UCLouvain

École polytechnique de Louvain

A Network Analysis of Cloud Gaming

Author: **Florian VRANCKX**
Supervisor: **Olivier BONAVENTURE**
Readers: **François MICHEL, Gorby KABASELE**
Academic year 2020-2021
Master [120] in Computer Science and Engineering

Abstract

Cloud gaming is a new paradigm on the video game scene. The game is no longer running on the user's hardware but everything is rendered in the cloud. This new way of playing video games took a step forward in 2019 as major IT companies released their own cloud gaming platforms. However, they did not disclose their used protocols and techniques. In order to work properly, it requires a low latency and a large bandwidth. By analyzing hours of recorded network trace of both Stadia from Google and Geforce Now from Nvidia, this document aims at understanding the mechanism hidden behind this new technology. We have also submitted these platforms to various perturbations to understand their reaction when running on a sub-optimal network.

Acknowledgements

I am extremely thankful to my supervisor, Olivier Bonaventure, Professor at the Uclouvain, for his involvement in every step of the analysis and writing of this work, for his advice and guidance through this process, and for putting pressure on me when I start to be late in my work.
I am grateful to my reader and advisor François Michel, Assistant at the UCLouvain, for taking the time to listen and review my analysis, for his interest in this topic, and for his advice during our weekly meetings.
Finally, I owe a thanking to the whole team of assistants with whom I've interacted during the analysis process and who invested their time in listening to my problems and helping me find solutions.

Contents

Chapter 1

Introduction

The video game industry is a constantly evolving matter since the first video games around 1960. Multiple ways to play that came and disappeared with the evolution of technology. When we think about video games in the 70's era, it was mostly about arcade rooms with a big bulky station that cost too much for the average user. This paradigm evolved with the introduction of the first game console and personal computers around the 80's. Namely with the first Atari game console that was released in 1978 and the Commodore 64 in 1982. From that point and forward, video games were mostly played at home. The market evolved taking a bigger and bigger place in the economy and at some point, it became obvious that some of the biggest actors of the IT industry would start getting interested in that market segment. We have seen Microsoft and Sony releasing their own console. Microsoft made changes in its own OS to better accommodate games and instigate itself as the sole OS to be able to play games on a computer.

In recent years, the network infrastructure improved enough to allow large bandwidth and low-latency networks between the average user and large companies. In 2010, we have seen the first try to use the network to make a cloud version of the gaming. A version where the hardware would not need to be physically present at the place where the game is played. The idea behind this was that the user could pay a monthly fee to have access to high-tier hardware to play games on any device rather than rely on his own expensive hardware. The issue was obviously that any kind of latency on the input and on the display would result in an immersion-breaking experience. In 2010 the idea didn't work because there were not enough customers interested in this paradigm with a good enough connection to the network. There were other attempts at this.

It was only in 2017 that the subject of cloud gaming started to gain in popularity and that the company Shadows started the first economically viable project. In 2018, 2019, and 2020 we have seen a few of the IT giants get involved in the new

way of playing video games. Nvidia, Google, Sony, and Microsoft all released their own version of cloud gaming under their respective name, Geforce Now, Google Stadia, Playstation Now, and xCloud.

1.1 Current status and history

1.2 Actors

There are 5 main actors in the domain, each of them offering a service that varies in terms of available games and licensing agreements. In this document, we will focus on Google Stadia and Geforce Now.

1.2.1 Shadow

Shadow is the cloud gaming service of a company called Blade. Blade is a french company that was the first to introduce a cloud gaming platform to the general public. It is still available today. Its service is built around the PaaS (Platform-as-a-Service) model. it enables the user to rent a "Shadow Box" which is essentially a virtual Windows PC with a dedicated Graphic card and a few shared CPU cores. Even if the Shadow service is actually profitable for the company, Blade is currently filing for bankruptcy. Currently, the service is still online but recent events indicate that the bankruptcy might lead to an end of the service rather than a change in ownership.

1.2.2 Geforce Now

Geforce Now is Nvidia's own attempt at cloud gaming. Nvidia has always heavily relied on its "gaming" line of products and, as of 2021, has an almost complete monopoly over the GPU market. It started its cloud gaming service in 2013 on the Nvidia Shield, a handheld console developed and sold by Nvidia. It's only in 2017 that they released a first beta version for PC with access to only one game. Finally, they opened in 2020, a browser version of their service that relies on Chromium to run. Allowing any device that can run a version of the chromium browser to access the service. Their system is similar to Shadow with the exception that it's closer to a SaaS (Software as a service). The user still sees some aspects of the virtual machine but cannot install anything and can only play the game associated with that virtual machine. Also, the machine is restored to its default state when the game is closed.

1.2.3 Google Stadia

Stadia is Google's cloud gaming service. The service was announced in 2019 with very ambitious goals in mind. Google announced the creation of Stadia Games and Entertainment, a game development studio alongside Stadia to develop exclusive games for the platform. This game studio was then close in 2021 without having released a single game. Stadia offers a pure SaaS experience, the user does not interact at all with the hosting virtual machine and can only play the game he purchased on the Stadia Store. As for all other Cloud gaming services, a monthly fee is also required to keep the service available.

1.2.4 xCloud

The name xCloud is a reference to the gaming console Xbox. xCloud is the cloud gaming service proposed by Microsoft. Contrary to the other platform cited in this chapter, xCloud is not a way to play PC games on a remote VPS (Virtual Private Server). xCloud only allows the user to play titles that are available on the Xbox One with the same parameter of quality and frame per second as a real Xbox One S would render. To achieve this, Microsoft says they have developed a blade server with the internals of four Xbox One S. Microsoft also said they were going to use their own cloud infrastructure (Azure) to host and deploy those servers around the globe [17].

1.2.5 Playstation Now

Similar to xCloud, Sony made its own attempt at cloud gaming. Its service is centered around offering the Playstation experience as a cloud gaming platform. As for their Microsoft counterpart, the video quality and framerate that the user can expect is exactly the same as if the game was rendered on an actual Playstation 2,3,4 or 5.

Chapter 2

State of the art

2.1 Real time communication over the internet

Real-time communication has been a consideration since the first release of a VoIP client in 1991 to the general public [7]. From the first pioneer Net2phone from an American company to Teams and Zoom in 2021, there has always been a demand from the user of the internet for an alternative over the classic and expensive phone lines. This kind of communication saw many various improvements as the network capability grew over the years. Starting with the audio quality, just needed to be enough for two people to understand each other, it became better than traditional phones. It was then improved with the addition of video allowing for video calls. This was made famous and widely available by Skype around the year 2005. Since then, the technology as improved even more with the addition of video conferencing, all of that leading to the current situation. Nowadays, video calls and conferencing have become a necessity with the current situation as of the events of 2020-2021.

2.1.1 VoIP Protocols

Skype, the most famous video conferencing platform used in the years 2000s, did not disclose the protocols of communication they used [2]. In the meantime, a first standard was published with the RFC 1889, which was trying to standardise the real-time network communication with the RTP protocol in 1996. [3]. It was later superseded in 2003 with RFC 3550 [4]. The change was mostly about the algorithm, the packet definition and fields were unchanged.

2.1.2 RTP

RTP stands for Real-Time Protocol. It was disclosed as a standardised way to handle real-time communication, ass audio for calls, video conferencing or even

regular real-time data stream like measurements. RTP in itself was introduced to work over UDP and does not handle re-transmission and does not provide any guaranty of ordered or un-ordered delivery of packets. On the other hand, RTP provides a series of functionalities adapted for end-to-end real time transmission. It provides a way of verifying the arrival of the packet and proposes a series of algorithms to improve the quality of the received stream of information with various congestion control techniques that take the stream type into account. Its capability is the monitoring of delivery over large multicast networks with the added usage of RTCP (Real-time control protocol).

RTP scenario

The RFC of the RTP includes multiple scenarios where the usage of RTP is recommended, starting with the Simple Multicast Audio Conference. The scenario described by the authors is an audio call with multiple receivers and one emitter like we could observe in a conference. In this scenario, packets would be distributed as a multicast defined in the RFC 1112. Therefore, recipients can join and leave the multicast group at any time. Each participant will then send their own RTCP report to the source in order to indicate any network congestion to the emitter.

The second scenario is the same with the use of an additional video on top of the audio conferencing. In this scenario, the video would be transmitted through a different UDP port. This would allow to separate the two streams completely and allow users to receive only one of the mediums if they decide it.

The third scenario proposed in the RFC talks about Mixers and Translators.In the case of one participant having a lower bandwidth connection than the other, the RFC suggests an alternative solution to avoid reducing the video quality for all participants. The use of an RTP-mixer close to the lower bandwidth client could be used. The main purpose of this mixer would be to re-encode or even mix the audio and video stream to the same stream and deliver the content at a lower bitrate. On the other hand, this scenario proposes the use of RTP-Translator for those whose firewall would not allow receiving any IP packet. The idea would be to use two Translators on either side of the firewall to funnel in the multicast packet.

In the scope of this document, this shows the usage the actors made of the RTP was not what it was designed for. This document will relate how Geforce and Stadia used WebRTC in combination with some experimental modifications to the RTP and RTCP report to make their own platform a good fit for cloud gaming. A usage with much lower tolerances for latency and loss as any kind of variation in the packet delivery can seriously hinder the user experience.

The RTP packet format

The RTP packet format as specified in the RFC, goes as follow:

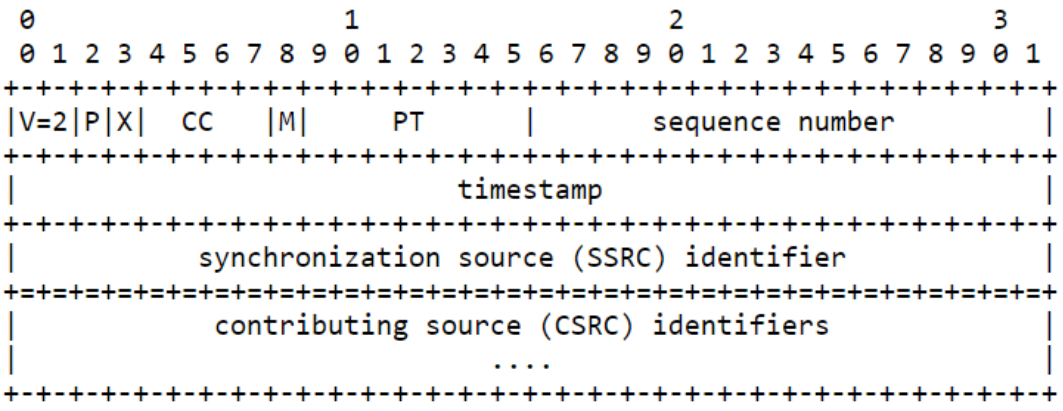


Figure 2.1: Layout of the RTP packet

V-Version

This field has a fixed value of 2. Values 1 and 0 were used prior to the first RFC and should not be used.

P-Padding

This value indicates if the packet contains padding at the end. This value is set to 1 if there is at least one bit of padding at the end of the packet. In this case, the last octet of padding should include the length of the padding including itself.

X-Extension

If this field is set to 1, the packet must have exactly one header extension.

CC-CSRC count

This field is used to indicate the amount of contributing sources. Since this field is limited to 4bit, only 15 CSRC can be used at the same time.

Field M-Marker

This field is the marker bit. It has no defined use but the application can choose to use for various custom purposes. It is commonly used in video transmission to signal the end of a frame.

PT-Payload Type

This field is reserved for the payload, it gives information about the payload. Values 1 to 14 are assigned to a known type. For example, type 34 is assigned to the video codec H263. Values from 72 to 76 are reserved. Values from 77 to 96 are unassigned in theory. They are sometimes used when all the dynamic values are already assigned. Finally, payload types 96 to 127 are dynamic formats, it is up to the application to decide the content of these formats.

Field sequence number

The sequence number is a 16bit value indicating the place of this packet in the sequence of packets. It is specific to the SSRC. If multiple RTP streams operate in parallel over a transport, they must each use their own sequence number. The initial value should be random for security reasons.

Timestamp

The timestamp value should be linked to a value of a monotonic linearly increasing clock. However, the actual value increase is linked to the payload type. With the exception of dynamic payload type where the application can decide how the timestamp increases. The RFC still recommend using a value related to the sampling of the source. The RFC also recommend using a resolution greater than one per packet but this is not always how things are implemented (see section 4.3.1). As for the sequence number, the start value should be random for security purposes.

SSRC

This value is used to identify the source of the media. Since multiple RTP streams can occur over the same UDP port, the way to tell them apart is via this value. This value needs to be different for the different parts of a stream. Traditionally, a video conference call is going to be composed of a video stream and an audio stream, each with its own SSRC.

CSRC

This field is optional and can be repeated up to 15 times. This field is used to link different sources together. This field is mainly used for an audio conference with multiple people speaking at the same time. All of the audio streams will have a different SSRC. Their value will be added as CSRC to each other so that the

receiver knows that it can mix them together in order to get the resulting audio stream.

2.1.3 Header Extension

RTP includes a mechanism for extending the header. The extension needs to go directly after the classic header. The layout of the header extension preseted in Figure 2.1.3:

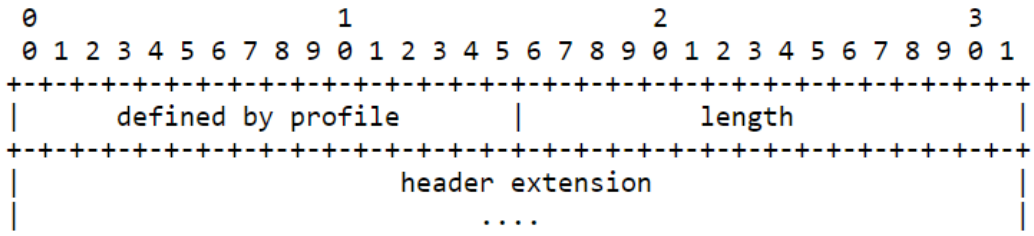


Figure 2.2: Layout of the RTP packet header extension

The length field is the number of extension present in the header rather than the total size in bytes. Various RTP extension headers were used during the analysis of chapter 4 but are discussed in more detail in the WebRTC section.

2.2 RTCP - Real time control protocol

The RTCP is an integral part of RTP. It consists of packets that report various information about the RTP stream. This feedback can be used for multiple purposes by the source. Its primary goal is to inform the source when the receiver does not receive all the packets it should for any reason. The source can then take various actions like changing the codec and/or the quality to reduce the stream bandwidth usage. RTCP is also used to introduce new information. For example, the RTCP can be used to inform a source that there is a new receiver on the multicast stream or that a receiver is leaving the stream. RTCPs are sent periodically by both the receiver and the emitter. There are 5 types of RTCP packets, RR (receiver report), SR (sender report), SDES (session description), BYE (end of participation) and APP (application-specific). Only SR and RR are in the scope of this document and will be detailed.

2.2.1 Sender report

The layout of an RTCP-SR packet can be found in Figure 2.3. Its layout is a compound between one header that contains the basic information about the layout and a series of reports. The main header layout includes some of the same fields as a classical RTP packet with a few exceptions. Since this packet is generated for a single source, only the SSRC is mentioned as the other sources mentioned in the CSRC field need to send their own report. Therefore the CSRC count field and the marker field are replaced by the RC (Report Count) field. This field is used to indicate the amount of report block present in the packet. The sequence number is replaced by a length field that counts the length in bytes of the RTCP packet minus one.

After that, multiple reports can be aggregated in the packet. Each report should report for one stream. In a classic usage with a video and an audio source coming from the same sender, we should see two report blocks. Each of these includes information about the received packet like the fraction of loss packets, the number of packets received, the jitter, and some information about the last SR in case it never arrived.

2.2.2 Receiver report

The layout of the receiver report is the exactly the same as the sender report. The only exception is the payload type set to 201 rather than 200.

2.2.3 Packet size consideration

From these information, we can deduce that a classical sender/receiver report should include 8 bytes for the header, 20 bytes for the sender info, and multiple 24 bytes report blocks. The RFC mentions that these reports should be sent as frequently as possible but staying under 5% of the total bandwidth usage.

2.3 STUN

The STUN protocol [8], or Session Traversal Utilities for NAT, is a protocol intended to allow a client hidden behind a NAT to discover its public IP address and port assigned by the NAT. This allows the creation of a UDP session through its NAT. It's also meant to be used to maintain the assigned port in the NAT by sending periodic keep-alive packets if no traffic is sent for a period of time. STUN works with almost any type of NAT with the exception of symmetric NAT.

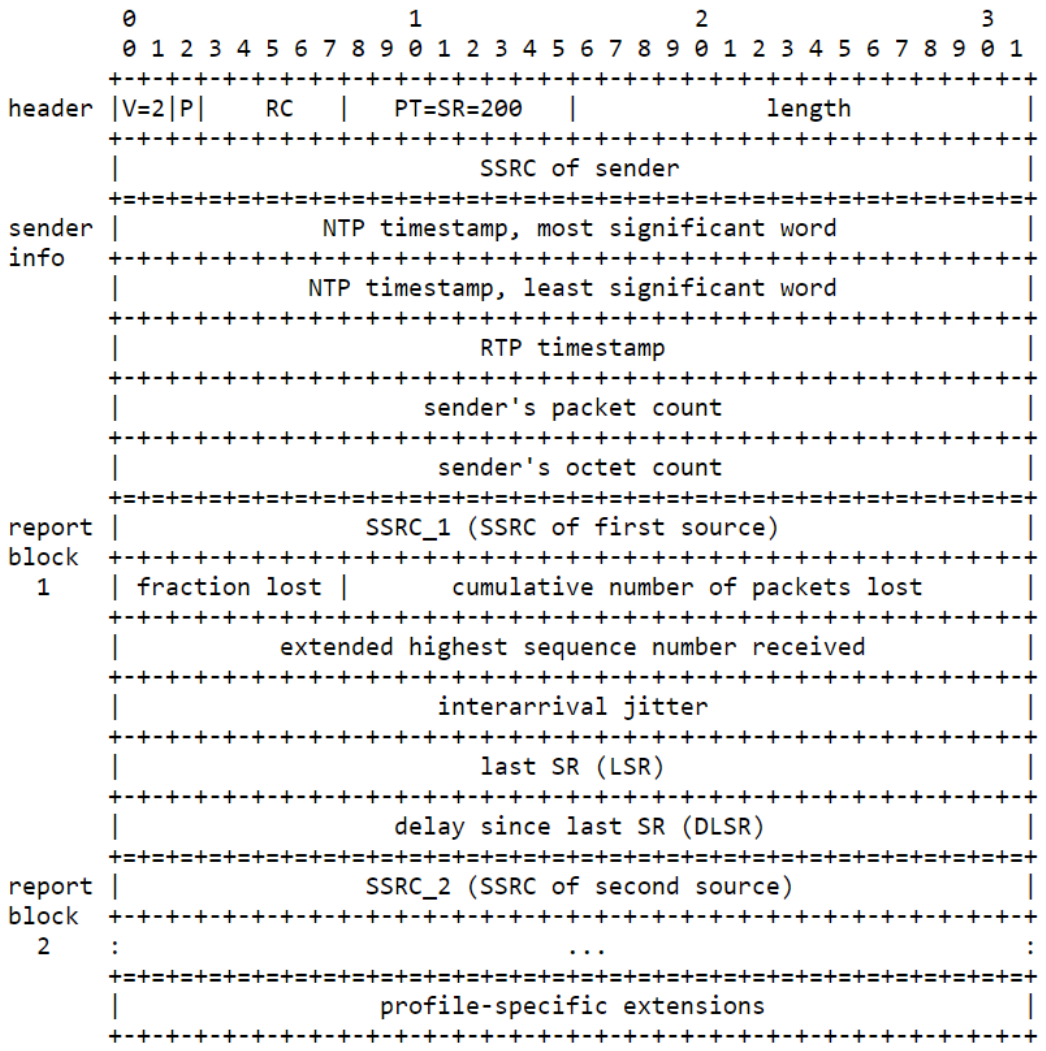


Figure 2.3: Description of a SR packet

2.3.1 Operation of STUN

STUN needs two actors in order to work, the client and a server. The client will send the STUN packet over UDP to the STUN server. Once the STUN server receives that packet, it is checked to gather the necessary information. Since the NAT of the client will have replaced the information in the UDP header with the client public IP address and used port, the server can read these fields in order to generate its answer. It can then send its answer to the client to let it know its own information. Since NAT can sometimes modify the address in the payload of a packet, this value is XOR mapped in order to make it invisible to the traversed NAT

and allow the client to decode its address once the packet is received. Since this is done over UDP, STUN should be using a retransmission timer on the client-side.

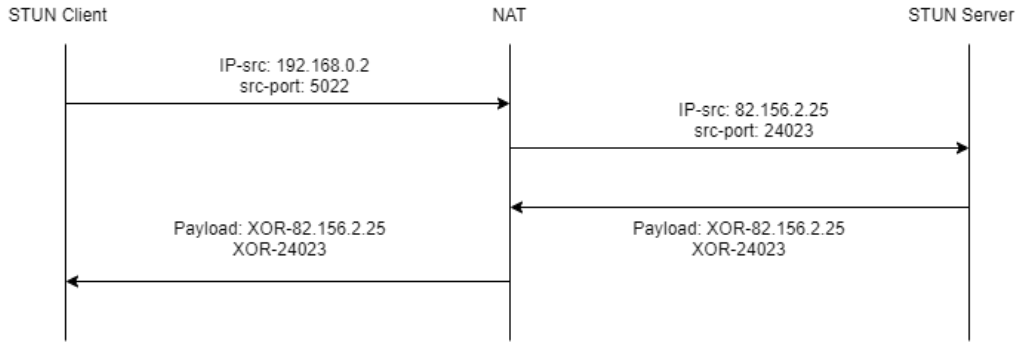


Figure 2.4: Diagram of the STUN packet flow

2.4 TURN

In some cases, the STUN protocol [9] is not sufficient to traverse the NAT and the use of TURN is needed. TURN was designed to be used by the ICE protocol to create a relay server between a client and the internet to traverse the NAT and allow incoming connections. Since the TURN service needs for high bandwidth to relay all communication, this solution is expensive and should only be used in case of a direct connection when STUN failed. Moreover, it can increase the latency for latency-sensitive applications.

2.4.1 Operation of TURN

In the classical operation of TURN, the first client makes use of TURN commands to trigger an allocation on the server. It then receives back a response containing a "relayed transport address". The client can then communicate this address to other peers on the network in order to allow them to connect to the client through the TURN server. The TURN server has to maintain a state with the first client address and port used through its NAT to forward the packets received through the network. This means that even a NAT that only redirects packets from the same pair of IP and port will accept the packet as they all come from the same TURN server.

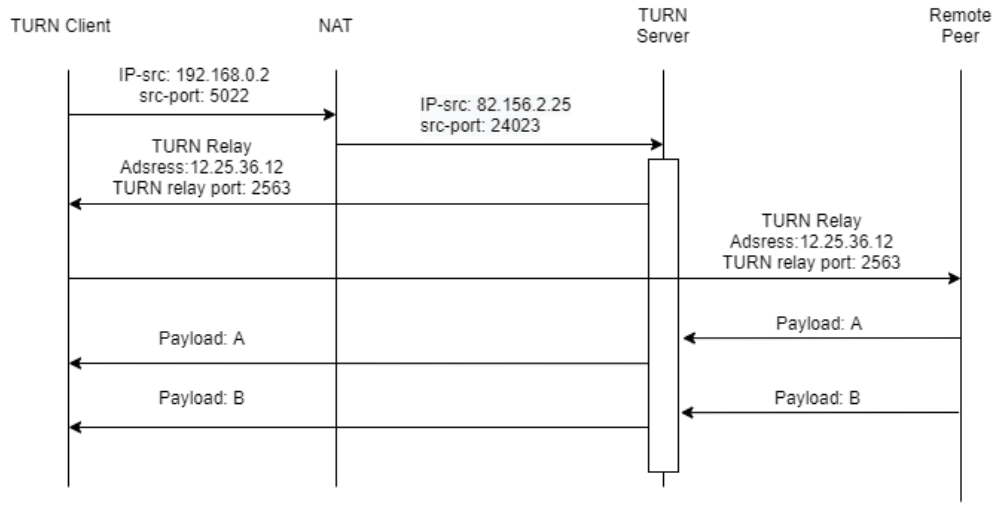


Figure 2.5: Diagram of the TURN packet flow

2.5 ICE

The ICE protocol or Interactive Connectivity Establishment [10] is a protocol used to establish a UDP connection between two peers over the network through their respective NAT configuration. Through ICE, it is assumed that the clients are ignorant of their own topology at first. ICE's role is to collect a series of potential pairs of IP and ports through various means and try them all until it finds one that works.

2.5.1 Operation of ICE

For ICE to work, a third server is used to relay the information of both peers. Initially, both peers have a connection to the third server. The ICE protocol will collect of each of the peers a series of potential IP and port pairs that could be used to connect to that peer. They are then sorted in a priority order. Starting with the physical candidate gathered on the network link, then the STUN candidate and finally a TURN candidate. This could be extended with other ways of gathering candidates if needed. Through the third server, both peers get all candidate pairs from the other peer. Then all the candidates get tested in order to find one that works. When testing the STUN candidate, both peers actually sends a STUN request to the other peer on the same port that will be used for the data once the link is in place. This results in a 4-way handshake of stun binding request.

2.6 WebRTC

WebRTC is an API developed by the W3C and the IETF in 2011 and its implementation in browser started around 2013. The idea behind this API is to simplify real-time peer-to-peer communication. It is now supported by all major browsers and chromium placed itself as a pioneer by extending the WebRTC API [6]. WebRTC is built around different protocols including ICE, STUN, TURN, DTLS and RTP (SRTP) and make the configuration of these protocols much easier for the programmer.

The API will first try to connect the two peers together using the ICE (Interactive Connectivity Establishment) protocol. Once the two hosts get connected they will establish a DTLS handshake before setting up any stream that is requested by the application. There are two main streams that can be built in WebRTC, either an RTP stream to carry video, sound, or a data stream. In the case of an RTP stream, an AES key will be shared over the DTLS connection and be used as the key for the encrypted variation of RTP, SRTP. Meanwhile, in the case of a data stream, the data will be carried over the DTLS connection.

2.6.1 WebRTC's SDP

The SDP or Session descriptor protocol is at the heart of the WebRTC configuration. It's this part that allows the application to set up the different parameters of the underlying protocols like RTP. The SDP packet carries vital information like an ICE password and configuration in order to allow the connection, RTP payload type, RTP header extensions and so on.

This packet is built with a combination of key and value. Each key designates a different parameter to be set. Only the most important parameters used in this document will be discussed in this section

RTPMAP

The RTP map is used to declare the payload type of the RTP communication if they are in the dynamic range. This allows the programmer to link a dynamic type not present in the base RTP specification to a codec that is implemented in the WebRTC API. A good example is the VP9 codec which is supported officially by the WebRTC implementation in the Chrome browser but not in the RTP payload allocation. Therefore if the programmer wants to link this codec to the RTP-payload 111, it needs to add the following line to his WebRTC's SDP configuration: `a=rtpmap:111 VP9/90000`.

Once the payload type is linked, the line "a=ssrc:" followed by a pseudo-random 32bit value in base10 has to be used in order to assign the source identifier to the

RTP stream.

RTCP-FB

The command `rtcp-fb` is used to add a non-conventional report at the end of the RTCP packet. It allows the application to change the way the RTP stream reports error and instability. These commands need to be set for each different payload type. For example, if we want to use negative acknowledgements rather than regular acknowledgements for the RTP stream with payload type 109, we can use the command `a=rtcp-fb:109 nack`. A list of all important `rtcp-fb` for this document is provided at the end of this section.

EXTMAP

The `extmap` key is used to add a header extension to the RTP stream. WebRTC has a list of defined header extensions that can be used for various purposes. Google has added some additional header extensions in their implementation of WebRTC available in their Chrome browser. In the command, the header ID field has to be set: for example, if we want to add the header extension `playout delay` with header id 2. The command would be `a=extmap:2 playout-delay`. A list of all important header extensions for this document is provided at the end of this section.

2.7 WebRTC's custom header extension for RTP

Various custom header extensions have been introduced recently by Google in Chrome. The date of the release and the date at which the documentation has been written is very close to Stadia's release and might indicate that those features of Chrome's WebRTC implementation were added with cloud gaming in mind.

2.7.1 Playout Delay

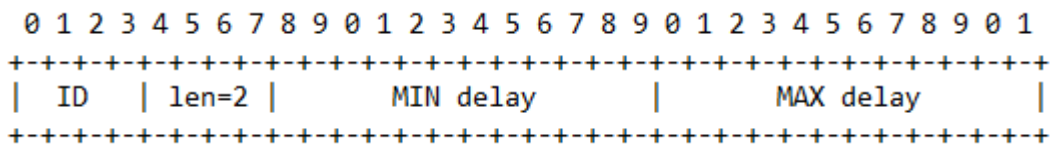


Figure 2.6: Layout of the Playout Delay header extension

Playout Delay is an experimental header extension, the documentation about this feature is only available on Google's git [11].

This header extension is intended to limit the amount of time that the client has to render the received frame. The documentation of this feature includes three possible scenarios: Cloud gaming, video playback and interactive communication.

For cloud gaming, it is said that playback time is critical. Any latency is very likely to hinder the user experience and the stream should prioritise the low latency rendering of the frame rather than do any smoothing on the video. In this case, the documentation suggests setting both the max end min delay to 0.

For the video playback, the documentation says that for the user experience to be the best as possible, there should be no change in the decoding delay. They suggest not to set the min delay to 0 in order to allow the decoder to have some buffer in case of network instability. They suggest using the same value for the max and the min delay so that the decoder is forced to maintain the same playout delay through the playback.

Finally, for interactive communication, this scenario is more hybrid as sometimes it is best to favour a low latency if the networks allow it but in the meantime the rendering delay might be increased dynamically in order to smooth out the playback in case of a transient network imperfection.

These delays are said to be best effort rather than an absolute value that the decoder has to follow.

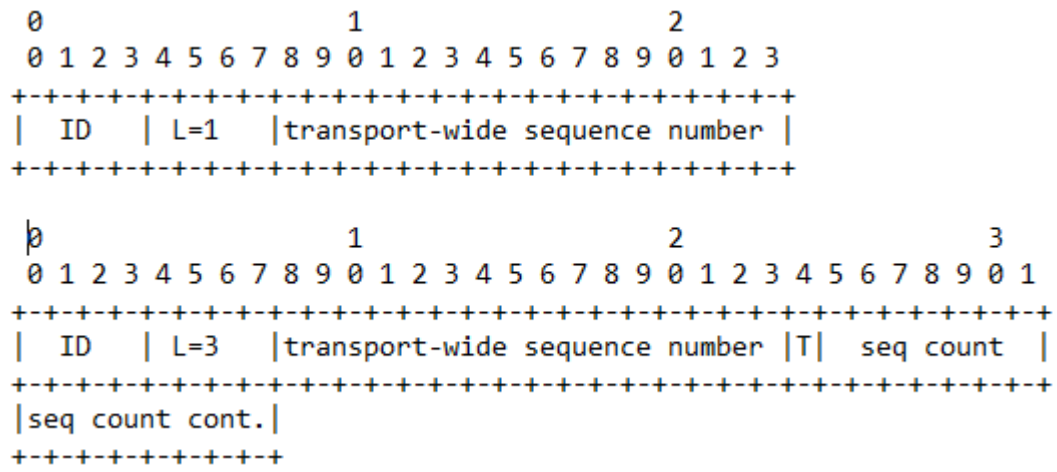
2.7.2 Transport-wide congestion control

The transport-wide congestion control is also an experimental extension header. Its documentation is only available on Google's git [12]. This is the second version that Google made and its name is actually "Transport-wide congenstion control 02"[1].

The advantage behind a transport-wide sequence number rather than using the RTP's media sequence number is that it allows detecting a packet loss quicker. For example, if the sender was to send 3 video packets followed by an audio packet and again 3 video packets. If the audio packet gets lost, without a transport-wide sequence number, this loss would only be detected at the reception of the next audio packet. Meanwhile in this case the loss is detected as soon as the next packet is received and correction or re-transmission can be triggered earlier.

This header extension has two different possible layouts. The second version is used to request feedback. When the T field is set to 1, the feedback needs to contain timing information. The seq count specifies the amount of packet that should be included in the feedback message and if this field is set to 0, no feedback should be generated. This is equivalent to using only the 1-byte header.

[1]This is not a typo, this is the actual name on Google's documentation



2.7.3 Absolute Send Time

This extension is also an experimental extension and the only documentation is on Google's Git [14]. The idea behind this extension is to provide a timestamp "as close to the metal as possible". The documentation notes that this timestamp should be set last on the packet as close as possible to the sending.

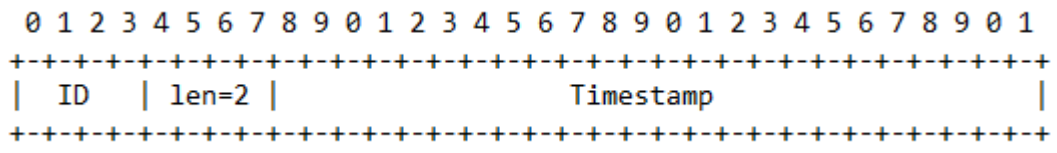


Figure 2.7: Layout of the Absolute Send Time header extension

2.7.4 Color Space

Color Space [15] is a header extension used in the rendering of HDR (High Dynamic Range) [16] content. This allows the use of a greater luminescence range and a bigger color volume than traditional SDR. There is two possible headers for this extension as presented in fig2.8. Both are valid and the only difference is that the second allows for the use of more information about the rendered scene. This header extension should only be present on the last packet of each frame.

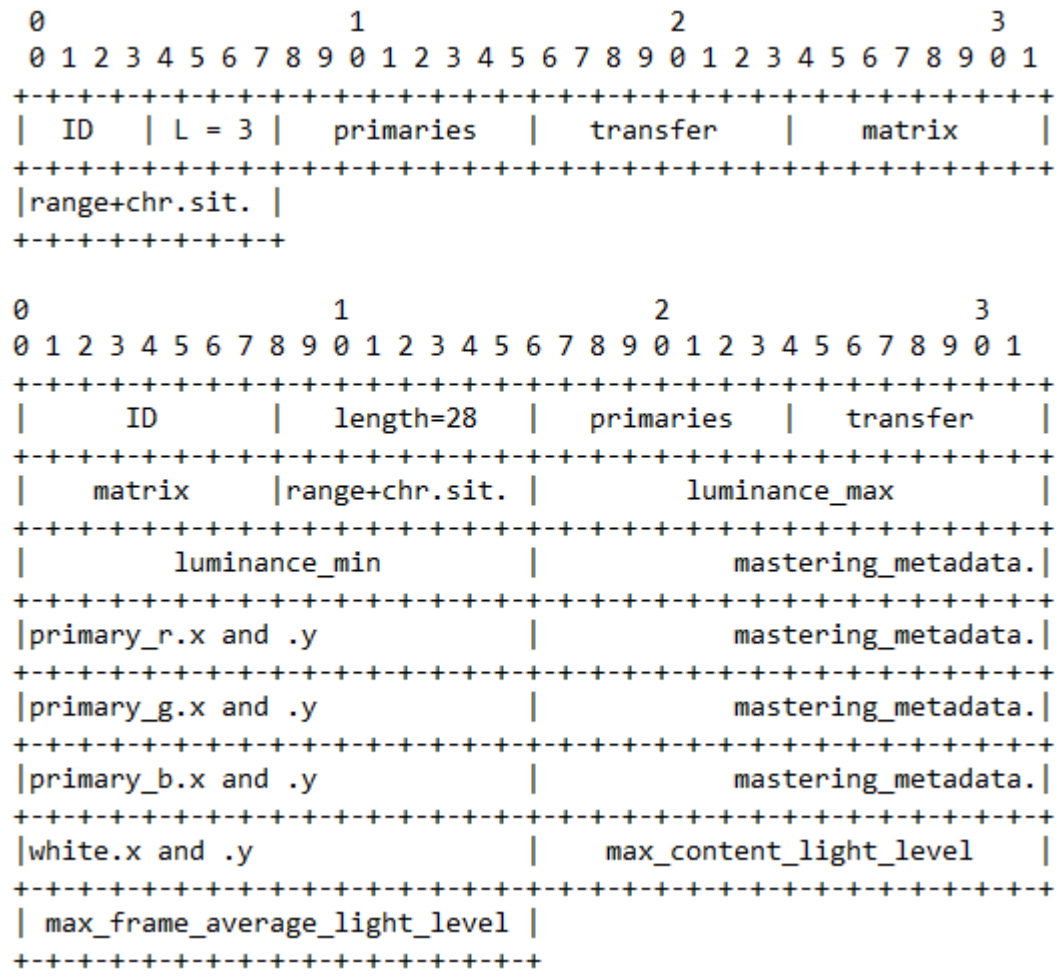


Figure 2.8: Layout of the Color Space header extension

2.8 WebRTC's custom implementation of RTCP feedback

As for the RTP header extension, some of the parameters that can be used inside WebRTC's implementation of Chrome are recent and were introduced by Google.

2.8.1 Goog-remb

The name `goog-remb` stands for Google - Receiver Estimated Maximum Bitrate. It is still an experimental addition and the documentation is under the form of an RFC draft [13]. This type of RTCP packet is meant to estimate the maximum bandwidth

available at the receiver's side. This feature is currently only implemented in the Chrome browser.

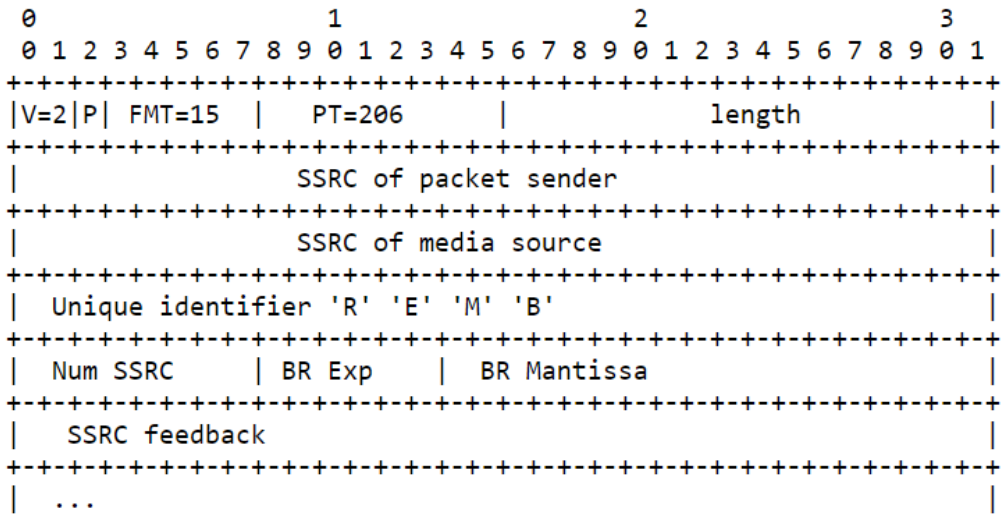


Figure 2.9: Layout of the goog-remb report

The "unique identifier" field needs to contain the 4 letters 'REMB' encoded in ASCII.

The estimated max bitrate of the receiver is transmitted via the fields 'BR Exp' and 'BR Mantissa' under the form $BR\_Mantissa * 2^{BR\_Exp}$ bits

2.8.2 transport-cc

Transport-cc is an RTCP format that is used with the RTP header extension. It is used to report for all the RTP streams in a single RTCP feedback. This is the most simple form of feedback that record precisely the timing of arrival of all the different packet report them to the sender.

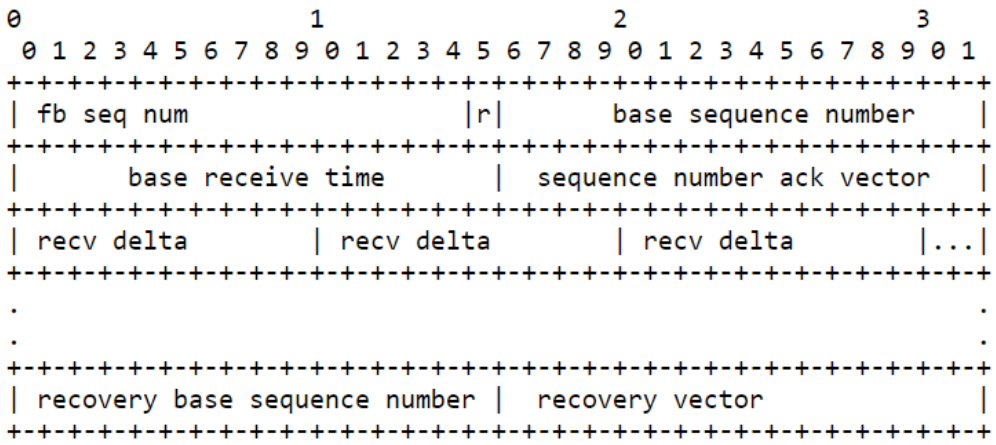


Figure 2.10: Layout of the transport-cc report

As we can see on the packet format. The core of the RTCP feedback consists of a series of received deltas. These are all the recorded times between the arrival of two packets. There are as many deltas as there are received packet since the last RTCP was sent. The amount of packets that should be monitored by each of the RTCP's is specified in the header extension of the RTP packet that made the request for a feedback [12].

Chapter 3

Methodology

In this section will describe the experimental test-bed and data collection details. It focuses on two actors of the Cloud Gaming industry, namely Geforce Now and Google Stadia.

3.1 Hardware

The platform used to collect data is a laptop containing capable hardware to ensure that the video encoding/decoding is not slowed down by the hardware. The computer relevant characteristics are the following:

CPU	AMD 5900HX
GPU	NVIDIA Geforce RTX 3070 (laptop version)
RAM	24Go
WiFi	Intel® Killer™ Wi-Fi 6 AX1650
Ethernet	Realtek PCIe FE GBE 2.5Gbps

To ensure that the video encoding/decoding was done on the discrete GPU rather than the integrated GPU of the CPU, the integrated GPU was disabled in the BIOS during the measurements collection.

3.2 Network

The internet Gateway is a combo router/modem Technicolor TC7210.V with a 400Mbps connection to the internet. During the wired testing, the computer is connected directly to the router modem using a 1 Gbit/s Ethernet interface using a 2 meter long CAT6 cable.

3.3 Software

The operating system was Windows 10 version 10.0.19043.

Both Google Stadia and Geforce Now were running via the Chrome Browser on version 91.0.4472.124 For the data collection, three different programs where used, namely Wireshark (v3.4.6), netlimiter (v4.1.10) and clumsy (v0.2)

Netlimiter is a paid Windows utility but was used under its free licensing. The free licence allows the user to set bandwidth limit to certain program independently of the network interface and other program running on the computer. This is the utility that is used to simulated bandwidth congestion.

Clumsy is an open source software that allows to emulate instability on the network. It allows the user to simulate those instabilities only on packet that correspond to a "wireshark like" style of filtering. It can simulate a added delay, packet loss, packet duplication, reordering and tampering of the packet. This is the utility that was used to simulate instability.

3.4 Scenario

For the testing, we had to keep the video moving, otherwise the measured bandwidth would drop and congestion would have no effect. Also, it's needed to keep the scenario as close as possible to an actual use case of the platform.

In order to simulate a gameplay on Geforce Now, the choosen game was Counter-Strike Global Offensive as it has a spectator feature. During the testing the game would be set to spectate a game played by other players and therefore emulate perfectly the network flow of a regular player on Geforce Now.

On Stadia, a compromise had to be made since the number of games available to a new user is very limited. Hitman 1 mission 1: "showstopper" was chosen since it has a lot of moving background for this scene. Since there are no spectator or replay feature for this game, the game was played normally during the recording of the network trace.

Both platforms had their network trace recorded for different available bandwidth, packet loss rate and delay. Since the games are not constant and can sometimes get too static or too dynamic, the recordings were all 400 sec long in order to get a decent average for each of the perturbation parameters.

3.5 Measurement Parameters

For the testing, a set of representative values had to be decided in function of the limitation of each platform. The underlying table shows the chosen value for each actor in each scenario.

	Bandwidth limit (Mbps)	Packet Loss (%))	Delaying(ms)
Geforce Now	40,30,20,10	0.5,1,2,3,5,7,10	25,50,75,100,125,150,175
Stadia	20,12	0.5,1,2,3,5,7,10	25,50,75

The chosen values are different for Nvidia and Stadia mainly because of limitation on Stadia. Since Stadia does not use more than 30 Mbps under normal use, setting a cap above 30Mbps would not change the result. Also, Stadia would simply not run with a cap set at 10Mbps. 12Mbps was the lowest value that would allow Stadia to run and was there for chosen as the lower bound for this testing scenario.

For the uniform loss percentages, this wide range of chosen values is due to the different way that Geforce Now and Stadia react to a packet loss. While Geforce reacts early and had strong reaction to even a 0.5% loss, Stadia only started to show a reaction when reaching 5% of loss. In order to get accurate measurements for both platforms and have a common methodology of testing, both platforms were tested from 0% to 10%.

Finally, for the delaying test, the main idea was to set increments of 25ms from 0 to 200ms. But Stadia stopped working as soon as the delay reached 100ms. Therefore there was no way to have any meaningful result above 75ms for Google Stadia. Those delays were added to the natural latency of the test setup which was measured at around 16ms.

Chapter 4

Protocols

This section details the custom parameters and implementation of the different protocols used by Geforce Now and Stadia.

4.1 WebRTC

Both Geforce Now and Stadia use WebRTC in their respective browser version. We can use this address inside of Chrome: "chrome://webrtc-internals/" to gather key information on the communication parameters used on WebRTC. For this trick to work, the link must be opened in a different tab before starting the WebRTC content we try to analyze. Doing this allows the capture of the SDP(Session Descriptor Protocol) which would normally be hidden in an encrypted DTLS packet and capturing it would not reveal any information about its content.

4.1.1 Geforce Now's WebRTC configuration

In the captured SDP, all the different payloads are mapped to their content. There were a lot more mapped RTP-payload actually used. All the mapped types are detailed in the table below.

As we can see in this SDP packet, there are a lot of different payloads declared for the exact same usage. From the different traces captured during the testing, only payload 96,97 and 111 were observed. This indicates that all other declared payload usage are either implemented for future usage or simply not used after being mapped. Payload 104 is supposed to be the only one linked to RED, following RFC 2198 that would normally be used for REDundant coding. But again, this packet was never recorded during the testing. Another type of packet that is declared but never used in the SDP is the payload 106, mapped to ulpfec (Uneven Level Protection Forward Error Correction). This type of packet should be used as

RTPMAP	Argument
96	H264/90000
97	RTX/90000
98	H264/90000
99	RTX/90000
100	H264/90000
101	RTX/90000
102	H264/90000
103	RTX/90000
104	RED/90000
105	RTX/90000
106	ulpfec/90000
110	telephone-event/48000
111	OPUS/48000/2

an error correction to reduce the impact of the loss of a single packet in a frame. In the SDP, we can see the RTP header extension and sometimes the link to their documentation, a screenshot of the captured SDP is shown in figure 4.1. From this, we can deduce the use of the header extensions of RTP. These are explained further in this chapter.

```
a=extmap:14 urn:ietf:params:rtp-hdrext:toffset
a=extmap:2 http://www.webrtc.org/experiments/rtp-hdrext/abs-send-time
a=extmap:13 urn:3gpp:video-orientation
a=extmap:3 http://www.ietf.org/id/draft-holmer-rmcat-transport-wide-cc-extensions-01
a=extmap:12 http://www.webrtc.org/experiments/rtp-hdrext/playout-delay
a=extmap:11 http://www.webrtc.org/experiments/rtp-hdrext/video-content-type
a=extmap:7 http://www.webrtc.org/experiments/rtp-hdrext/video-timing
a=extmap:8 http://www.webrtc.org/experiments/rtp-hdrext/color-space
a=extmap:4 urn:ietf:params:rtp-hdrext:sdes:mid
a=extmap:5 urn:ietf:params:rtp-hdrext:sdes:rtp-stream-id
a=extmap:6 urn:ietf:params:rtp-hdrext:sdes:repaired-rtp-stream-id
a=recvonly
```

Figure 4.1: Links in the captured SDP

4.1.2 Stadia's WebRTC configuration

The above table shows the RTPMap available in the SDP packet. This table is much smaller and clearer than Geforce's and shows only used payload types.

The first information we can deduce from this map is the use of the VP9 codec for video encoding. Since this is a custom addition of Chromium that is not standard in the WebRTC implementation, this could explain why Stadia only runs on Chromium-based browsers.

RTMAP	Argument
96	RTX/90000
109	VP9/90000
111	OPUS/48000

Secondly, payload type 96 is defined as re-transmission packets. It can be observed in the chapter about network instability that Stadia uses this kind of packet for more than just re-transmissions. It has also been implemented as a bandwidth probing tool.

Finally, as for Geforce Now, payload type 111 carries the audio of the game. But contrary to Geforce, this is not a two-way communication and is only used to carry the sound of the game to the client.

4.2 Startup

Both Geforce Now and Stadia have the same startup sequence. The sequence used is the classic WebRTC. They start with a STUN phase in order to prepare for the connection to the server that is going to be used as a host for the game. Then it opens the different data channels that are going to be used to communicate the inputs of the player and get the keys used for the SRTP stream. Finally, the different SRTP streams start.

The main difference between the two actors on the startup is with the initial bandwidth allocation. Geforce Now uses the user's set bandwidth cap as a reference or the default pre-encoded value. Meanwhile, Stadia does a network probing with its type-96 RTP packet as shown in figure 4.2

4.3 RTP

4.3.1 Headers

Both Nvidia and Google use the same header field in their respective RTP packet. Their parameters are highlighted in Table 4.1. There is only one difference in the main header between Nvidia and Google. It is the way the timestamp is computed. Geforce Now uses different ways to compute the timestamps in function of the media type carried over the RTP session. The timestamps are linear with the time but do not increase at the same rate. For payload type 111 (audio), the timestamp increases at a rate of 28.8/ms. Meanwhile, the timestamp of the video (payload type 96 and 97) increases at a rate of 91.8/ms.

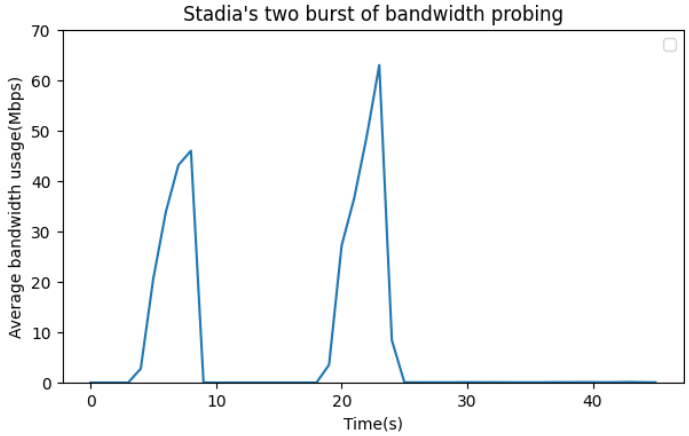


Figure 4.2: Cumulative distribution of the bandwidth usage for Geforce Now

	Value
Version	2
Padding	False
Extension	True
Contribution source identifier	0
Marker	0 or 1
Sequence Number	variable
Timestamps	variable
SSRC	fixed
Extension length	3

Table 4.1: Base value for the RTP's header

For Google Stadia, the timestamp computation for the video (payload 96 and 109) is different. Contrary to the RFC that says "one tick per video frame is typically not sufficient". It is exactly the way that Google does it, with an increment of one at each frame. For the audio (payload 111), the timestamp is computed in the same way as for Geforce Now with an increase of 28.8/ms.

4.3.2 Geforce Now's Header extension

In figure 4.3, a screenshot of the header extension is provided. On this screen, the three different header extensions of a video packet are shown. To understand what these headers mean we can use the information of fig 4.1.

The first header extension with id 2 is called "Absolute Send Time" by the WebRTC documentation. The idea of this header is to be a timestamp much closer

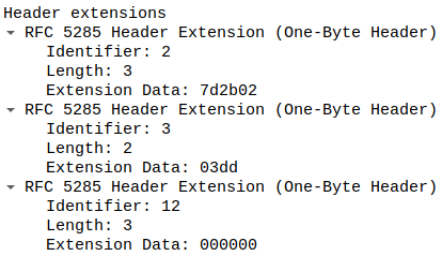


Figure 4.3: "Geforce Now header"

to the actual departure from the system. It is said in the documentation that this field should be set to its value as close as possible to the departure of the packet. This extension is also present on the audio stream and uses the same time reference as the video stream.

The second header with identifier 3, should be the "transport wide sequence number". Following the draft for the RFC linked to this feature, the main benefit of this feature would be a faster detection of packet loss and trigger a congestion control or a re-transmission faster than if it was only detected per media. This doesn't seem to be the case and the feature does not seem to have been fully implemented at this time. Even if the video packets do feature this sequence number in the header with id 2. This does not seem to be "transport wide" as the audio stream does not include a header extension with id 3 and has no other header extension that increases as a sequence number would. However, the retransmission packets with payload 97 do have this extension and they do fall in sequence with the regular video packets.

The third header extension with id 12 is a "Playout Delay". This is an instruction for the receiver that gives a maximum and minimum playout delay. The documentation of this feature in the WebRTC says that for interactive gaming, this field should be kept to 0 as the receiver is not supposed to wait or smooth out the playing but rather reduce the latency as much as possible. Sure enough, Geforce Now always sets this value to 0-0 in order to force the client to play each frame as soon as possible.

Finally, there is one last header extension that can only be found in the audio stream with id 1. Following the information in the SDP, this header extension is an indication of the audio level in the packet. The documentation of this feature indicates that its main usage is in conferences when the recipient does not need to process all audio streams but only the loudest ones. It is unclear how this feature is used in this context.

4.3.3 Stadia's Header extension

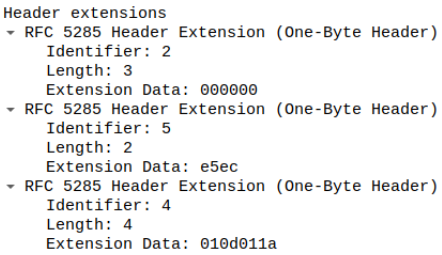


Figure 4.4: "Google Stadia header"

First of all, because the header extension id is mapped in the SDP, there is no direct link between the id number of Geforce Now and Stadia.

For Stadia, the header with id 2 is the "playout delay". As for Geforce Now and as it should be for all interactive gaming, this value is always set to 0.

The second header extension with id 5 is an extension to the "transport wide sequence number" already seen in Geforce Now. This version allows for a longer than "one-byte extension". This extension adds the optional fields "T", media's seq number, and the count of seq number that should be included in the feedback packet. However, this is implemented exactly as the first version of the extension, and only the transport-wide congestion number is provided. As for Geforce Now, the audio stream does not use this extension but the re-transmission stream with payload 96 does.

The third header with id 4 is the color space extension. This value is fixed, even across sessions and games. This header allows the client to render the video stream as HDR content. It should be noted that this header is present on all frames and is present on all video packets. Meanwhile, the documentation of this feature mention that this header should be ignored on all packet expect the last on of a frame.

Finally, no header extension is present at all on the audio packet.

4.4 RTCP

The above table shows the argument passed to webRTC to generate the RTCP feedback for Stadia. The exact same table is used for Geforce Now with obviously different payload numbers. There are a few remarks to be made here.

First of all the first line of the table is the only line referring to the payload type 111 (audio). This means that the feedback of the audio is only based on the transport-wide sequence number. Since the audio does not have a header extension

Command	Argument
rtcp-fb:111	transport-cc
rtcp-fb:109	nack
rtcp-fb:109	nack pli
rtcp-fb:109	transport-cc
rtcp-fb:109	ccm fir
rtcp-fb:109	goog-remb

with the transport-wide sequence number. It means that the feedback for the audio is only based on the video transmission.

Then there are two lines that fit together, `nack` and `nack pli`. This means that negative acknowledgment will be used to generate the RTCP report. Only missing packets should be reported. And PLI means picture loss information, a report should be generated as soon as the decoder is missing a frame for any reason.

The next extension is `transport-cc`, this extension simply means that the time should be recorded at each received frame and reported as one RTCP packet when requested by the corresponding RTP header extension.

The following argument is `ccm fir`. This stands for "codec control message" and fir for "full intra-frame". This means that at any moment the decoder can request a full intra-frame. Usually, because it lost a frame previously and needs to reconstruct the stream of frames from a new base.

Finally, we see `goog-remb`. This is a special RTCP that tries to estimate the maximum bitrate on the receiver side. More details about this extension are provided in the State of the art section.

4.4.1 Nvidia

RTP Payload-96

From the RTPMAP provided in the beginning of this section, it can be found that the type linked to this payload is H264. This indicates that this payload is used to carry the video. When recording the trace, this payload type appears in bursts of packets sent with the same timestamp and with the last packet of each sequence having is marker bit set to 1. The amount of marked packets received per second is equal to 60. Since the frame rate set inside of Geforce Now's setting is 60 per second, this result indicates that each of these bursts and their marked packet compose a single frame.

RTP Payload-97

From the RTPMAP found in the beginning of this section, it can be found that the type linked to this payload is H264. This is a classic WebRTC format to indicate a re-transmission channel that appears to be linked with payload-96. During regular testing with no packet loss, this payload is never observed and is only sent in the event of a packet loss.

RTP Payload-111

As it is observed in the RTPMAP, the codec assigned to this payload is OPUS. This payload type is therefore used to carry the audio. It is interesting to note that this is the only bi-directional RTP communication. From the server to the client direction, we can observe a constant stream of 50 packets per second carrying the sound of the played game. For the other direction, from the client to the server, these packets are only sent if the user has enabled the microphone access in the browser. Other than that, this stream is very similar with the same bandwidth usage and the same amount of packet per second.

4.4.2 Google

RTP Payload-111

Following our RTPMAP decoded earlier, this payload type is used for audio communication. The main difference with Geforce Now is that this payload is used only from the server to the client. There is no microphone recording happening in Stadia. The parameter is also exactly the same as Geforce Now, with a bitrate of 110Kbps and a packet rate of 50/sec.

RTP Payload-109

For the video, it is transmitted through payload 109 under the VP9 codec. We can clearly see the 60 frames per second that the client display as there is 60 marked RTP packet sent per second alongside a burst of RTP packet. The size of a single packet inside a burst is generally around 1200 bytes. The size of the burst depends on the Stadia quality configuration, the network stability, and the available bandwidth as it will be shown in the section about network instability.

RTP Payload-96

Type-96 is the re-transmission channel. It is almost not used in a perfect scenario. We can see these packets use only 0.03Mbps of bandwidth and the packet rate

oscillate between 20 and 50 packets per second. From the observation of shown in the network instability section, these packet seems to have a role in the network probing and congestion control throughout the use of the platform and on the startup.

4.5 Mark

Both Geforce Now and Stadia set the RTP mark bit to 1 on the last packet of a frame. This is how the application knows when it has received a full frame that can be displayed. This behavior becomes obvious when we decide to plot the amount of marked packet received per second as shown in figure 4.5. We can see a constant 60 packets per second for the whole transmission. This is coherent because the user setting about the framerate was set to 60 inside Geforce Now.

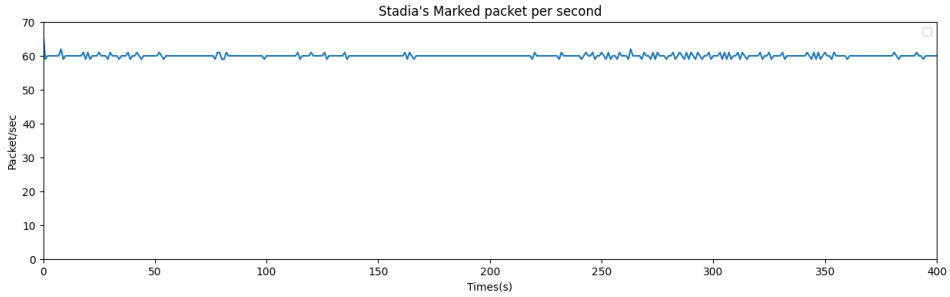


Figure 4.5: "Number of marked packet per second on Stadia"

4.6 Conclusion

In conclusion, both Geforce Now and Stadia are similar in terms of their WebRTC implementation. Both actors use 3 RTP streams, one video, one audio, and one for re-transmission or error correction. They both chose to use identical Feedback methods in their WebRTC configuration. Where they differ is in the header extensions used for their different streams.

Geforce's implementation of WebRTC seems to be a work in progress as many different RTP streams are opened but only a few are actually used. This is probably due to the fact that Nvidia advertises their support for chrome as in the Beta stage. There is a lot of features that could improve the user experience that seems to be instantiated in WebRTC but not implemented in the RTP traffic itself. There are also a few features that do not fit the use case are present anyway, as the audio

level in the audio RTP stream. This feature is aimed at conferences with multiple audio streams and it is unclear why Geforce has this extension enabled.

On the other hand, both actors make use of recent experimental WebRTC features. Some of which seem to be tailored to make cloud gaming possible inside WebRTC. Most of these experimental features are actually presented by Google in a time frame close to Stadia's public release.

Chapter 5

Network instability

This section will analyze how both Geforce Now and Stadia reacts to the different perturbations mentionned in the methodology. The results of these tests appear in the form of recorded network traces that were analyzed and generally represented as temporal bandwidth usage and cumulative distribution function (CDF). The goal of the CDF's is to give us a general idea of the statistical distribution of the bandwidth usage and how it is affected by the perturbation. Meanwhile the temporal distributions offer a visual way to see a transition or a repeating pattern. With both of these tools, this section will try to decipher and explain the different triggers and mechanisms put in place by both Nvidia and Google in their respective platform to handle network instability on the user side.

This section is the result of setting up and recording network traces for more than two days. With a total of 50Gb for over 5 hours of actual recorded data separated into 400sec segments.

5.1 Near ideal network condition

The ideal network recommended for Geforce Now is a network with a 50Mbps bandwidth, less than 0.5% packet loss, and a latency to their server under 20ms. Stadia does not have requirements but only a test tool that the user can run. That test tool does not give any detail and will always return a positive result if the available bandwidth is greater than 35Mbps. This makes our test network a good fit for testing both platforms under their ideal condition

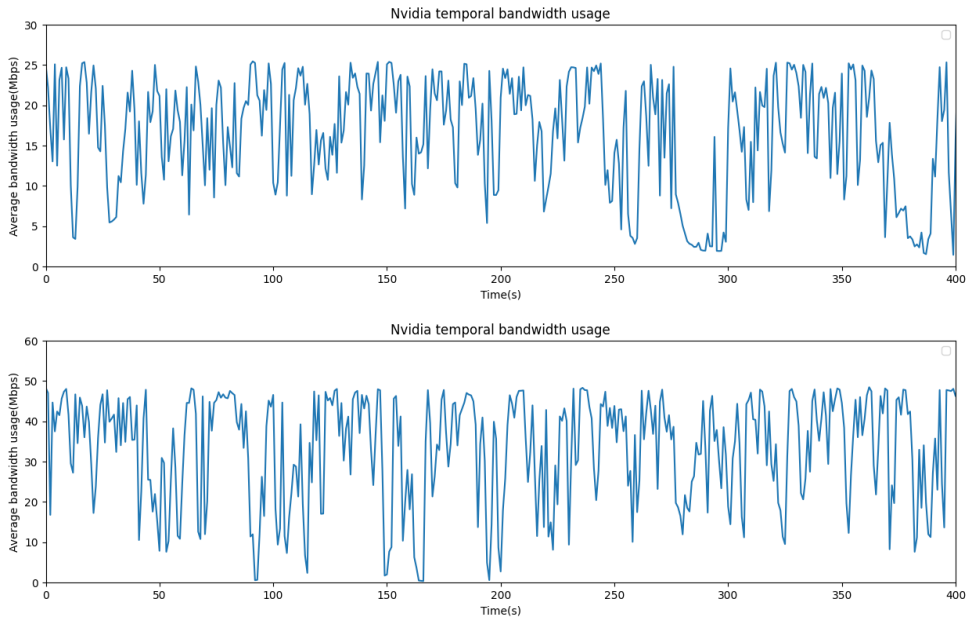


Figure 5.1: Temporal representation of the bandwidth usage for Geforce Now

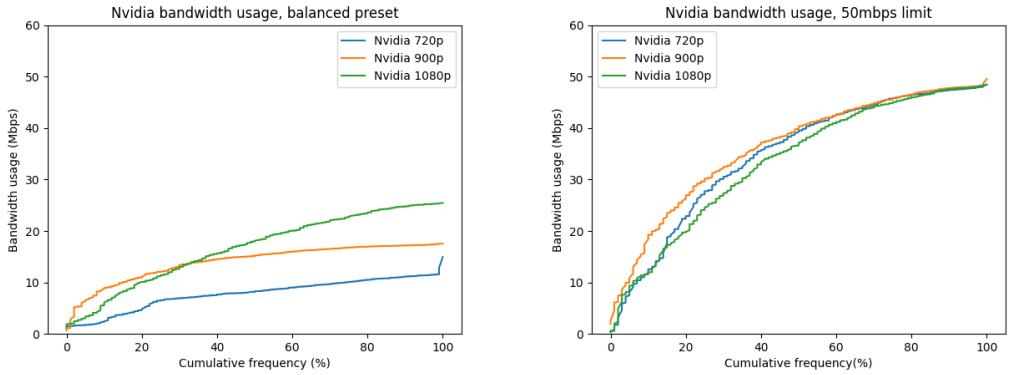


Figure 5.2: Cumulative distribution of the bandwidth usage for Geforce Now

5.1.1 Geforce Now under ideal conditions

Figure 5.2 shows the cumulative distribution of the bandwidth usage summed per second over 400 seconds. The detailed scenario used here is provided in the methodology section. It consists of a recording of a game of Counter-Strike Global Offensive in spectator mode. As we can see on the left of figure 5.2, the maximum bandwidth automatically set by the platform is dependant on the resolution chosen by the user. The lower the resolution is set, the lower the automated cap is set by

the program. When the user switch from "auto" to "custom" for the bandwidth limit, the previous limit appears, showing us that the limits were 12,18 and 27mbps for 720p, 9900p, and 1080p respectively. This result is coherent with the measurements shown on the left part of figure 5.2.

In comparison, the right part of Figure 5.2 shows the same test but with the bandwidth cap manually set to 50mbps at all resolutions. As we can see, the Geforce algorithm tries to use as much as possible of the available bandwidth.

To expand a little more on the optional bandwidth limit, Figure 5.1 shows us the temporal variation of the bandwidth usage between the automatic mode and the 50 Mbps manual limit. Keep in mind that the scale of the y axis of both of these graphs is not the same and the actual bandwidth usage is much lower for the automatic mode. Do not forget that both of these recordings are for the same game and same scene but were played by a different player and therefore the video transmitted is not exactly the same. Therefore we cannot compare their shapes in a one-on-one comparison. We can only observe the amount and overall shape of the variation. Both of which are very similar. From both the CDFs and the temporal representation of the bandwidth usage, we can see that a bandwidth cap does not change the overall working of the compression algorithm.

5.1.2 Stadia under ideal conditions

On the Stadia side of things, there is no slider to manually set a bandwidth cap, we are only left with the possibility to change the resolution. The temporal representation of the bandwidth usage in figure 5.3 tells a similar story to Geforce. Even if we don't seem to have the same hard cap on the bandwidth usage, we see the same kind of variation. On the bottom part of Figure 5.3 we can see Stadia's bandwidth cumulative distribution. It's interesting to notice that those CDFs have a similar shape to the ones found on Geforce's side.

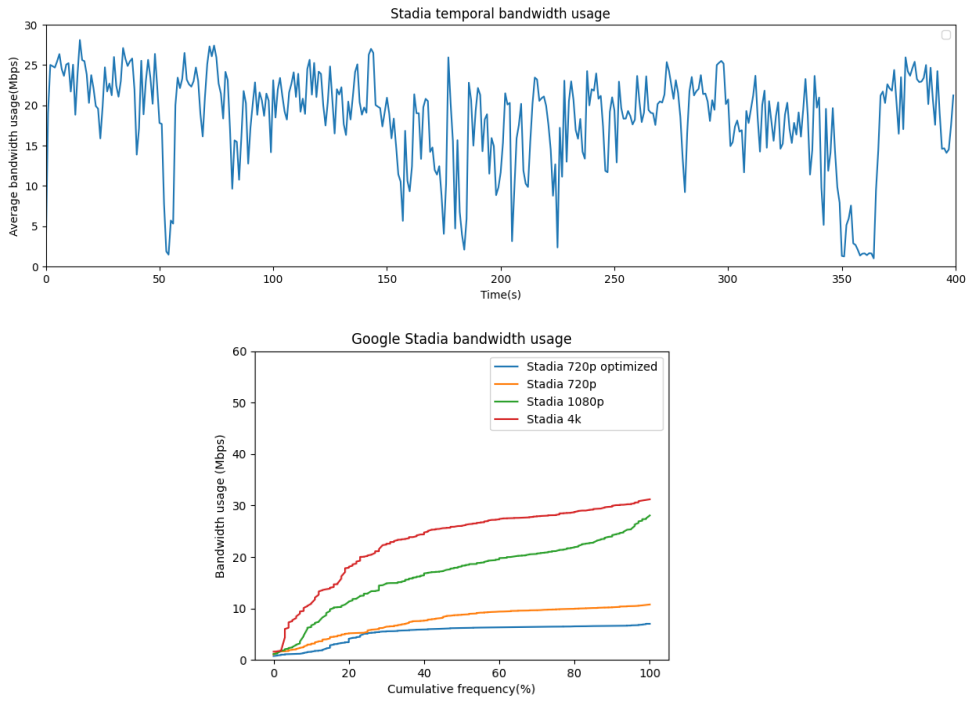


Figure 5.3: Representation of the bandwidth usage for Stadia

5.2 Reduced bandwidth

This section will analyze the way Geforce Now and Stadia react to reduced available bandwidth. First, it will look at how these two platforms react to a static reduced bandwidth and then it will analyze the short-term reaction and the transition period in case of a sudden change.

5.2.1 Geforce now with Bandwidth reduction

For this test, the bandwidth setting inside the Geforce Now interface was set to 50Mbps and all measurements are 400sec long. On figure Figure 5.4, we can make the first observation with the CDF representation of all measures. All recorded traces follow the same type of curvature. This means that there is some adaptation of the platform to the bandwidth perturbation. If there were no adaptation since this is a UDP communication, we would expect to see the same start of the curve but the rest would stay capped at the limit set during the test.

Figure 5.5 shows us how the congestion control really works. These two recordings were made using a slightly modified test to make the video more constant with a less static image in order to get to the most stable representation

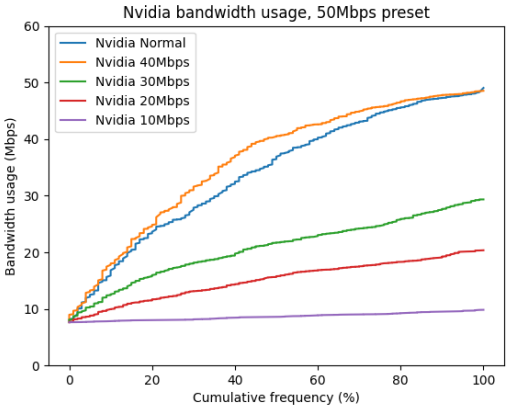


Figure 5.4: Geforce Now bandwidth usage with limited available bandwidth

of the video traffic. In Figure 5.5, We can see that the congestion control works very similar to the regular TCP's congestion control algorithm.

There are three different states, the first one is the steady-state. In this mode, the bandwidth requirements are satisfied and Geforce doesn't need to adapt to a network condition. In this mode, the bandwidth is only capped by the internal bandwidth limit that the user set in the option and cannot be set higher than 50Mbps. The second mode is the small congestion, In this mode, only a single or a few packets were lost due to the congestion. To avoid further loss, the algorithm temporarily reduces the internal bandwidth limit of the application to around 60% of the previous cap and starts to slowly increase the bandwidth linearly.

In the second part of the Figure 5.5 we can see that increase being around 4Kbps/s. The first mode is in case of severe congestion, that is a loss of more than a few packets. We can observe a reaction close to the "congestion avoidance" of TCP. The internal bandwidth cap increases linearly until it reaches new congestion or the user's set limit in the application.

On the third part of Figure 5.5. We can clearly observe this behaviour with the first part before the red line is the part with a bandwidth cap of 5Mbps and the second part after the red line being totally throttled. The red line highlighting the exponential effect was obtained via regression of the different highest points of the data line. The chosen regression was an exponential regression as it had the lowest average relative error and the highest correlation coefficient. The result of this regression yields this black line with equation $e^{-6.4667+0.0625*x}$.

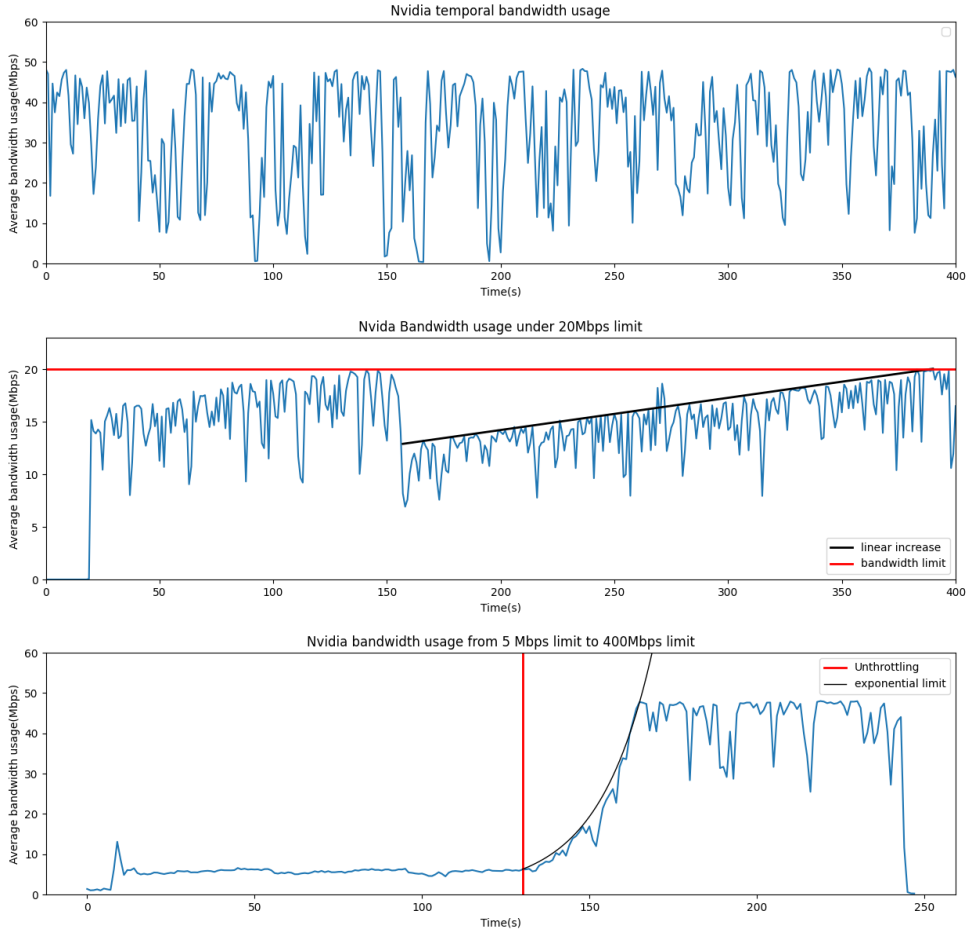


Figure 5.5: Temporal representation of the bandwidth usage for Geforce Now

5.2.2 Stadia with Bandwidth reduction

For this test, the 1080p preset was selected in the parameters of the web interface. Since Stadia won't let the game run if the bandwidth available is too low, the minimum bandwidth that allowed this test to complete was 12Mbps and therefore in the lowest measurement in this testing. It's also worth noting that the baseline does not exceed the 30Mbps limit and therefore the test with a 30Mbps limiter yielded the same result as the baseline and is not included in this section. All the recordings were 400sec long.

First, as for Geforce Now, we can analyze the resulting CDF of the bandwidth usage. This result is available in the Figure 5.6. The most interesting line on this graph is the 20Mbps line. We can see in the first part that the median and average value stay well under the 20Mbps mark. Furthermore, there is an irregular spike

at the end of the CDF that seems to indicate a height bandwidth usage during only 17% percent of the time. This result will be further analyzed in the temporal representation of this measurement

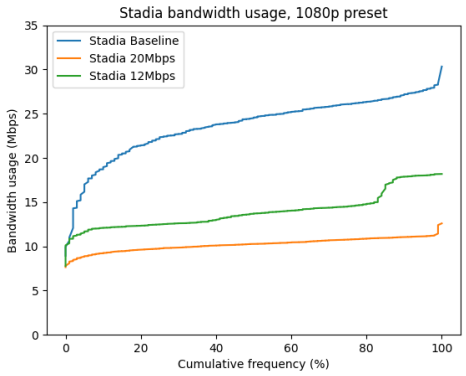


Figure 5.6: CDF representation of the reaction of Google Stadia to different bandwidth limit

In Figure 5.9 we can observe the temporal bandwidth usage of Stadia under a 20Mbps limiter. The first graph shows use the total bandwidth, the second one shows us only the bandwidth usage of the RTP packet with type 96, and the third one shows the bandwidth usage of all the other packets than those with type 96.

From this result, it appears that under certain conditions, Stadia will send bursts of these type 96 packets every 35 seconds on average. These packets are not correlated with the bandwidth usage of the video and the sound, which remained constant throughout the testing as seen in the third part of Figure 5.9. The spike seen on the second graph just before the 200th-second mark is higher than all other spikes. This can be explained by a drop in video bandwidth usage. This drop is most likely due to a more static scene in the game which required fewer video data to be sent to the user. This goes to show that the behavior of these type-96 packets is to use as much as possible of the available bandwidth and are probably send in a much higher quantity only to be blocked by the limiter. This will be further analyzed in the loss packet testing.

Although it is not shown in any figure, the type-96 bandwidth usage for the baseline never goes above 0.03Mbps. This shows us that these types of packet bursts are only used in case of network instability.

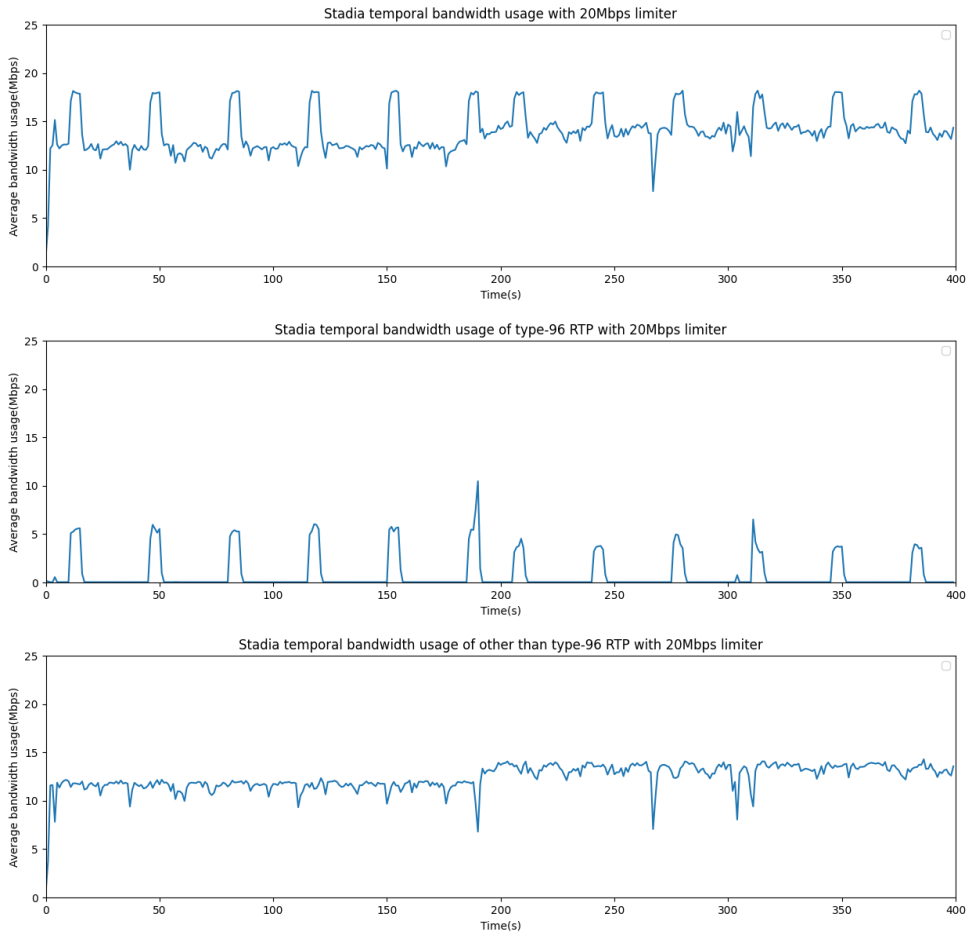


Figure 5.7: Temporal representation of the bandwidth usage for Stadia with a 20Mbps limiter

5.3 Uniform Loss

This section will analyze the way Geforce Now and Stadia react to a continuous loss of a certain percentage of packets.

5.3.1 Nvidia with constant uniform loss

Figure 5.8 shows us the big picture of Geforce Now reacting to a uniform loss of packets. It's pretty clear that under any condition of a constant loss, the service is not functional. The video quality degradation from the congestion control algorithm renders the service unusable due to a combination of stuttering and visual quality loss. On this figure, only the scenario with no loss yields a usable experience while

all other loss scenarios, even with only 0.5% of packet loss made this congestion control algorithm drop to 5Mbps. This indicates that the primary indicator that the congestion control algorithm used by Geforce Now is the packet loss and any loss packet is triggering a reduction in bandwidth. This is coherent with earlier observation during the limited bandwidth test where the congestion control would trigger only when a packet was lost due to being too close to the bandwidth limit.

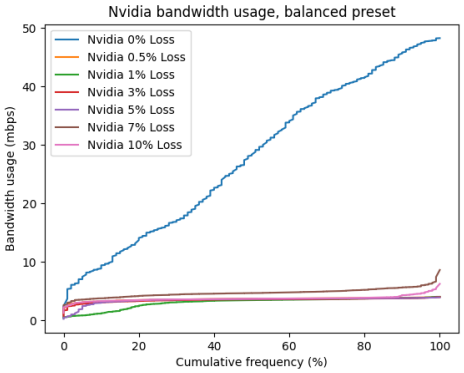


Figure 5.8: CDF of bandwidth usage of Geforce Now under uniform loss

5.3.2 Stadia with constant uniform loss

Figure 5.10 shows us the CDFs of the bandwidth usage of Stadia for different packet loss rates. It's interesting to notice that contrary to Geforce Now, the bandwidth usage doesn't collapse until a loss of around 7%. A few things are to be noted here. First of all, the CDFs of all measurements under 7% of loss are similar. This indicates that the congestion control algorithm of Stadia is not directly triggered by packet loss. What's more interesting is that the 7% and 10% curves end up with a significant increase in bandwidth for around 10% of the traffic. Contrary to what we could believe, this is not a redundancy of information in order to predict and replace the loss of packets. As we will see in the temporal analysis, this has to do with the same phenomena as in the bandwidth restriction.

Ultimately, by comparing the reaction to a restricted bandwidth and a loss of packet we can deduce that any loss of a packet is going to trigger a probing response using a different type of RTP packet. These probes are going to be used by the congestion control algorithm to determine the best bandwidth cap. This behavior has its advantages compared to the behaviors of Geforce Now. This allows the game to keep playing at a playable level with a minor drop in image quality and stuttering with a low amount of packet loss.

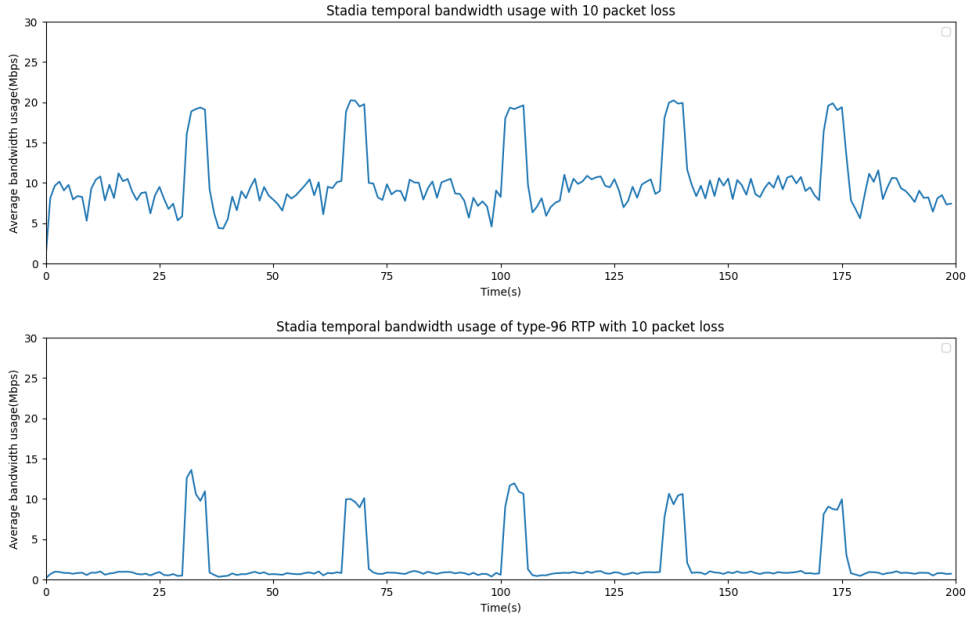


Figure 5.9: Temporal representation of the bandwidth usage for Stadia with a 20Mbps limiter

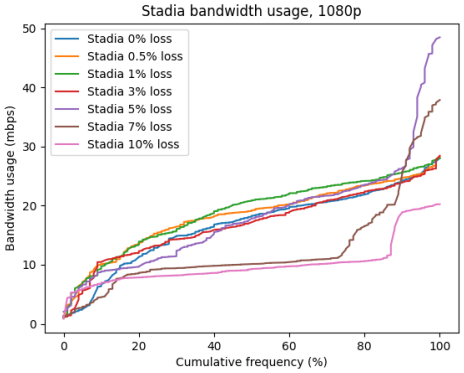


Figure 5.10: CDF of bandwidth usage of Stadia under constant packet loss

5.4 Delay

This section will analyze the way Geforce Now and Stadia react to a network with a higher delay.

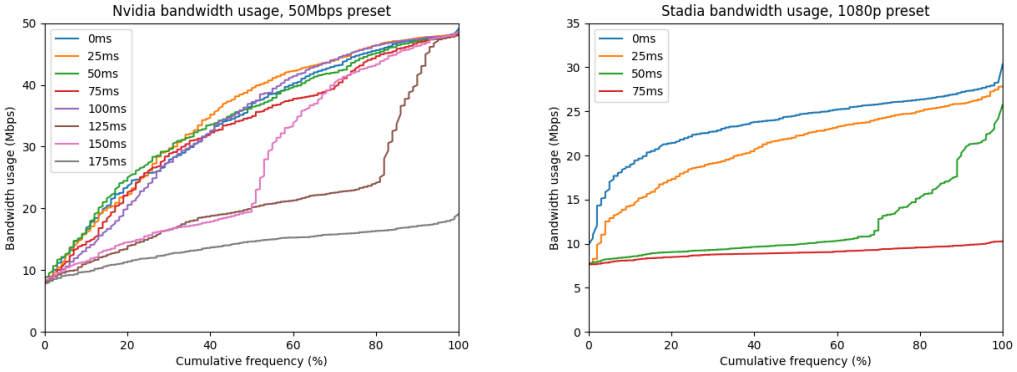


Figure 5.11: CDF of bandwidth usage for Geforce Now and Stadia under added latency

5.4.1 Geforce Now with a constant delay

First of all, Geforce Now is supposed to have a maximum requirement of 50ms of latency. During this testing, the actual network latency will be tested with the base latency of our test setting and added latency. The base latency is 16ms, for the sake of clarity, only the added latency will be shown in the legend of the figures.

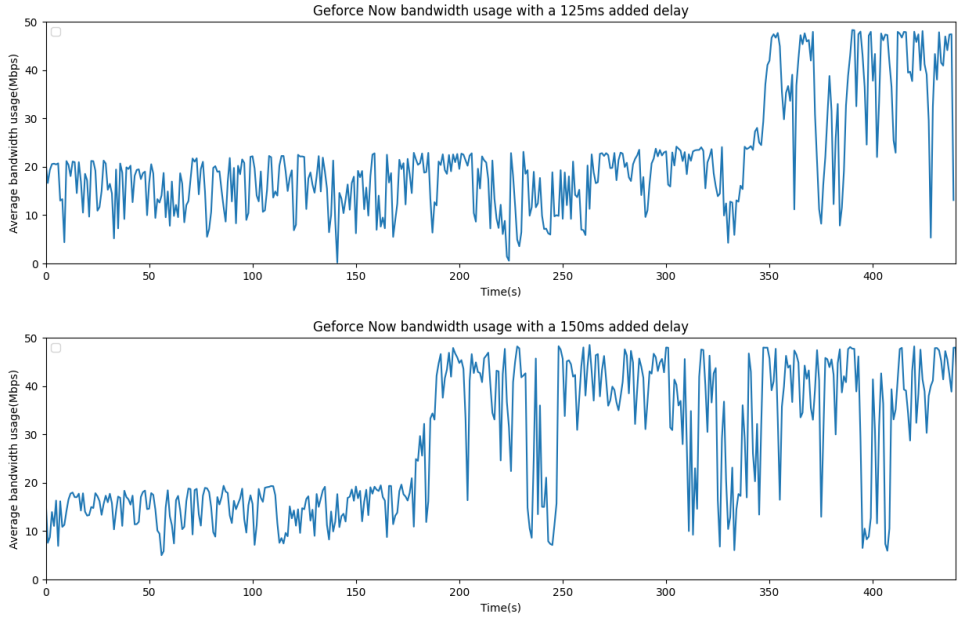


Figure 5.12: CDF of bandwidth usage for Geforce Now and Stadia under added latency

The left part of Figure 5.11 shows us the reaction of Geforce Now under an added latency between 25ms and 175ms. This figure shows that the added delay did not change the way the platform works for values under 125ms. For 125ms and 150ms, we see a curve that takes a steep increase midway through the graph. This is an indicator of a radical change during the process which is much clearer to see on the temporal representation of Figure 5.12.
To better understand the temporal representation of 125ms and 150ms of delay of Geforce Now, we need to keep in mind that these tests were done back to back and therefore they are a transient state that reaches stabilization at the end of the recording. These graphs show a variation in the internal bandwidth cap of the application. When Geforce first notices the increase in RTT, its first reaction is to lower the bandwidth cap. After some time of this increase latency, an exponential bandwidth increase kicks in and restores the bandwidth cap to its maximum. This is most likely due to the algorithm favoring the latency to the bandwidth usage in order to ensure the best user experience. The algorithm needs to make sure the increased latency is not due to bandwidth congestion. This reaction also explains the "steep curves" in our CDF representation.

5.4.2 Stadia with a constant delay

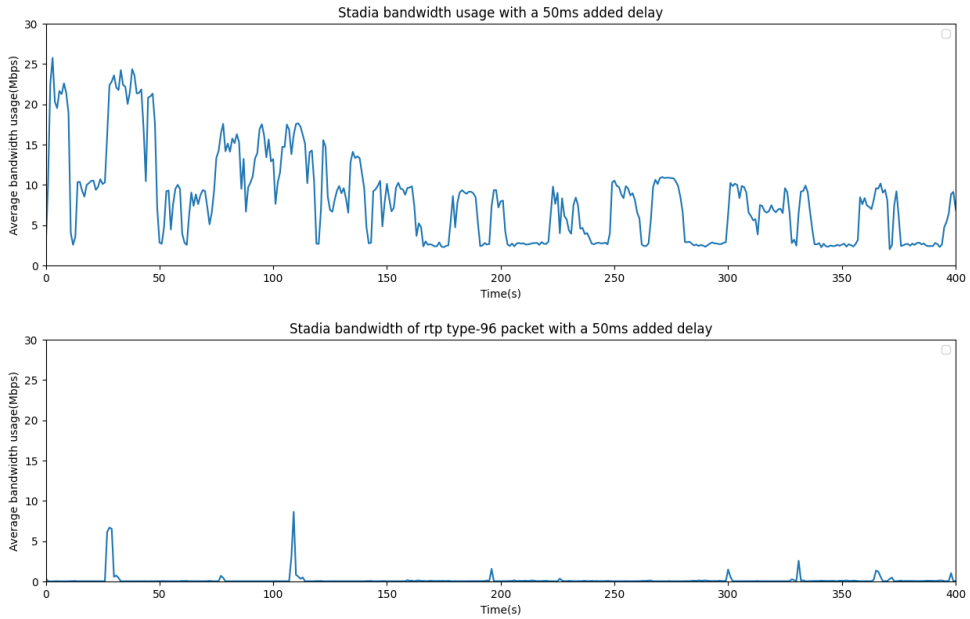


Figure 5.13: CDF of bandwidth usage for Geforce Now and Stadia under added latency

Stadia tells a different story in this test. First of all, Stadia did not run at all with a 100ms added delay. The application decided that the connection was "not good enough" and shut down the game. This indicates a hard limit for all users with more than 100ms of latency to the closest google server.

From the right part of Figure 5.11, we can see that the 25ms test did not really change the way Stadia works. Meanwhile, the 50ms and 75ms significantly decreased the used bandwidth and with that the overall image quality to the point that the game became unplayable. It's interesting to note that this test did not trigger the RTP type-96 probing that we could observe in the packet loss test.

Figure 5.13 shows the temporal representation for the 50ms delay test. We can see the opposite reaction to Geforce Now with a steady decrease in the used bandwidth. This decrease does not seem to be the result of the probing being limited as the full bandwidth was still available. We can also see that the probing mechanic was small to non-existant in the second part of the graph. This indicates that Stadia does also react only to the RTT rather than the direct loss.

5.5 Conclusion of network instability

5.5.1 Geforce

Geforce Now is very "packet loss oriented" and tries to minimize at all costs the amount of lost packet in the transmission between the server and the client. This first shows in the restricted bandwidth test where it applies a very classical congestion control algorithm to stay as much as possible beneath the bandwidth limit that the user has. This is very apparent in the packet loss test, even if the impact of a 0.5% loss is small and that this remains inside of the Geforce Now requirement, the congestion control algorithm respond strongly to this loss and make the experience unplayable for the user due to the massive drop in image quality. Where Geforce's algorithm is actually good, is with the delay of the network. Their algorithm can make the difference between a sudden change of delay due to congestion and the added delay due to actual network restriction. This makes the platform usable for non-latency-sensitive games like strategy games even with high network latency. Although it should be kept in mind that even a 50ms latency is perceptible in fast-paced games like first-person shooters and racing games.

5.5.2 Stadia

Stadia is very "latency" oriented, this approach tries to minimize latency between the server and the user by reducing the bandwidth usage. This approach has

its advantages and is inconvenients as we saw with the delay test. The main advantage of this approach is that the game remains playable over small packet loss. Up to 3% of packet loss did not hurt considerably the user experience. This approach is best for some low-quality 5Ghz WiFi that could result in some packet drop without triggering an aggressive congestion control. This approach also helps in scenarios where the WiFi access point has to serve other users and the loading of a web browser page on another computer could result in some short-term perturbation. Meanwhile, this approach doesn't work at all for people with a high latency connection. But again, Cloud gaming is a service designed from the ground up with low latency in mind and won't deliver a playable experience on a high latency network anyway.

5.5.3 Comparison and conclusion of both actors

Overall, Nvidia has chosen a very aggressive algorithm that tries to react to any small instability, even those that fit within the requirement stated on the Geforce Now website. Meanwhile, Stadia uses a more subtle approach, closer to the real-world scenario. Only a consistent perturbation can trigger a response and their probing mechanic allows them to have a better understanding of the client's network connection and its capabilities. Even if the game might lose a few frames, and stutter a little bit, the experience is not worsened for a few minutes like on Geforce Now. This yields an overall better user experience.

Chapter 6

Conclusion

The recent arrival of the GaaS (gaming as a service) paradigm on the market over the last few years seems to have introduced some new features and ideas. The idea of GaaS attracted some of the biggest actors of the sector that already had a wide enough cloud infrastructure and wanted to take their part of the cake in this new market segment. With the recent and unpredicted rise of the price of electronics, this new paradigm made its place in the gaming scene.

Even if there were attempts at this technology before, it's only in 2018 that it really became accessible to the general public with the release of most of the platforms that are available today. It is around that time that Google got involved with Stadia, its own cloud gaming platform. They quickly figured out a way to make it work using reliable protocols that were already well established. As we have shown in the analysis of the WebRTC parameter, they even added some custom features to make them better for their use case. They tweaked some aspects of WebRTC and even proposed drafts for new standard extensions of the WebRTC and RTP protocols. With these new experimental extensions implemented in their own Chromium browser, they were able to release Stadia to the general public.

Meanwhile, Nvidia had already developed their own cloud gaming service with their own program but its use was limited on some of its own hardware and later some Windows environments. It's only about a year after the Stadia was released that they decide to make use of the new features that the Chrome browser had to offer. By the analysis of the SDP done in section about the protocols, we have show that their implementation of WebRTC seems to be rather anarchic for now but it has proven to be effective and to work.

Finally, with their fully working platforms. Google and Nvidia still needed to address one of the major issues that had made cloud gaming impossible until now, the end-user network connection: most homes are linked to the internet through consumer-grade hardware and low bandwidth compared to most enterprise networks. Both platforms needed to find ways to deal with network instabilities.

We have analysed in chapter 5 how they both tackled the problem. We have shown by the different testing that Nvidia tried to limit as much as possible the amount of packet loss by using a strong bandwidth cap that could change at any time if any imperfection was detected. This approach is aimed at reducing the video quality in order to keep a perfectly smooth animation and avoid stuttering. On the other hand, our analysis of Stadia has shown that Google has chosen a different approach. They tried to make their service work and be enjoyable even if there 1is some loss on the network. It first try to get a general idea of the network capability of the client and use this information to shape the perfect bandwidth usage. They added a few algorithms to keep gathering information about the client network during the game in search of a sudden change. This allows them to work better on less reliable networks even if a few frames are lost during the playout.

As both solutions have their own advantages and inconvenient, it is up to the user to choose the service that works best for him. The rise of GaaS might see some interesting development in the future as Microsoft and Sony get more involved. Whether the end-user gets attracted to this remains to be seen. But after realising how well it actually works, there might be some hope that it gets its own niche market.

Bibliography

[1] Andrea Di Domenico, Gianluca Perna, Martino Trevisan, Luca Vassio, Danilo Giordano *A network analysis on cloud gaming: Stadia, GeForce Now and PSNow.*

[2] *J. Costeux, F. Guyard and A. Bustos, "QRP08-5: Detection and Comparison of RTP and Skype Traffic and Performance," IEEE Globecom 2006, 2006, pp. 1-5, doi: 10.1109/GLOCOM.2006.462..*

[3] *Ron Frederick, Stephen L. Casner, Van Jacobson, Henning Schulzrinne. (1996). RTP: A Transport Protocol for Real-Time Applications.* https://arxiv.org/abs/2012.06774

[4] *Henning Schulzrinne, Stephen L. Casner, Ron Frederick, Van Jacobson. (2003). RTP: A Transport Protocol for Real-Time Applications.*

[5] *Playout Delay : https://webrtc.googlesource.com/src/+/refs/heads/main/docs/native-code/rtp-hdrext/playout-delay*

[6] *Googles webrtc features, https://www.chromestatus.com/featureswebrtc*

[7] *Pike, R. (2021, March 4). History of VoiP Conversations| Cloud Vision Online. Cloud Vision Technology. https://cloudvisiononline.com/the-history-of-voip/*

[8] *Philip Matthews, Jonathan Rosenberg, Dan Wing, Rohan Mahy. (2008). RFC 5389, Session Traversal Utilities for NAT (STUN).*

[9] *Tirumaleswar Reddy.K, Alan Johnston, Philip Matthews, Jonathan Rosenberg. (2020). RFC 8656: Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN).*

[10] *Jonathan Rosenberg. (2010). Interactive Connectivity Establishment (ICE): RFC 5245: A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols.* https://www.w3.org/TR/webrtc/

[11] *Playout Delay. https://webrtc.googlesource.com/src/+/refs/heads/main/docs/native-code/rtp-hdrext/playout-delay/README.md. Accessed 18 Aug. 2021.*

[12] *Transport-Wide Congestion Control. https://webrtc.googlesource.com/src/+/refs/heads/main/docs/native-code/rtp-hdrext/transport-wide-cc-02/README.md. Accessed 18 Aug. 2021.*

[13] *Draft-Alvestrand-Rmcat-Remb-03. https://datatracker.ietf.org/doc/html/draft-alvestrand-rmcat-remb-03. Accessed 18 Aug. 2021.*

[14] *Absolute Send Time. https://webrtc.googlesource.com/src/+/refs/heads/main/docs/native-code/rtp-hdrext/abs-send-time/README.md. Accessed 18 Aug. 2021.*

[15] *Color-Space - Src - Git at Google. https://webrtc.googlesource.com/src/+/refs/heads/main/docs/native-code/rtp-hdrext/color-space. Accessed 20 Aug. 2021.*

[16] *End-to-End Guidelines for UHD Phase A Implementation – Ultra HD Forum. https://ultrahdforum.org/phasea-guidelines-description/. Accessed 20 Aug. 2021.*

[17] *'Game Stack - Project XCloud'. Game Stack, https://developer.microsoft.com/en-us/games/products/project-xcloud/. Accessed 21 Aug. 2021.*

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**