de usar um nó de borda dedicado para renderização, o RenderLink propõe uma estratégia P2P que aproveita os dispositivos ociosos dos usuários para essa tarefa. Em uma implementação comercial, os usuários que expõem seus recursos de *hardware* e cooperam com rede podem ser recompensados com moedas virtuais. Ainda assim, nessa abordagem a complexidade dos jogos renderizados é limitada pela disponibilidade dos recursos de hardware na rede. Os testes realizados com o projeto RenderLink mostraram uma taxa de quadros média de 55,65 quadros por segundo (FPS) em 720p em uma conexão com fio e desvio padrão de 13,48. Observou-se que a maioria das implementações de WebRTC começa a transmitir com baixa qualidade, aumentando gradualmente até uma condição estável para acomodar restrições de largura de banda em cerca de 1 minuto e 20 segundos. Essa característica das implementações WebRTC pode prejudicar a QoE nesse período de tempo.

3.1. Considerações sobre os trabalhos relacionados

Este trabalho usa a abordagem WebRTC para transmitir cenas de jogos de um servidor de renderização na borda da rede para o dispositivo móvel do jogador por meio do uso de RTP, conforme sugerido por EdgeGame e RenderLink, em um *pipeline* de mídia que aproveita a codificação acelerada por *hardware* para melhorar o desempenho e o ganho de QoE. Além disso, a virtualização é usada como uma plataforma para instanciar e gerenciar servidores de renderização usando um componente orquestrador. Nenhum dos trabalhos relacionados forneceu detalhes de implementação suficientes para reprodutibilidade. A Tabela 1 apresenta uma comparação entre a proposta GOTE e a literatura analisada que levou jogos remotos para o contexto de EC.

Tabela 1. Comparativo entre GOTE e trabalhos relacionados

	Games@Large	RenderLink	EdgeGame	GOTE
Renderização	Dispositivo do jogador	Servidor de borda	Servidor de borda	Servidor de borda
Multijogador	X	✓	✓	X
WebRTC	X	✓	✓	✓
RTP	✓	✓	✓	✓
Codificação acelerada por *hardware*	X	Desconhecido	Desconhecido	✓
Sem instrumentação do código do jogo	X	X	Desconhecido	✓
Forneceu implementação	X	X	X	✓

4. Visão geral da arquitetura

A arquitetura GOTE é baseada na comunicação direta entre um servidor de renderização e um cliente de dispositivo móvel do jogador. Em seguida, os comandos de jogo são envi-

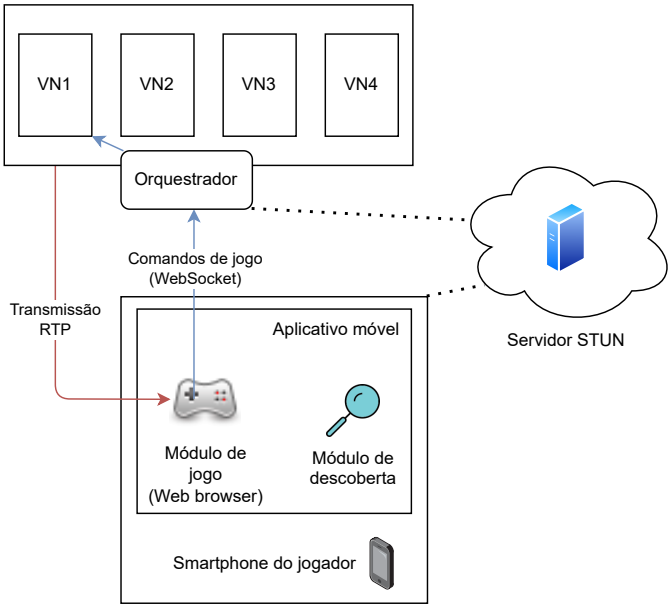

Figura 1. Visão geral da arquitetura GOTE

ados do dispositivo do jogador para o servidor de renderização por meio de WebSockets. Por sua vez, o servidor de renderização renderiza a cena do jogo e transmite o vídeo de volta ao cliente usando RTP. Esse servidor é implantado em um nó virtual (VN) de um PC *desktop* ou outro dispositivo de borda com recursos de processamento gráfico capazes de iniciar o jogo solicitado e um *pipeline* de mídia para o fluxo RTP. Esse *pipeline* deve ser eficaz para transmitir com FPS e resolução que maximizem a QoE do jogador. O dispositivo móvel deve possuir um aplicativo com um módulo de jogo, responsável por exibir o *streaming* de vídeo do jogo e enviar os comandos de jogo, e um módulo de descoberta, que permite a descoberta de um orquestrador acessível pelo cliente. O componente orquestrador é responsável por instanciar e gerenciar VNs enquanto encaminha os comandos de jogo do módulo do jogo para processamento no servidor de renderização correspondente. Esse processamento consiste em interpretar os comandos remotos para encaminhamento ao processo do jogo como comandos locais. A Fig. 1 apresenta uma visão geral da arquitetura. Nesta solução, cada VN é responsável por uma sessão de jogo para exibição do *stream* RTP no navegador web do usuário em conformidade com o padrão WebRTC.

Conforme descrito na Seção 2.1, o uso do WebRTC requer um servidor de sinalização que atue como um mediador para estabelecer a conexão entre os pares. Embora o servidor de renderização GOTE atue como um mediador durante o processo de sinalização, um servidor STUN em nuvem ainda é necessário para adquirir informações de acessibilidade dos pares em conexões pela Internet. Nesse caso, essa arquitetura depende de servidores STUN em nuvem que sejam acessíveis tanto pelo cliente quanto pelo servidor de renderização via Internet. A comunicação com os servidores em nuvem ocorre apenas durante a sinalização e não é impactante na QoE do jogo. Ainda, a comunicação com o servidor de STUN não é necessária quando os pares compartilham a mesma rede local.

Os VNs são fornecidos pelo componente orquestrador quando uma solicitação de sessão de jogo é recebida vindo do dispositivo cliente do jogador, conforme mostrado na

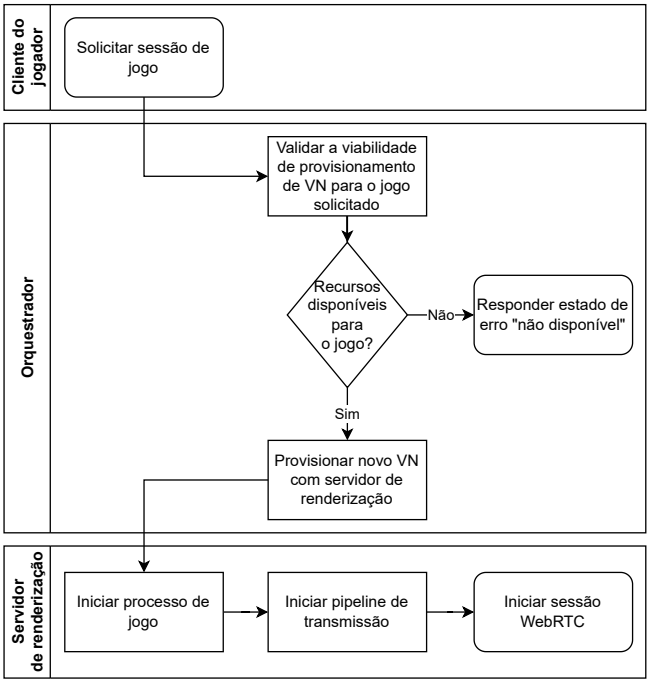

Figura 2. Fluxograma do processo de provisionamento do VN e do servidor de renderização

Fig. 2. Os VNs são responsáveis por executar os processos do jogo e transmitir dados de vídeo para os clientes enquanto compartilham recursos de *hardware* do hospedeiro. Assim, quando um cliente solicita uma sessão de jogo ao orquestrador, este verifica se há recursos suficientes para instanciar um novo VN para atender aquele jogador. Caso seja possível, uma aplicação de servidor de renderização é implantada em um VN com Windows instanciado na plataforma HyperV. Essa aplicação é responsável por iniciar os processos referentes a jogos e *streaming* de vídeo. O *pipeline* de mídia é implementado usando o GStreamer enquanto aproveita a codificação acelerada por *hardware* NVENC das GPUs NVIDIA acessíveis a VM por meio de passagem de GPU.

A Fig. 3 apresenta o *pipeline* GStreamer montado para o sistema. O elemento *dxgiscreencapsrc* é responsável por capturar os dados RGBA (vermelho, verde, azul e alfa) da tela do jogo a uma taxa de 60 FPS. O elemento seguinte, *nvh264enc*, usa a API do NVENC para codificar o *stream* de vídeo com a compactação H.264. Em seguida, o *rtph264pay* empacota o *stream* de vídeo codificado em H.264 como carga útil dos pacotes RTP. Por fim, o GStreamer disponibiliza essa transmissão para conexões WebRTC em *webrtcbin*. WebSockets são usados para enviar comandos de entrada do jogo para os VNs, e também para toda a comunicação relacionada à sinalização devido às suas capacidades bidirecionais.

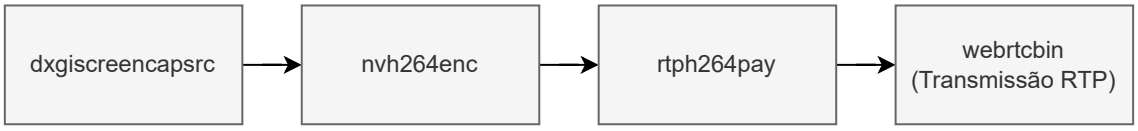

Figura 3. *Pipeline* GStreamer

Um exemplo prático da arquitetura GOTE pode ser dividido em duas fases. Primeiramente, na fase de provisionamento de jogos e *streaming*, o usuário jogador inicia sua interação com a arquitetura por meio de um cliente de aplicativo móvel. Este aplicativo é responsável por descobrir e se comunicar com um orquestrador local para estabelecer uma sessão de jogo. A descoberta do orquestrador é realizada pelo módulo de descoberta do aplicativo móvel por meio de um servidor centralizado em nuvem, ou um protocolo de descoberta de serviço como SSDP ou SLP. Em seguida, o jogador pode solicitar um jogo ao orquestrador a partir de um catálogo de jogos instalados no VN e disponíveis para execução remota. Após receber uma solicitação de jogo, o orquestrador realiza o provisionamento do VN para um hospedeiro, conforme apresentado na Fig. 2. O orquestrador retorna um erro se não houverem recursos de hardware suficientes para o jogo solicitado. Se o provisionamento for bem-sucedido, o módulo de jogo do aplicativo móvel abre o navegador da web para iniciar o estabelecimento da sessão WebRTC, disponibilizando o *streaming* de vídeo do jogo e permitindo que o usuário comece a jogar. Isso estabelece o início da fase de jogo remoto. Neste ponto, todas as entradas de comandos de jogo, juntamente com um identificador de sessão, serão enviadas ao orquestrador para encaminhamento ao VN responsável por hospedar a sessão de jogo atual. A Fig. 4 apresenta as interações entre os componentes em um exemplo com provisionamento de VN bem-sucedido.

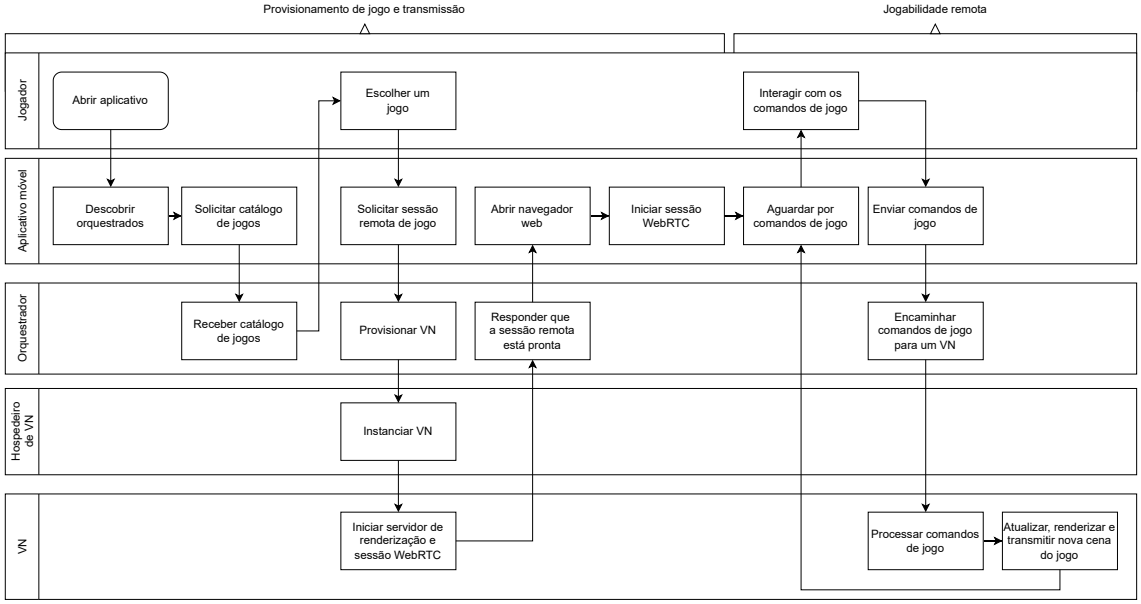

Figura 4. Diagrama de interação entre componentes da arquitetura GOTE

5. Experimentos e resultados

Experimentos foram conduzidos para avaliar a arquitetura GOTE em cenários de complexidade crescente. O cenário "Local" consiste em executar o cliente do jogador e o servidor de renderização localmente no mesmo *hardware* físico para estabelecer uma base de desempenho para a arquitetura. O cenário "LAN sem fio" consiste em executar o cliente do jogador em um *smartphone* e o servidor de renderização em um PC *desktop* que compartilham a mesma LAN sem fio em 5 GHz, emulando um ambiente de EC de alto

desempenho. O último experimento, rotulado como “4G”, consiste em executar a arquitetura durante o *streaming* de cenas de um servidor de renderização de PC *desktop* para um dispositivo cliente conectado ao 4G, emulando um cenário de EC mais realista.

As métricas de desempenho de *streaming* de vídeo escolhidas para avaliar a arquitetura GOTE são jitter, perda de pacotes, taxa de *bits* e quadros perdidos e enviados por segundo para o dispositivo cliente do jogador. Além disso, o atraso de entrada de comandos de jogo (GID - *Game Input Delay*) foi medido como a diferença de tempo entre a interação do jogador e a chegada do comando no servidor de renderização a cada segundo. Jitter, perda de pacotes e perdas de quadros de vídeo são métricas diretamente relacionadas à estabilidade da conexão e transmissão de dados entre o servidor de renderização e o cliente do jogador. Além disso, a quantidade de dados de vídeo transmitidos pela rede e sua variação ao longo do tempo são representados pelos dados de taxa de *bits* (*bitrate*), enquanto os dados de GID foram usados para avaliar como diferentes cenários de rede impactam a resposta em tempo real do jogo.

Um PC com Windows 10 e processador i5-9400F 2,90 GHz, 16 GB de RAM e placa de vídeo NVIDIA GeForce RTX 2060 hospedou o servidor de renderização em todos os experimentos, juntamente com um *smartphone* Redmi M2101K7AI com 6 GB de RAM como dispositivo cliente do jogador. O navegador web escolhido foi o Google Chrome 96.0.4664.45 executado no Android 11. Um servidor STUN de nuvem pública foi usado durante a sinalização para estabelecer a conexão entre os pares. O componente do codificador NVENC H.264 do *pipeline* GStreamer foi definido para uma taxa de *bits* constante de 500 KBps com a predefinição de baixa latência, conforme recomendado pela NVIDIA para casos de uso de *streaming* de jogos [NVIDIA 2022]. O componente de captura de tela usou uma resolução de origem de 1280x720 *pixels* (720p) a 60 quadros por segundo. Os testes gráficos 1 e 2 do benchmark Time Spy do *software* de *benchmarking* 3DMark foram transmitidos do servidor de renderização em todos os experimentos. Os *benchmarks* duraram 150 segundos no total, com uma tela de carregamento no início e entre os testes. O *benchmark* 3DMark foi selecionado para os experimentos, em oposição a um jogo real, por favorecer a reprodutibilidade dos cenários propostos.

Mesmo que os cenários reais de jogos remotos envolvam *streaming* de vídeo e comunicação de comandos de jogos simultaneamente, é improvável que o tráfego envolvido no envio de comandos leves de entrada do usuário para o servidor de renderização influencie profundamente os resultados da transmissão. Portanto, os dados de GID foram coletados em experimentos independentes para todos os três cenários de rede utilizando relógios de Lamport para sincronização das tomadas de tempo. Esses experimentos foram executados em uma janela de tempo de 90 segundos, o que é suficiente para capturar o impacto dos diferentes cenários de rede na resposta de entrada do jogo e na experiência de jogo remoto. A Fig. 5 e a Tabela 2 apresentam os resultados de todos os experimentos.

Essas métricas foram coletadas para uma transmissão do servidor de renderização para um cliente de jogador e, portanto, o componente de orquestração não foi usado durante a experimentação. Além disso, o tempo para o estabelecimento da conexão relacionado ao fornecimento e sinalização de VN não foi levado em consideração durante os experimentos.

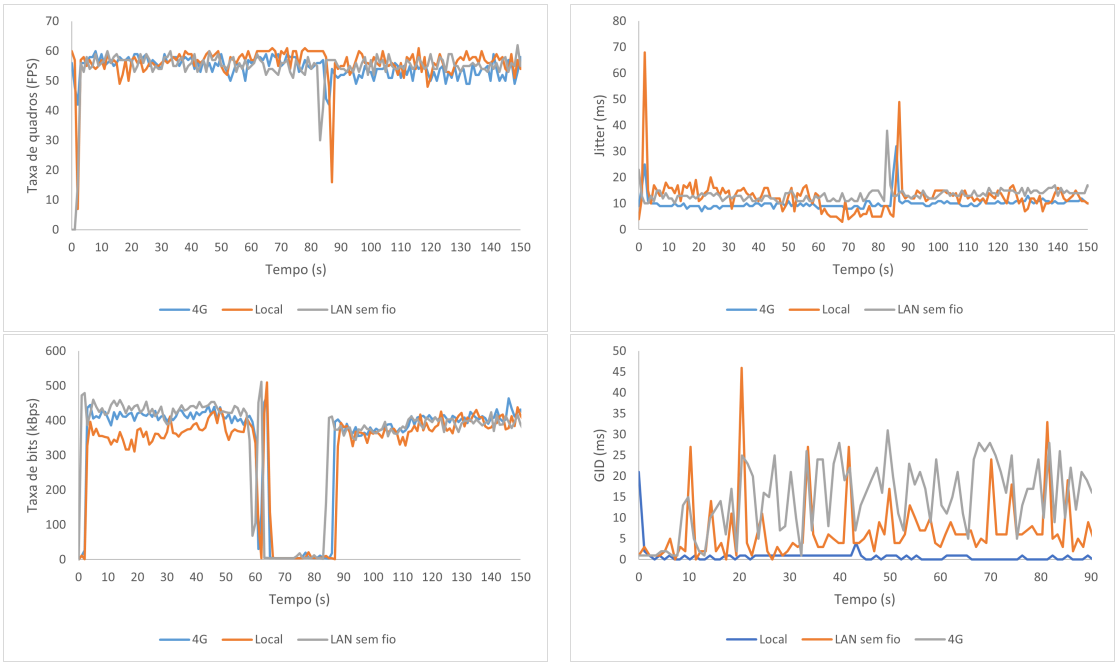

Figura 5. Resultados de FPS, Jitter, taxa de bits transmitidos e GID

6. Discussão

A taxa de processamento de quadros de vídeo (*frame rate*) é um recurso do vídeo particularmente importante para a experiência do jogador. Embora 60 FPS sejam desejáveis, sabe-se que alguma variação entre 30 e 60 FPS não afeta significativamente a QoE dos jogos [Zadtootaghaj et al. 2018]. Todos os experimentos mostraram taxas de processamento de quadros de vídeo estáveis em torno de 60 FPS. No entanto, os cenários Local e LAN sem fio mostraram o maior desvio da marca média de 5,68 e 7,61 FPS, respectivamente. Parte significativa desses desvios aconteceu por conta da diminuição do FPS no início do *streaming* e perto da marca dos 90 segundos, durante as transições entre as telas de carregamento e o início dos *benchmarks*. Há também picos de *jitter* em todos os experimentos nas mesmas janelas de tempo. O H.264

Tabela 2. Resultados de média e desvio padrão para as métricas de desempenho

Métrica (Unidade)	Média (Desvio padrão)		
	Local	LAN sem fio	4G
Taxa de quadros (FPS)	56,17 (5,68)	54,47 (7,61)	54,43 (3,17)
Jitter (ms)	12,45 (6,41)	13,33 (2,52)	10,04 (2,53)
Taxa de *bits* (kbps)	318,54 (138,64)	344,56 (146,57)	343,29 (148,27)
GID (ms)	0,80 (2,23)	6,85 (7,63)	14,78 (8,50)

usa compensação de movimento, que é uma técnica de previsão de quadros futuros com base no movimento da câmera ou objetos em quadros vizinhos do vídeo, melhorando assim a taxa de compressão e o tempo de codificação do fluxo de *bits* resultante [Chen et al. 2001, Ludwich and Fröhlich 2011, Union 2021]. Por conta disso, a transição das telas de carregamento para os *benchmarks* pode ter gerado um atraso na etapa de compressão do *pipeline*, o que impactou a taxa de processamento de quadros de vídeo tanto no início quanto na marca dos 90 segundos. Em um cenário de jogo real, esse atraso na compressão é mais provável de ocorrer em jogos cinematográficos, onde os cortes de câmera são frequentes, e menos provável de ocorrer em jogos de estratégia, onde a compensação de movimento pode tirar proveito de imagens que mudam pouco de um quadro para o outro. O impacto da compensação de movimento pode ser visto nos dados de taxa de *bits* entre as marcas de 60 e 90 segundos, durante a transmissão de uma tela de carregamento onde a única região de movimento no vídeo é a barra de carregamento. Nesse caso a taxa de *bits* na transmissão cai próxima a zero.

O experimento 4G teve 4 pacotes perdidos durante o experimento. Isso é crítico para o fluxo RTP, pois o UDP não possui mecanismo nativo de recuperação para lidar com pacotes perdidos. Além disso, essa métrica tem um impacto significativo na qualidade percebida do *streaming* de vídeo [Pande et al. 2013]. Ainda assim, os 4 pacotes perdidos na marca de 140 segundos não tiveram impacto na estabilidade do vídeo, conforme visto nas taxas de processamento de quadros de vídeo e de *bits*.

O experimento Local teve 6 quadros perdidos durante a janela de tempo de 150 segundos. Executar localmente o servidor de renderização e o cliente do navegador pode ter impactado o desempenho devido à competição por recursos de hardware, favorecendo consequentemente a perda de pacotes. Ainda assim, nenhum cenário foi significativamente impactado pela perda de quadros ao longo dos experimentos.

Medições de jitter abaixo de 100 ms não prejudicam a QoE do jogador, mesmo para jogos dinâmicos como os de tiro de ação multijogador [Amin et al. 2013]. Essa métrica permaneceu abaixo de 100 ms em todos os experimentos, mesmo nas transições das telas de carregamento, o que significa que o *buffer* usado pelo WebRTC foi eficiente no sequenciamento de pacotes para o fluxo de vídeo e nenhum componente no *pipeline* de mídia criou atrasos significativos na entrega de quadros.

A taxa média de *bits* para todos os experimentos permaneceu entre 310 e 350 KBps, o que significa que, em média, todos os cenários suportaram uma taxa semelhante de transmissão de dados de vídeo para o cliente do jogador. Essa taxa é inferior aos 500 KBps especificados no codificador de vídeo do *pipeline* GStreamer devido às condições de largura de banda e compensação de movimento. Além disso, os experimentos de LAN sem fio e 4G tiveram maiores variações de taxa de *bits* quando comparados ao experimento Local devido à natureza remota da transmissão. É perceptível que a variabilidade da taxa de *bits* aumentou conforme a complexidade dos experimentos, do Local ao 4G.

O GID dos experimentos ficou abaixo de 20 ms e aumentou conforme a complexidade do cenário. Este resultado favorece a QoE mesmo quando comparado a 60 ms, considerado pequeno até para jogos online de ação [Quax et al. 2004, Amin et al. 2013]. Além disso, as métricas de GID foram coletadas a cada segundo para uma entrada simples de comandos de jogo, o que significa que esses resultados podem variar para jogos

complexos que exigem maior taxa de interação do jogador.

A arquitetura GOTE permite o *streaming* de jogos em um servidor de renderização sem a necessidade de qualquer instrumentação do código do jogo, como feito em outros trabalhos relacionados [Oros and Bâcu 2020]. A não necessidade de instrumentação nesse caso é uma vantagem pois permite que qualquer jogo já existente compatível com Windows possa ser executado na borda de acordo com a arquitetura proposta. A necessidade de instrumentação faria com que as desenvolvedoras de jogos tivessem que disponibilizar versões de seus produtos com compatibilidade específica com o edge gaming. Além disso, a escala de uma aplicação comercial baseada em GOTE pode ser aumentada com um sistema de recompensas em moedas virtuais em troca de recursos de *hardware* doados, habilitando formas flexíveis de monetização.

O projeto RenderLink relatou uma média de 60 FPS em uma resolução de 1600x900 [Oros and Bâcu 2020]. Além disso, o Games@Large mostrou um pico de 26 FPS e problemas para rodar jogos em dispositivos móveis sem recursos de aceleração de *hardware* [Nave et al. 2008]. A abordagem proposta GOTE foi capaz de atingir FPS mais alto em 1280x720 (720p) quando comparada à RenderLink e Games@Large. Além disso, o GID testado em 4G foi menor quando comparado ao atraso de rede de de 16,2 ms relatado pelo EdgeGame [Zhang et al. 2019].

7. Considerações finais

Este trabalho propôs a arquitetura GOTE, que permite que jogos complexos sejam executados em dispositivos móveis com baixa latência, aproveitando infraestrutura de borda com recursos de processamento gráfico, sem a necessidade de instrumentação do código de jogo. Além disso, os principais componentes da arquitetura foram implementados em uma prova de conceito usando WebRTC, GStreamer e NVENC. O código da implementação foi disponibilizado em repositório público para auxiliar futuras pesquisas na área. Todos os experimentos foram capazes de sustentar taxas de processamento de quadros de vídeo desejáveis com qualidade e transmissão estável durante o período testado.

A implementação da arquitetura baseou-se no WebRTC para exibir um fluxo RTP em um navegador *web* para dispositivo móvel. Aplicações comerciais dessa arquitetura devem considerar a implementação de uma API compatível com WebRTC em outras plataformas ou usar outra abstração para fornecer uma transmissão de vídeo de baixa latência para clientes móveis. Serviços como este podem usar um aplicativo móvel com recursos de descoberta de serviço, via SSDP ou outros, para se comunicar com um orquestrador local sem a necessidade de conexão com a Internet. Essa abordagem pode ser útil para eventos fechados, navios de cruzeiro, trens e outros meios de transporte sem conexão estável com a Internet, por exemplo.

O codec H.264 foi aplicado ao vídeo sendo transmitido via RTP para o cliente do jogador. Além deste, o VP8 é um codec moderno e também é suportado por navegadores compatíveis com WebRTC [Mozilla 2021]. Outras iterações dessa arquitetura devem considerar o uso de VP8 e uma comparação com H.264 e outras otimizações de *pipeline* de mídia. Além disso, todos os experimentos contaram com codificação de vídeo acelerada por *hardware* usando NVENC. Trabalhos futuros devem avaliar outras técnicas de codificação e *hardware* para esta tarefa.

Este trabalho implementou e testou experiências para jogos remotos com um só jogador. Portanto, trabalhos futuros devem repetir os experimentos propostos em um cenário multijogador, e investigar seu impacto nas métricas de transmissão de vídeo e no consumo de recursos do servidor de renderização. Trabalhos futuros também devem considerar cenários de estresse de rede para o *streaming* de vídeo e o processo de sinalização, uma vez que todos os experimentos foram executados em condições de rede estáveis.

Além disso, métricas de GID foram coletadas periodicamente e representam apenas a latência do cliente para o servidor de renderização. Portanto, trabalhos futuros devem explorar o GID para jogos com alta taxa de interação do jogador, e os potenciais efeitos de congestionamento de rede e atraso no caminho entre o servidor e o cliente.

A mobilidade da arquitetura GOTE pode ser aprimorada melhorando o algoritmo de orquestração para que ele descubra localmente o *hardware* de borda qualificado e calcule a estratégia de provisionamento do VN mais eficiente de acordo com o jogo solicitado, o número de jogadores em uma sessão de jogo, a distância do dispositivo cliente do jogador, condições de rede entre outros parâmetros. Tal estratégia pode até concluir que executar o jogo solicitado localmente no dispositivo móvel do jogador é a decisão mais eficiente, no caso de recursos insuficientes disponíveis na borda. Também em relação à mobilidade dos dispositivos, há desafios na transferência contínua de sessão de jogo de um VN para outro sem instrumentação do código do jogo e mantendo QoE.

Os jogos remotos de borda representam um potencial sucessor do modelo tradicional de *streaming* baseado em nuvem. Portanto, trabalhos futuros devem avaliar as vantagens e desvantagens, tanto para o jogador quanto para o provedor de serviços, entre a arquitetura GOTE e as arquiteturas de jogos em nuvem, como PlayStation Now e GeForce Now.

Referências

Amin, R., Jackson, F., Gilbert, J. E., Martin, J., and Shaw, T. (2013). Assessing the impact of latency and jitter on the perceived quality of call of duty modern warfare 2. In Kurosu, M., editor, *Human-Computer Interaction. Users and Contexts of Use*, pages 97–106, Berlin, Heidelberg. Springer Berlin Heidelberg.

Cai, W., Shea, R., Huang, C.-Y., Chen, K.-T., Liu, J., Leung, V. C. M., and Hsu, C.-H. (2016). A survey on cloud gaming: Future of computer games. *IEEE Access*, 4:7605–7620.

Chen, J., Koc, U.-V., and Liu, K. R. (2001). *Design of digital video coding systems: a complete compressed domain approach*. CRC Press.

Eisert, P. and Fechteler, P. (2007). Remote rendering of computer games. *SIGMAP*, 7:438–443.

GStreamer (2021). Application development manual.

Hardie, T. and Turner, S. (2021). Rtcweb Status Pages: Real-Time Communication in WEB-browsers (Concluded WG).

Jansen, B., Goodwin, T., Gupta, V., Kuipers, F., and Zussman, G. (2018). Performance evaluation of webrtc-based video conferencing. *SIGMETRICS Perform. Eval. Rev.*, 45(3):56–68.

Lin, L., Liao, X., Jin, H., and Li, P. (2019). Computation offloading toward edge computing. *Proceedings of the IEEE*, 107(8):1584–1607.

Liu, F., Tang, G., Li, Y., Cai, Z., Zhang, X., and Zhou, T. (2019). A survey on edge computing systems and tools. *Proceedings of the IEEE*, 107(8):1537–1562.

Loreto, S. and Romano, S. P. (2014). *Real-time communication with WebRTC: peer-to-peer in the browser*. "O'Reilly Media, Inc.".

Ludwich, M. K. and Fröhlich, A. A. (2011). Optimizing motion estimation for h. 264 encoding. In *Anais do XVII Simpósio Brasileiro de Sistemas Multimídia e Web*, pages 198–204. SBC.

Messaoudi, F., Ksentini, A., Simon, G., and Bertin, P. (2017). Performance analysis of game engines on mobile and fixed devices. *ACM Trans. Multimedia Comput. Commun. Appl.*, 13(4).

Mozilla (2021). WebRTC supported video codecs.

Nave, I., David, H., Shani, A., Tzruya, Y., Laikari, A., Eisert, P., and Fechteler, P. (2008). Games@large graphics streaming architecture. In *2008 IEEE International Symposium on Consumer Electronics*, pages 1–4.

Newzoo (2021). Global games market report.

NVIDIA (2022). NVENC Video Encoder API Prog Guide: Recommended NVENC Settings.

Oros, B.-I. and Bâcu, V. I. (2020). Renderlink remote rendering platform for computer games: A webrtc solution for streaming computer games. In *2020 IEEE 16th International Conference on Intelligent Computer Communication and Processing (ICCP)*, pages 555–561. IEEE.

Pande, A., Ahuja, V., Sivaraj, R., Baik, E., and Mohapatra, P. (2013). Video delivery challenges and opportunities in 4g networks. *IEEE MultiMedia*, 20(3):88–94.

Quax, P., Monsieurs, P., Lamotte, W., De Vleeschauwer, D., and Degrande, N. (2004). Objective and subjective evaluation of the influence of small amounts of delay and jitter on a recent first person shooter game. In *Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*, pages 152–156.

Rosenberg, J. (2010). Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols. RFC 5245.

Rosenberg, Mahy; Matthews, W. C. (2008). Session Traversal Utilities for NAT (STUN). RFC 5389.

Union, I. T. (2021). H.264: Advanced video coding for generic audiovisual services. ITU-T H.264 (V14) (08/2021).

Zadtootaghaj, S., Schmidt, S., and Möller, S. (2018). Modeling gaming qoe: Towards the impact of frame rate and bit rate on cloud gaming. In *2018 Tenth International Conference on Quality of Multimedia Experience (QoMEX)*, pages 1–6. IEEE.

Zhang, X., Chen, H., Zhao, Y., Ma, Z., Xu, Y., Huang, H., Yin, H., and Wu, D. O. (2019). Improving cloud gaming experience through mobile edge computing. *IEEE Wireless Communications*, 26(4):178–183.