GOTE: An Edge Computing Architecture for Mobile Gaming

Gabriel Robaina and Adriano Fiorese[a]

Graduate Program in Applied Computing - PPGCAP, Santa Catarina State University - UDESC, Joinville,

Keywords: Edge Computing, Mobile Gaming, Real-Time Video Streaming.

Abstract: The mobile games market has grown in relevancy compared to traditional gaming platforms. The standard architecture for these games requires the processing of game logic and graphics using the device's own hardware. Alternatively, cloud based architectures for remote gaming on smartphones present high game input delay at a high cost for the service provider. This poses a limitation to the variety and complexity of games that target these platforms as well as constraining user QoE. To address that limitation, this work proposes the Gaming On The Edge (GOTE) architecture, that aims to enable complex games to be played on smartphone devices while leveraging edge computing infrastructure into graphics processing and content distribution systems. A GOTE architecture's proof of concept is developed and tested using WebRTC with an RTP streaming pipeline that exploits NVENC for achieving low latency video encoding. Experimental results show that GOTE architecture is a viable alternative to cloud based remote gaming on smartphones 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.

1 INTRODUCTION

The mobile games market has grown to be the biggest one when compared to traditional platforms like PC and consoles. In 2021, tablet and smartphone games added up to a revenue of 96 billion dollars, 52% of market share, and a trend of growth for the following years (Newzoo, 2021). The standard architecture for these games requires the processing of game logic and graphics using the device's own hardware. This poses a limitation to the variety and complexity of games that target these platforms, since mobile devices have less hardware capabilities when compared to specialized gaming PC or consoles (Messaoudi et al., 2017).

One possible solution to processing game logic and graphics relies on cloud infrastructure instead of the player's mobile device. Companies like Sony, NVIDIA and Paperspace have been providing cloud gaming services (Lin et al., 2019). In the cloud gaming architecture the player interactions are sent to a cloud server and a rendered game scene is sent back as a video stream (Messaoudi et al., 2017). Although scalable, the cloud approach requires the game code to be offloaded to one or multiple cloud servers, making the architecture susceptible to high player input delay on poor network conditions, which leads to low Quality of Experience (QoE). Besides, this model imposes non-trivial infrastructure costs to the gaming service provider since the most part of the computational workload is performed in the computing provider (Cai et al., 2016). Recently, Google announced shutting down its cloud gaming platform Stadia (Google, 2022).

Edge computing (EC) is a paradigm that takes the processing of data to the edge of the network instead of a centralized cloud. Dedicated edge infrastructure, such as cloudlets (Lin et al., 2019), or devices like routers and mobile phones can exchange workloads and achieve low-latency communication inside a local network while still being able to send post-processed data to the cloud if needed (Liu et al., 2019). These edge nodes can make use of virtualization to host the execution of code offloaded from external applications, like mobile games (Zhang et al., 2019). This strategy enables sophisticated applications to the mobile users while extending battery lifetime since most of the computational load is being performed in the edge of the network (Mach and Becvar, 2017). Therefore, the problem being faced by this work is how to provide an opportunity for games to be played on mobile devices using computing resources that are more prone to be found in infrastructure services.

[a] https://orcid.org/0000-0003-1140-0002

Robaina, G. and Fiorese, A.
GOTE: An Edge Computing Architecture for Mobile Gaming.
DOI: 10.5220/0011962000003467
In *Proceedings of the 25th International Conference on Enterprise Information Systems (ICEIS 2023) - Volume 1*, pages 721-730
ISBN: 978-989-758-648-4; ISSN: 2184-4992

To accomplish that, this work proposes an Edge Computing (EC) architecture to mobile gaming named Gaming On The Edge (GOTE), leveraging the proximity between edge nodes and mobile devices to achieve low input delay. Similarly to the cloud gaming architecture, this alternative prevents game logic and rendering from being performed by the player's mobile device by offloading the related code to a nearby edge node, enabling sophisticated games to be played on smartphones and increasing user QoE.

In this sense, this work provides the following contributions: 1) An edge-based remote gaming architecture that aims to enable resource intensive games to be played on mobile devices; 2) A video streaming pipeline that achieves low-latency game video feedback in the edge context, without any instrumentation of game code; 3) An open source proof of concept implementation[1] of the architecture's core in order to assist further research in this area.

This work is organized as follows. Section 2 denotes background concepts to the understanding of the proposed work. Section 3 presents and discusses previous work that took the remote gaming approach to the edge computing context. Section 4 defines the GOTE architecture. Furthermore, Section 5 describes the experiments performed with the GOTE architecture and its results, that are discussed on Section 6. Section 7 concludes this paper and proposes future work.

2 BACKGROUND

This section introduces concepts and tools used on the system architecture and implementation. The Web Real-Time Communication (WebRTC) standard is used on the GOTE architecture for establishing communication between the player client and the rendering server, while the GStreamer framework is responsible for the media pipeline that enables real-time streaming of the game scenes. Also, the built pipeline leverages the hardware based Nvidia Encoder (NVENC) for achieving low latency video feedback.

2.1 WebRTC

WebRTC is a standard that provides Application Programming Interfaces (APIs) that enable real-time Peer-to-Peer (P2P) communication to HTML5 browsers, and it is commonly used for web conferencing. It also enables Real-time Transport Protocol (RTP) streams to be displayed on an HTML5 video tagged page (Loreto and Romano, 2014). RTP takes advantage of the User Datagram Protocol (UDP) instead of the Transmission Control Protocol (TCP) on the transport layer in order to achieve low latency communication and high data throughput. For the peers to connect, first they must go through the signaling process, in which each peer shares information about supported media types, codecs and related configuration by means of the Session Description Protocol (SDP). Also, reachability information of each peer, such as public Internet Protocol (IP) addresses, are collected from a Session Traversal Utilities for Network Address Translation server (STUN), and shared through the Interactive Connectivity Establishment (ICE) technique (RFC5245)(Rosenberg, 2010). Traditionally, all communication and information exchange in this process is mediated by a dedicated signaling server (Loreto and Romano, 2014). In GOTE the streaming server also provides signaling functionalities and acts as a mediator of this process. WebRTC is advantageous for the GOTE architecture since it provides a standard for establishing a streaming session between the rendering server and the smartphone client. It also enables the game stream to be easily displayed on smartphones by leveraging the WebRTC API available in modern browsers.

Fig. 1 presents the WebRTC session sequence diagram used in GOTE for the rendering server. The signaling and rendering modules are components of the GOTE rendering server application. First, both the player client and the rendering module communicate with the signaling module in order to retrieve a common session identifier. Then, the player client creates an SDP offer based on the media type it is able to play. In parallel, the rendering module creates an SDP offer based on the media it can transmit. Next, the ICE candidates are retrieved from the STUN servers by both the player client and the rendering module. These candidates carry the available methods and addresses the peers can use to communicate with each other. Finally, the signaling module mediates the exchange of ICE candidates and SDP offers between the player client and the rendering module. The rendering module can start the video stream once this exchange has finished.

2.2 STUN

The existence of different network topologies can increase the complexity of the connection establishment between peers in the WebRTC environment. For example, peers can be in different private networks relying on the Network Address Translation protocol

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

(NAT) for being reachable over the internet through public IP addresses. GOTE leverages the STUN protocol for enabling the connection between the rendering server and the player client on a wider range of network topologies.

STUN is a protocol used in WebRTC's (Rosenberg, 2010) signaling process for collecting public IP address information from the peers (Loreto and Romano, 2014). The peers can request their public IP addresses from a centralized cloud STUN server, creating a NAT binding on each peer's router. This binding maps a public IP and port to an IP in the private network, enabling peers to be reachable over the Internet. STUN can also be used to maintain NAT bindings via periodic connectivity checks (Rosenberg, 2008). Finally, the peers can exchange their public IP addresses through ICE as part of WebRTC's signaling process.

2.3 GStreamer

GStreamer is a framework for streaming media applications. It enables multimedia pipelines to be built with a broad variety of input and output format and sources. A GStreamer pipeline consists of elements that are interconnected in order to take multimedia data from a source to an output (GStreamer, 2021). Video encoding must be executed by one element of the media pipeline in accordance with the allowed media formats informed by the destination peer at the session agreement. In the GOTE case, since RTP is supported by the client's web browser, the video stream acts as the output of the media pipeline.

2.4 Hardware Encoding

Software encoding involves video encoding using CPU resources. It is capable to achieving high video quality at a speed that varies on the type of architecture and performance of the CPU. It has been widely used on Internet media. However, it is not suitable for real-time video streaming since software encoders can take up to several hours to compress a short video in high definition using recent codecs (Kufa and Kratochvil, 2017).

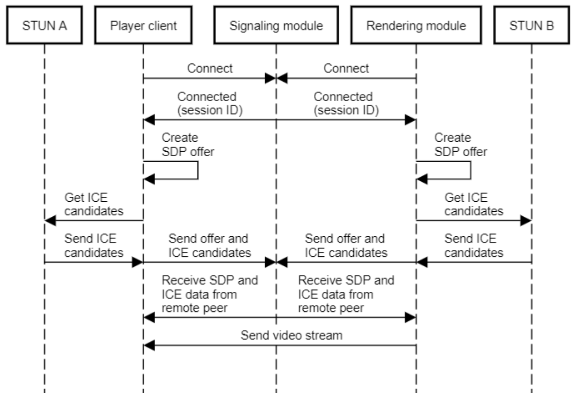


Figure 1: GOTE WebRTC session sequence diagram.

In contrast, hardware encoding uses a dedicated GPU for video encoding tasks with higher performance. The encoding speed from hardware encoding can be up to ten times higher when compared to conventional software encoding (Kufa and Kratochvil, 2017), making it suitable for real-time video streaming. NVENC is NVIDIA's hardware accelerated encoder. It is independent of the graphics performance of the GPU, and during encoding, the graphics engine and CPU are free for other tasks (NVIDIA, 2021). In GOTE's architecture, NVENC accelerates encoding for a H.264 video stream.

3 RELATED WORK

The Games@Large project (Nave et al., 2008) aimed to research, develop and implement an architecture for remote execution of games using code offloading to local servers. This architecture's use cases include hotels, cruise ships and Internet cafes. Instead of streaming the game scenes as a video back to the player's device, this approach requires the scenes to be rendered locally using the mobile device's hardware resources (Eisert and Fechteler, 2007). This was achieved by capturing the commands sent by the game logic to the related graphics API and redirecting it to the player's mobile device for rendering. The tests showed low frame-rates for mobile devices, ranging from 7 FPS on a business strategy game to 18 FPS on a casual game with an average of ≈ 0.34 Mb/s sent over the local network.

The EdgeGame project (Zhang et al., 2019) proposed an EC based architecture for mobile gaming and built a prototype that offloads the processing of game logic and rendering to edge nodes using virtualization. The game scenes are then sent back as a video stream to the mobile user using the WebRTC standard. A congestion control algorithm is used for dynamically adjusting the rate at which data is transferred based on network conditions (Jansen et al., 2018). This standard fits the mobile gaming use case since it provides adaptability on unstable networks and real-time communication of player input and game video stream. In EdgeGame the player can locate available edge nodes by sending requests to a centralized data center, that is also responsible for managing user accounts and providing login services. The tested network delays experienced on the EC approach were significantly lower (16.2ms) when compared to a cloud based one (44.2ms). Also, the user's QoE on EdgeGame was 20% higher when compared to a cloud based alternative.

The RenderLink project (Oros and Bâcu, 2020) also adopts the approach of game code offloading to the edge while sending a video stream back to the client using WebRTC. Instead of using a dedicated edge node for rendering, RenderLink proposes a peer-to-peer (P2P) strategy that leverages idle user devices for this task. In a commercial implementation, users that expose their hardware resources and cooperate in the network may be rewarded with virtual currency. Still, the P2P approach limits the complexity of the games rendered based on the hardware resources available in the network. The tests performed with RenderLink project showed an average frame rate of 55.65 frames per second (FPS) on 720p over a wired connection with standard deviation of 13.48. It was noted that most implementations of WebRTC begin streaming with low quality and gradually ramp up to a stable condition, that accommodates bandwidth constraints, after around 1 minute and 20 seconds. This characteristic of the implementations may hinder the Quality of Service (QoS) during that time period.

3.1 Considerations About the Related Work

EdgeGame (Zhang et al., 2019) and RenderLink (Oros and Bâcu, 2020) took similar approaches to service discovery, leveraging a centralized data center for starting a game session. Even so, a mobile client can discover a Local Area Network (LAN) rendering server through Simple Service Discovery Protocol (SSDP) (Donoho et al., 2020) or Service Location Protocol (SLP) (Day et al., 1999), which decouples the client from any centralized server since they do not need to know about each other before starting the communication to create a game session. Also, the usage of LAN protocols enables discovery to be performed without Internet connection. In the case of WebRTC, this is only possible if a local signaling server exists.

In general, the usage of RTP showed promising results on other projects that took the game scene streaming approach. This protocol was used by EdgeGame and RenderLink in conjunction with the WebRTC APIs, while having the frame rate stabilization time as a drawback. The frame rate and resolution results of the local rendering approach presented by the Games@Large (Nave et al., 2008) project were surpassed by the WebRTC video streaming initiatives (Oros and Bâcu, 2020). Also, the local rendering of graphics is not ideal since it increases the amount of data being transferred to the device, and limits the complexity of games to the player's device hardware capabilities.

Still, for the video streaming approach it is noted that the QoE is directly affected by the frame rate at the rendering source (Oros and Bâcu, 2020). So, the rendering server and the client should have at least equivalent hardware resources in order for the offloading to be worthwhile. Using a peer mobile device as a rendering server, as done by RenderLink, limits the complexity of games that can be streamed based on the hardware resources of the peers (Oros and Bâcu, 2020). On the other hand, the dedicated edge server strategy of EdgeGame is more expensive, but enables complex games to be streamed to mobile clients such as smartphones (Zhang et al., 2019).

This work takes the WebRTC approach for streaming game scenes from a rendering server to the player's mobile client by means of the RTP usage, as suggested by EdgeGame and RenderLink, while constructing a media pipeline that takes advantage of hardware encoding for improving performance and QoE gain. Also, virtualization is used as a platform for instantiating and managing rendering servers using a developed orchestrator. None of the related works provided enough implementation details for reproducibility. Table 1 presents a comparison between the GOTE proposal and analyzed literature that took the remote gaming approach to the edge computing context.

4 ARCHITECTURE OVERVIEW

GOTE architecture is based on direct communication between a rendering server and a player smartphone client. Game input from the player is sent to the server through WebSockets, that renders the game scene and streams video back to the client using RTP. Such server is deployed on a virtual node (VN) of a desktop PC or some other edge device with graphics processing capabilities that is able to start the requested game and a media pipeline for the RTP stream. This pipeline must be efficient enough to stream at frame rates and resolutions that maximize the player's QoE. The smartphone client has a mobile application with a game module, responsible for displaying the game stream and transmitting game controller input, and a discovery module, that enables the discovery of a compliant orchestrator reachable by the client. The orchestrator component is responsible for instantiating and managing VNs while forwarding game inputs from the game module to the corresponding rendering server. Fig. 2 presents an overview of the architecture.

Table 1: Comparison between GOTE and related work.

	Games@Large	RenderLink	EdgeGame	GOTE
Rendering	Player device	Edge server	Edge server	Edge server
Multiplayer	X	✓	✓	X
WebRTC	X	✓	✓	✓
RTP	✓	✓	✓	✓
Hardware encoding	X	Unknown	Unknown	✓
Instrumentation of game code	✓	✓	Unknown	X
Provided implementation	X	X	X	✓

In this solution each VN is responsible for the game session of one player client, and the RTP stream is displayed on the game module in a web browser, since it is compliant with the WebRTC standard.

As described in Section 2.1, the usage of WebRTC requires a signaling server that acts as a mediator for establishing the connection between the peers. Although the GOTE rendering server acts as a mediator during the signaling process, a cloud STUN server is still necessary for acquiring reachability information from the peers. This architecture relies on cloud STUN servers that are accessible by both the client and the rendering server. The communication with cloud servers only happen during signaling and is not impactful to the gameplay QoE.

The VNs are provisioned by the orchestrator component when a gaming session request is received from the player client, as shown on Fig. 3. VNs are responsible for running the game processes and streaming video data to the clients while sharing hardware resources from the host. GPU resources are shared via GPU passthrough, available from Windows Server 2016 onwards (Microsoft, 2022).

Thus, when a client requests a gaming session

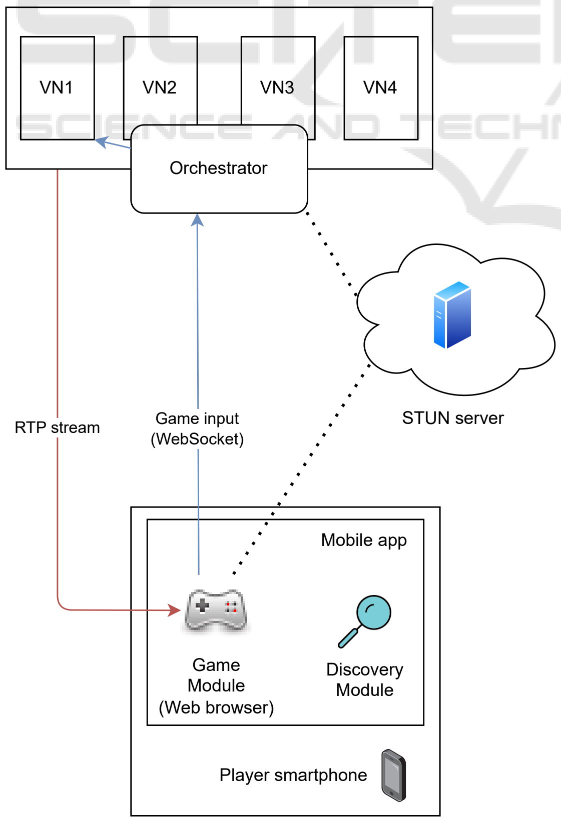


Figure 2: Overview of the GOTE architecture.

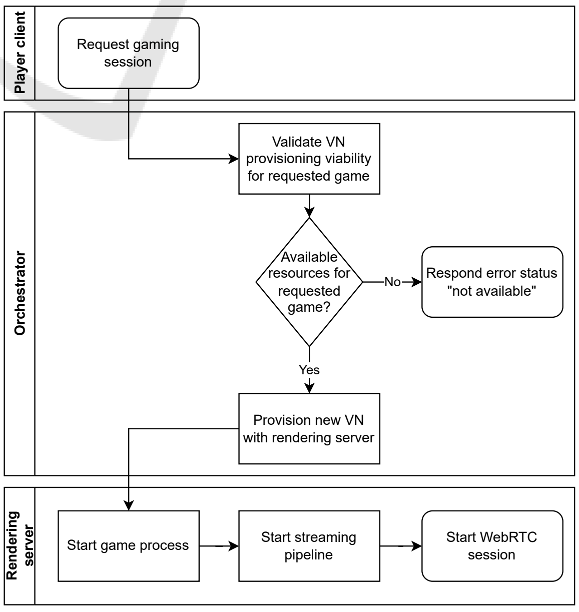

Player client
Request gaming session
Orchestrator
Validate VN provisioning viability for requested game
Available resources for requested game?
No
Respond error status "not available"
Yes
Provision new VN with rendering server
Rendering server
Start game process
Start streaming pipeline
Start WebRTC session

Figure 3: Flowchart of the VN and rendering server provisioning process.

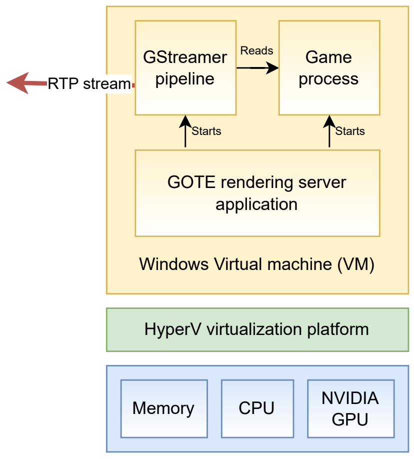

RTP stream
GStreamer pipeline
Reads
Game process
Starts
Starts
GOTE rendering server application
Windows Virtual machine (VM)
HyperV virtualization platform
Memory
CPU
NVIDIA GPU

Figure 4: VN architecture.

to the orchestrator, it verifies if there are enough resources to instantiate a new VN to serve that player. In case it is possible, a rendering server application is deployed on the VN. Such application is responsible for starting the game and video streaming processes. The media pipeline is implemented using GStreamer while leveraging NVENC hardware encoding from NVIDIA GPUs. Fig. 4 presents the proposed VN architecture.

Fig. 5 presents the GStreamer pipeline assembled for the system. The *dxgiscreencapsrc* element is responsible for capturing RGBA (red, green, blue and alpha) data of the game screen at a rate of 60 FPS. The following element, *nvh264enc*, uses the NVENC encoder API to encode the video stream with the H.264 compression. Then, the *rtph264pay* packages the H.264 encoded video stream into the payload of the RTP packets. Finally, GStreamer makes the stream available for WebRTC connections on *webrtcbin*.

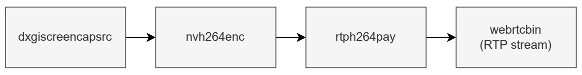


Figure 5: GStreamer pipeline.

WebSockets are used for sending game input commands to the VNs and for all communication related to signaling due to its bidirectional capabilities.

A practical example of the GOTE architecture can be divided into two phases. First, at the game and stream provisioning phase, the player (user) initiates its interaction with the architecture by means of a mobile application. This application is responsible for discovering and communicating with a local orchestrator in order to establish a game session. The orchestrator discovery is performed by the discovery module of the mobile application by means of a centralized cloud server, or a service discovery protocol such as SSDP or SLP, for example. Then, the player can request a game to the orchestrator based on a catalog of games installed on the VN and available for remote play. After receiving the game request, the orchestrator will perform the VN provisioning to a VN host, as presented on Fig. 3. The orchestrator returns an error if there are not enough hardware resources for the requested game. If the provision is successful, the game module of the mobile application opens the web browser in order to begin the WebRTC session establishment, as described on Fig. 1, making the game video stream available and allowing the user to start playing. This establishes the beginning of the remote gameplay phase. At this point, every player controller input, along with a session identifier, will be sent to the orchestrator for forwarding to the corresponding VN responsible for hosting the current game session. Fig. 6 presents the interactions between the components on an example with successful VN provisioning.

5 EXPERIMENTS AND RESULTS

Experiments were conducted to evaluate the GOTE architecture on scenarios of increasing complexity. The ”Local” scenario consists on running the player client and rendering server locally on the same physical hardware in order to establish an architecture base line performance. The ”Wireless LAN” one consists on running the player client on a smartphone and the rendering server on a desktop PC that share the same 5 GHz wireless LAN, emulating a high performance edge computing environment. The last experiment, labeled ”4G”, consists on running the architecture while streaming game scenes from a desktop PC rendering server to a smartphone player client over 4G, emulating a more realistic edge computing scenario.

The performance metrics chosen to evaluate the GOTE architecture comprising video streaming are jitter, packet loss, bitrate, frames dropped and sent per second to the player’s client device. Also, the game input delay (GID) was measured as the time difference between player interaction and command arrival on the rendering server on every second. Jitter, packet loss and frames dropped are metrics directly related to the stability of connection and data transmission between the rendering server and the player client. In addition, the amount of video data being transmitted over the network, and its variation over time, is represented by the bitrate data while GID data was used to assess how different network scenarios impacted game real-time response.

A Windows 10 PC with an i5-9400F 2.90GHz

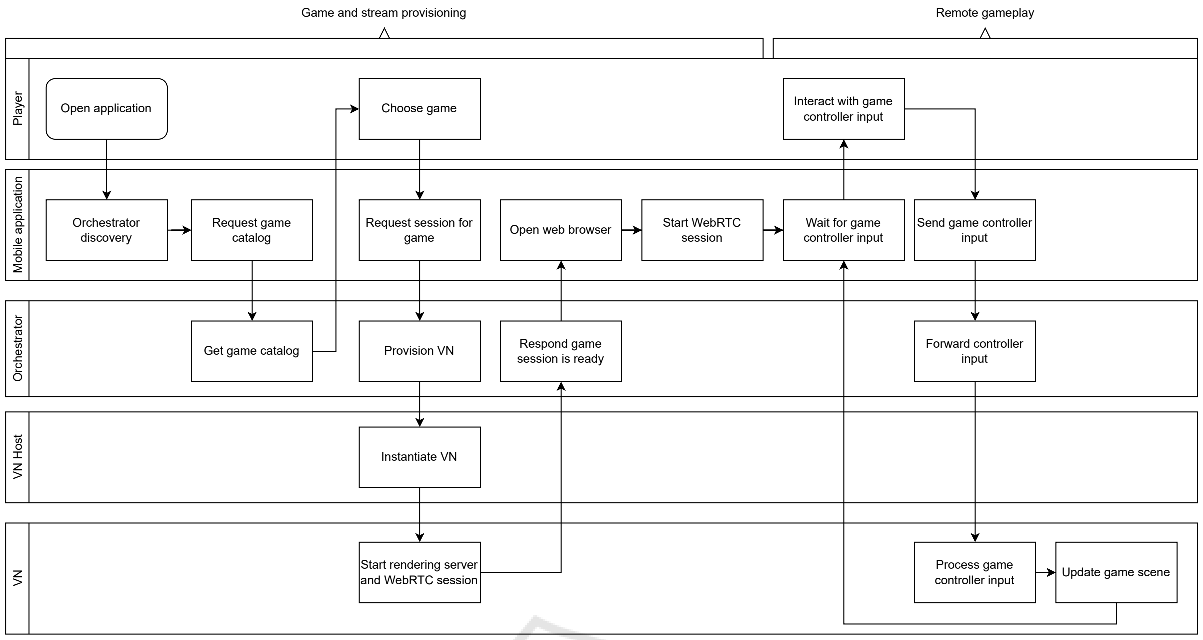


Figure 6: GOTE Components interaction diagram.

processor, 16GB RAM and NVIDIA GeForce RTX 2060 video card hosted the rendering server on all experiments, along with a Redmi M2101K7AI smartphone with 6GB RAM as the player client. The chosen web browser was Google Chrome 96.0.4664.45 running on Android 11. A public cloud STUN server was used during signaling to establish the connection between the peers. The NVENC accelerated H.264 encoder component of the GStreamer pipeline was set to a constant bitrate of 500 kBps with the low latency preset, as recommended by NVIDIA for game-streaming use cases (NVIDIA, 2022). The screen capturing component used a source resolution of 1280x720 pixels (720p) at 60 frames per second. Graphics test 1 and 2 from 3DMark's Time Spy benchmark were transmitted from the rendering server on all experiments. The benchmarks lasted for 150 seconds in total, with a loading screen at the start and in between tests.

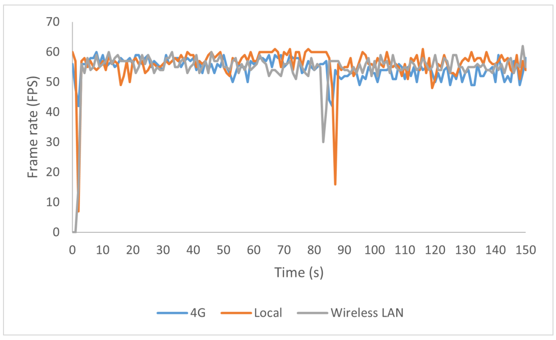
Even though real remote gaming scenarios involve video streaming and game command communication simultaneously, it is unlikely that the traffic involved in sending lightweight user input commands to the server would deeply influence the stream results. Therefore, GID data was collected on independent experiments for all three network scenarios. These experiments ran over a 90 seconds time window, which is sufficient for capturing the impact of the different network scenarios in game input feedback and remote gaming experience. Fig. 7 and Table 2 present the results from all experiments.

These metrics were collected for one rendering server streaming to one player client and hence the orchestration component was not used during the experimentation. Also, the time for the connection establishment related to VN provision and signaling were not taken into account during the experiments.

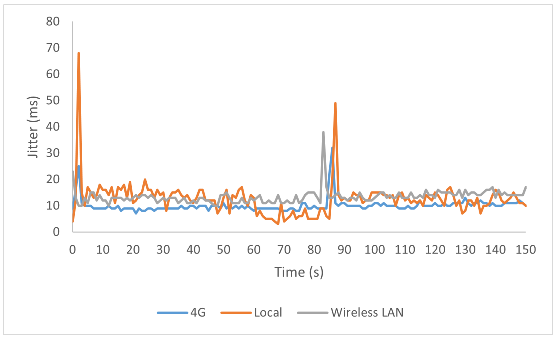
6 DISCUSSION

Frame rate is an important feature of video motion particularly important for the player experience. Although 60 FPS is desired, it is known that some variation between 30 and 60 FPS not impact gaming QoE significantly (Zadtootaghaj et al., 2018). All experiments showed stable frame rates around 60 FPS. However, the Local and Wireless LAN scenarios showed the highest deviation from the average FPS mark of 5.68 and 7.61 frames per second, respectively. A significant part of these deviations was due to frame rate drops at the start of the streaming and close to the 90 seconds mark, at the transitions between the loading screens and the beginning of the benchmarks. There are also peaks in jitter for all experiments at the same time windows.

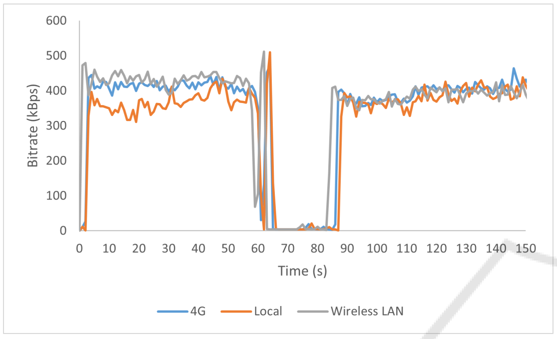
The abrupt transition from the loading screens to the benchmarks may have created a delay in the compression step of the pipeline due to motion compensation, which is a technique that predicts future frames based on camera motion and objects in neighbor frames of the video (Chen et al., 2001). Since H.264 uses motion compensation (ITU, 2021), it im-



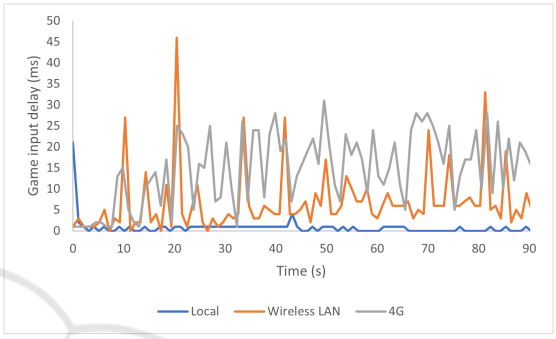
(a) Frame rate



(b) Jitter



(c) Bitrate



(d) GID

Figure 7: Experimental results.

pacted frame rate both at the beginning and at the 90 seconds mark. On a real gaming scenario, this delay in compression is more likely to occur in cinematic oriented games, where camera cuts are frequent, and less likely to occur in strategy games, for example, where motion compensation can take advantage from fewer image changes from one frame to the next. The motion compensation impact can be seen on bitrate data between the 60 and 90 seconds marks, during the transmission of a loading screen, in which the only motion region in video is the loading bar and the bitrate falls close to zero.

The 4G experiment had 4 packets lost during the experiment. This is critical for the RTP stream since UDP has no recovery mechanism for handling lost packets. Also, this metric has significant impact on the perceptual quality of the video stream (Pande et al., 2013). Still, the 4 packets lost on the 140 seconds mark had no impact on the stability of the video, as seen in the frame rate and bitrate data.

The Local experiment had 6 frames dropped during the 150 seconds time window. Running the rendering server and the browser client locally may have impacted performance due to the competition for hardware resources, consequently favouring packet drops. Still, no scenario was significantly impacted by frame drops along the experiments.

Jitter values under 100ms do not damage player QoE even for multiplayer action shooting games (Amin et al., 2013). This metric remained under 100 ms on all experiments, even at the transitions to and from the loading screen, meaning that the buffer used by WebRTC was efficient at sequencing packets for the video stream, and no component in the media pipeline created significant delays in frame delivery.

Table 2: Average and standard deviation results for the performance metrics.

		Average (standard deviation)		
	Unit	Local	Wireless LAN	4G
Frame rate	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)
Bitrate	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)

The average bitrate for all experiments remained between 310 and 350 kBps, meaning that on the average case all scenarios supported a similar rate of video data streaming to the player client. This rate is lower than the 500 kBps specified on the video encoder of the GStreamer pipeline because of bandwidth conditions and motion compensation. Also, the LAN and 4G experiments had higher bitrate variations when compared to Local because of the remote nature of the stream. Is is noticeable that the bitrate variability increased as the complexity of the experiments increased, from Local to 4G.

GID on experiments stayed below 20 ms and increased with the complexity of the scenario. This result favor QoE when compared to 60 ms, considered small even for action online games (Quax et al., 2004). Also, GID metrics were collected every second for a simple game input, which means that, these results may vary for complex games that require higher player interaction rate.

The GOTE architecture enables game streaming on an edge rendering server without the need for any instrumentation of game code, as done by literature work (Oros and Bâcu, 2020). This means that the rendering server is able to stream any game that runs on a Windows PC edge node regardless of the technology it uses. Then, the scale of a commercial application based on GOTE can be increased with a system that rewards virtual currency in exchange for donated hardware resources, enabling flexible ways of monetizing the usage of a service that leverages this architecture for providing remote gaming to customers.

RenderLink reported an average of 60 FPS on a resolution of 1600x900 (Oros and Bâcu, 2020). In addition, Games@Large showed a peak of 26 FPS and problems running games on edge devices with no hardware acceleration features (Nave et al., 2008). The proposed approach was able to achieve higher FPS at 1280x720 (720p) when compared with RenderLink and Games@Large. Also, the 4G tested GID was lower when compared to EdgeGame's reported network delay of 16.2 ms (Zhang et al., 2019).

7 FINAL CONSIDERATIONS

This work proposed the GOTE architecture, that enables complex games to be played on smartphone devices, leveraging edge infrastructure with graphics processing capabilities. Also, the architecture's core was implemented using WebRTC, GStreamer and NVENC. All experiments were able to sustain desirable frame rates, quality and stable streaming across the tested time frame.

The implementation of the architecture relied on WebRTC for displaying an RTP stream on a player smartphone browser client. Commercial applications of this architecture should consider implementing a WebRTC compliant API on other platforms, or use another abstraction to deliver low latency video streaming to mobile clients. Services like this make use of a mobile app with service discovery capabilities, via SSDP or others, to communicate with a local compliant orchestrator without the need for Internet connection. This approach can be useful for closed events, cruise ships, trains and other transportation means without stable Internet connection, for example.

The implementation applied the H.264 codec to the video being streamed via RTP to the player client. Besides this codec, VP8 is also supported by WebRTC compliant browsers (Mozilla, 2021). Further iterations of this architecture should consider the usage of VP8 and a comparison with H.264 and other media pipeline optimizations. Also, all experiments relied on hardware-accelerated video encoding using NVENC. Future work should evaluate other encoding techniques and hardware for this task.

This work implemented and experimented with single player experiences for remote gaming. The value proposition of a remote gaming service increases if it supports multiplayer gaming. Therefore, future work should repeat the proposed experiments in a multiplayer scenario, and investigate its impact in video streaming metrics and the resource consumption on the rendering server. Further work should also consider network stress scenarios for the video streaming and the signaling process, since all experiments ran under stable network conditions. Also, GID metrics were collected periodically. Therefore, stress scenarios should also test GID for games with high player interaction rate.

GOTE architecture's mobility can be enhanced by improving the orchestrating algorithm so that it discovers eligible edge hardware locally, and calculates the most efficient VN provisioning (code offloading) strategy according to the game being requested, the number of players on a gaming session, the distance from the player client device, network conditions and other parameters. Such strategy may even conclude that running the requested game locally on the player's smartphone device is the most efficient decision, in case of insufficient edge resources available. Also regarding mobility, there are challenges in seamless game session handover from one VN to another without instrumentation of game code while maintaining QoE.

Edge remote gaming represents a potential successor of the traditional cloud based streaming model. Therefore, future work should evaluate the advantages and disadvantages, both for the player and the service provider, between the GOTE architecture and cloud gaming architectures such as PlayStation Now®and GeForce Now®.

ACKNOWLEDGEMENTS

This work received financial support from the Coordination for the Improvement of Higher Education Personnel - CAPES - Brazil - PROAP-AUXPE/PDPG-CONSOLIDACAO-3-4 1367/2022.

REFERENCES

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.

Day, M. D., Perkins, C. E., Veizades, J., and Guttman, E. (1999). Service Location Protocol, Version 2. RFC 2608.

Donoho, A., Roe, B., Bodlaender, M., Gildred, J., Messer, A., Kim, Y., Fairman, B., and Tourzan, J. (2020). UPnP Device Architecture 2.0.

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

Google (2022). A message about Stadia and our long term streaming strategy.

GStreamer (2021). Application development manual.

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

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.

Kufa, J. and Kratochvil, T. (2017). Software and hardware hevc encoding. In *2017 International Conference on Systems, Signals and Image Processing (IWSSIP)*, pages 1–5. IEEE.

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.”.

Mach, P. and Becvar, Z. (2017). Mobile edge computing: A survey on architecture and computation offloading. *IEEE Communications Surveys Tutorials*, 19(3):1628–1656.

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).

Microsoft (2022). Deploy graphics devices using Discrete Device Assignment.

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 (2021). NVIDIA Video Codec SDK.

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.

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.