Comparative Analysis and Preliminary Demo of WebRTC-based Streaming Approaches in Game Engines

Week | Special Task

Assignment

Description

To Dos

- [ ] Figure out how to make the IP accessible outside the network
- [ ] Search about security of web RTC

Definitions

✅ **WebRTC (Web Real-Time Communication)** is a technology that enables real-time communication between web browsers and other applications. It is peer-to-peer communication, where clients establish direct connections with each other without the need for a centralized server.

✅ A **serious game** or **applied game** is a game designed for a primary purpose other than pure entertainment. The "serious" adjective is generally prepended to refer to video games used by industries like defense, education, scientific exploration, health care, emergency management, city planning, engineering, politics and art.

✅ Latency is **the time it takes for data to pass from one point on a network to another**.

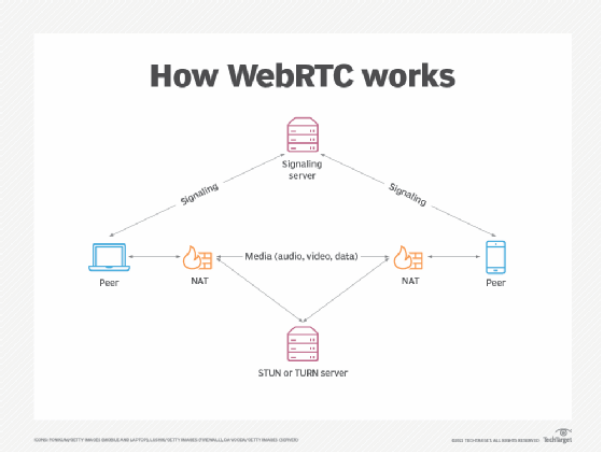
✅ Pixel is a minute area of illumination on a display screen, one of many from which an image is composed.

✅ **Signaling** is the process by which clients exchange information about how to set up the connection.

✅ **Bandwidth** is the data transfer capacity of a network in bits per second (Bps).

How does WebRTC work?

WebRTC uses JavaScript, APIs and Hypertext markup language to embed communications technologies within web browsers. It is designed to make audio, video and data communication between browsers user-friendly and easy to implement. Low latency (proper protocol udp), simplify its use.
Before audio and video files are sent, they must be compressed due to their large size. Also, media that is received over a peer connection must be decompressed. WebRTC uses a codec process to do this.



- A wants to connect to B
- A finds out all possible ways the public can connect to it
- B finds out all possible ways the public can connect to it
- A and B signal this session information via other means
  - WhatsApp, QR, Tweet, WebSockets, HTTP Fetch..
- A connects to B via the most optimal path
- A & B also exchanges their supported media and security

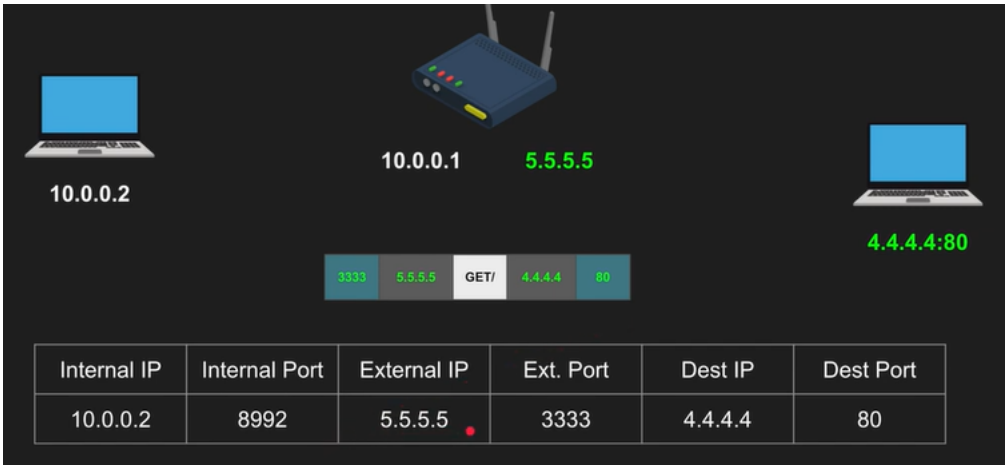
1. A wants to connect to B
2. A creates an “offer”, it finds all ICE candidates, security options, audio/video options and generates SDP, the offer is basically the SDP
3. A signals the offer somehow to B (whatsapp)
4. B creates the “answer” after setting A’s offer
5. B signals the “answer” to A
6. Connection is created

Peer 1 creates an offer for connection SDP(session description protocol). This is then stored in a server which can be read and returns SDP answer.

ICE (Interactive Connectivity Establishment) is a standard that helps in the discovering of IP addresses . ICE candidates are composed by a port and an IP). These ICE candidates are created using WebRTC by sending a serious of commands to a STUN server (session traversal utilities) owned by google.

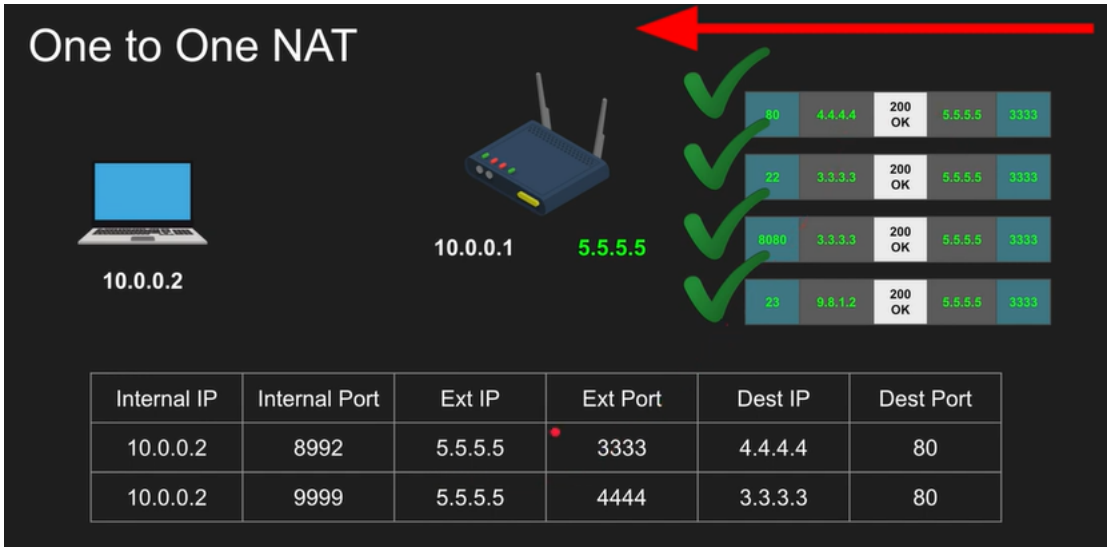
Network Address Translation (NAT)

✅ Subnet masking is searching for whether or not an IP is part of a specific subnet.



Internal IP	Internal Port	External IP	Ext. Port	Dest IP	Dest Port
10.0.0.2	8992	5.5.5.5	3333	4.4.4.4	80

▼ Types of NAT



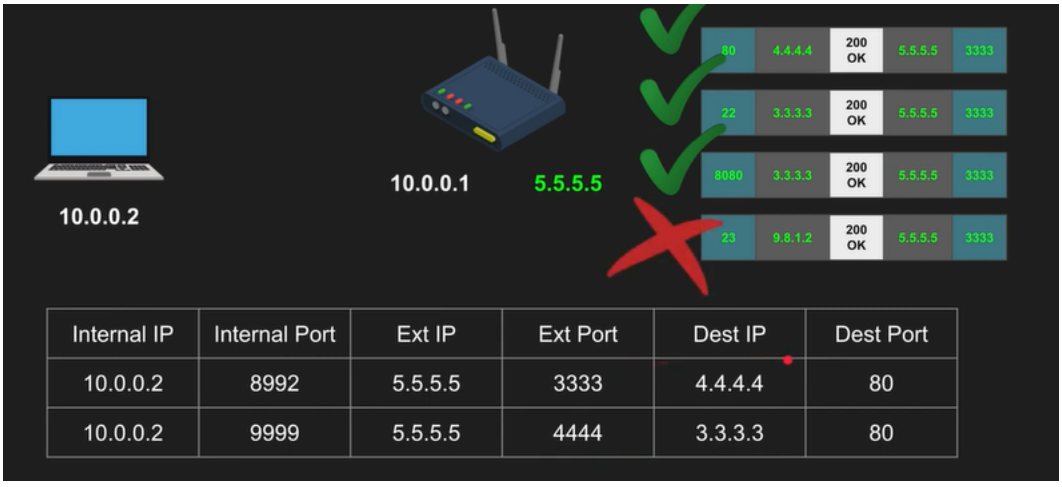
Internal IP	Internal Port	Ext IP	Ext Port	Dest IP	Dest Port
10.0.0.2	8992	5.5.5.5	3333	4.4.4.4	80
10.0.0.2	9999	5.5.5.5	4444	3.3.3.3	80

Full Cone NAT: Peer to peer data transmission without caring where its is coming from once the details are on the NAT table.

Address Restricted NAT

- Packets to external IP:port on the router always maps to internal IP:port as long as source address from packet matches the table (regardless of port)
- Allow if we communicated with this host before

Internal IP	Internal Port	Ext IP	Ext Port	Dest IP	Dest Port
10.0.0.2	8992	5.5.5.5	3333	4.4.4.4	80

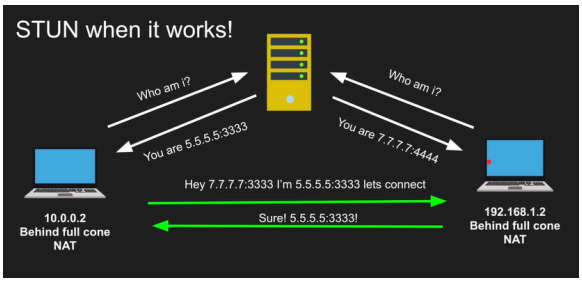


Internal IP	Internal Port	Ext IP	Ext Port	Dest IP	Dest Port
10.0.0.2	8992	5.5.5.5	3333	4.4.4.4	80
10.0.0.2	9999	5.5.5.5	4444	3.3.3.3	80

- Session Traversal Utilities for NAT
- Tell me my public ip address/port through NAT
- Works for Full-cone, Port/Address restricted NAT
- Doesn't work for symmetric NAT
- STUN server port 3478, 5349 for TLS
- Cheap to maintain

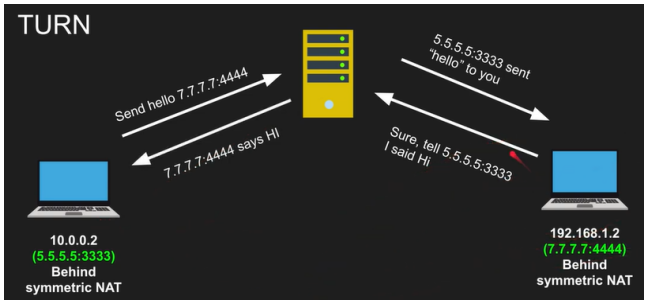
Session Traversal Utilities for NAT (STUN)

Google has a lot of STUN servers available for use. It finds out public address (presence) to enable communication.



Traversal Using Relays around NAT

Used when direct peer to peer can't be used.



Interactive connectivity establishment (ICE)

ICE collects all available candidates (local IP addresses, reflexive addresses – STUN ones and relayed addresses – TURN ones)

Called ice candidates

All the collected addresses are then sent to the remote peer via SDP

Session Descriptipn Protocol (SDP)

- Session Description Protocol
- A format that describes ice candidates, networking options, media options, security options and other stuff
- Not really a protocol its a format
- Most important concept in WebRTC
- The goal is to take the SDP generated by a user and send it “somehow” to the other party

- SDP Signaling
- Send the SDP that we just generated somehow to the other party we wish to communicate with
- Signaling can be done via a tweet, QR code, Whatsapp, WebSockets, HTTP request DOESN’T MATTER! Just get that large string to the other party

WebRTC Pros & Cons

- Pros
  - P2p is great ! low latency for high bandwidth content
  - Standardized API I don’t have to build my own
- Cons
  - Maintaining STUN & TURN servers
  - Peer 2 Peer falls apart in case of multiple participants (discord case)

Technology Evolution

History

History # When learning WebRTC developers often feel frustrated by the complexity. They see WebRTC features irrelevant to their current project and wish WebRTC was simpler. The issue is that everyone has a different set of use cases. Real-time communications has a rich history with lots of different people building many different things. This chapter contains interviews with the authors of the protocols that make up WebRTC. It gives insight into the designs made when building each protocol, and finishes

https://webrtcforthecurious.com/docs/10-history-of-webrtc/

Current applications in serious games

For anything other than entertainment

Current applications in digital twins

For remote monitoring and control, adjustments, and receive feedback on the impact of their actions.

Current applications in user interfaces

For interactivity and user experience.

Unreal Engine's Pixel Streaming:

Unreal Pixel Streaming allows developers to stream rendered frames and audio from a remote GPU enabled computer (e.g., cloud) to their users through a desktop or mobile web browser, without the need for the users to have special GPU hardware. More than just a stream, Pixel Streaming allows users to interact with the 3D application, as mouse and keyboard inputs are communicated back to the remote server and processed with low latency. This gives the user the experience as if they were using the application right on their own computer or mobile device. Some common use cases for this are for games, simulations, or immersive experiences such as configuring a new car or walking through a virtual showroom. Applications are developed in Unreal Engine and then exported out as a Windows executable to be placed on a GPU capable server, which then communicates with a local WebRTC NodeJS server to broker the stream and interactions to and from the client's browser connection.

Unity's Render Streaming:

Unity's Render Streaming enables real-time streaming of rendered frames from a Unity application to a client device using WebRTC. It allows users to play Unity games remotely without installing them locally and supports a range of platforms, including web browsers, mobile devices, and virtual reality headsets.

Render Streaming works by capturing the rendered frames from the Unity application, encoding them, and transmitting them over a network connection via WebRTC. The client device receives the video stream, decodes it, and renders the frames on the user's screen. Users can interact with the game by sending input data back to the server, enabling real-time gameplay and interaction.

Render Streaming can be utilized for remote gaming, virtual reality experiences, interactive streaming content, or other applications where real-time rendering and remote interaction are required. It leverages WebRTC's video streaming and communication capabilities to provide a flexible and accessible way for users to engage with Unity applications remotely.

Practical Example

https://github.com/hnasr/javascript_playground/tree/master/webrtc

“Server”

```
const lc = new RTCPeerConnection() //local connection

const dc= lc.createDataChannel("channel"); //data channel

dc.onmessage = e => console.log("Just got a message" + e.data);

dc.onopen = e=> console.log("Connection opened!");

lc.onicecandidate = e => console.log("New Ice Candidate! reprinting SDP" + JSON.stringify(lc.localDescription))
lc.createOffer().then(o => lc.setLocalDescription(o) ).then(a => console.log("set successfully!")) //local SDP

VM3761:1 New Ice Candidate! reprinting SDP{"type":"offer","sdp":"v=0\r\no=- 8587226863409573109 2 IN IP4 127.0.0.1\r\ns=-\r\nt=0 0\r\na=gro
VM3761:1 New Ice Candidate! reprinting SDP{"type":"offer","sdp":"v=0\r\no=- 8587226863409573109 2 IN IP4 127.0.0.1\r\ns=-\r\nt=0 0\r\na=gro
VM3761:1 New Ice Candidate! reprinting SDP{"type":"offer","sdp":"v=0\r\no=- 8587226863409573109 2 IN IP4 127.0.0.1\r\ns=-\r\nt=0 0\r\na=gro
VM3761:1 New Ice Candidate! reprinting SDP{"type":"offer","sdp":"v=0\r\no=- 8587226863409573109 2 IN IP4 127.0.0.1\r\ns=-\r\nt=0 0\r\na=gro
VM3761:1 New Ice Candidate! reprinting SDP{"type":"offer","sdp":"v=0\r\no=- 8587226863409573109 2 IN IP4 127.0.0.1\r\ns=-\r\nt=0 0\r\na=gro
2VM3761:1 New Ice Candidate! reprinting SDP{"type":"offer","sdp":"v=0\r\no=- 8587226863409573109 2 IN IP4 127.0.0.1\r\ns=-\r\nt=0 0\r\na=gr
const answer = {"type":"answer","sdp":"v=0\r\no=- 237967498524090967 2 IN IP4 127.0.0.1\r\ns=-\r\nt=0 0\r\na=group:BUNDLE 0\r\na=extmap-all
lc.setRemoteDescription(answer)
dc.send("Hello Micael!")
```

“Client”

```
//signaling (doesnt matter how it gets the ICE candidates)
const offer= {"type":"offer","sdp":"v=0\r\no=-
8587226863409573109 2 IN IP4 127.0.0.1\r\ns=-\r\nt=0 0\r\na=group:BUNDLE 0\r\na=extmap-allow-mixed\r\na=msid-semantic: WMS\r\nm=application
typ host generation 0 network-id 2\r\na=ice-ufrag:clwg\r\na=ice-pwd:6s5XYfnf42l6mT7RMmvJAAb3\r\na=ice-options:trickle\r\na=fingerprint:sha-

const rc= new RTCPeerConnection(); //remote connection

rc.onicecandidate = e => console.log("New Ice Candidate! reprinting SDP" + JSON.stringify(rc.localDescription))
e => console.log("New Ice Candidate! reprinting SDP" + JSON.stringify(rc.localDescription)) // SDP

rc.ondatachannel= e => {
  rc.dc= e.channel; //create variable as part of the channel
rc.dc.onmessage = e=> console.log("new message"+ e.data)
rc.dc.onopen = e => console.log("connection open!")
}

rc.setRemoteDescription(offer).then(a=> console.log("offer set!"))

rc.createAnswer().then(a => rc.setLocalDescription(a)).then(a=>console.log("answer created"))

rc.dc.send("Hello!")
```

Extras

Render Streaming

- Render Streaming: Render Streaming is a feature that allows you to stream the rendered frames of a game directly from the game engine running on a server to a client device. The client device doesn't need to run the entire game; it only needs to display the streamed frames, making it suitable for lightweight devices like smartphones or low-end computers. Uses DASH and HTTP requests.

bottleneck created if more than 4 players in the same render streaming

main features:

- Video streaming
- Audio streaming
- Remote control

Pixel Streaming

- Pixel Streaming: streams the actual pixels of the game's rendered frames to the client device. Pixel Streaming requires the client device to have a compatible browser or application that can process and display the pixel data, meaning it requires more processing power on the client's side.

Web RTC can be used in this mainly for communication purposes

Digital Twins help companies understand the past, view present conditions, and prevent future problems.

Differences

Render Streaming and **Pixel Streaming** are technologies used to enable remote gameplay and rendering in game development. They both allow users to experience games and interactive applications running on remote servers without the need for powerful local hardware. Instead, the server handles the heavy rendering and processing tasks, and the client device (such as a web browser or a mobile device) receives the game's visuals and input, providing a responsive experience.

Differences between the Engines

Game Engines:	Unreal Game Engine	Unity Game Engine
ease of use	steeper learning curve	straightforward learning curve
performance	high performance	high performance
scalability	AAA games, complex environments, high-quality graphics	highly scalable for differently sized projecs
compatability	cross-platform compatibility	cross-platform compatibility
support	visually impressive graphics	easy and rapid prototyping

Differences between the technologies

Technologies	Pixe Streaming	Render Streaming
ease of use	handling the pixel data on the client's side. Additional considerations are needed to ensure compatibility and performance across different devices and browsers.	requires minimal client-side processing and is often more straightforward to integrate into existing projects.
performance	more efficient in terms of server resources because it transfers only pixel data, which is smaller compared to full rendered frames. However, it may require more processing power on	involves streaming the rendered frames, the performance impact on the server is higher, especially if multiple clients are connected. It requires more server resources to handle multiple

	the client's side to decode and display the pixel data, especially for graphically demanding games.	rendering processes simultaneously.
compatability	requires a client device with a more powerful GPU and capable browser or application to handle the pixel data decoding and rendering.	more compatible with various devices since it only requires a compatible web browser or a lightweight application on the client side.
support	used for high-quality, graphically demanding games and applications where the client's GPU can render the frames with better fidelity.	suitable for applications where the emphasis is on providing a lightweight, accessible, and responsive experience on a broader range of devices. It's often used for remote gaming, lightweight interactive experiences, and virtual tours.

Advantages of Web RTC

Peer to Peer: Players can connect directly to each other without the need for a central server, reducing latency and the need for dedicated server resources.

Designed for real-time communication, so it aims to minimize latency, making it suitable for fast-paced multiplayer games.

Tutorial Based on Unity Documentation

Adding Render Streaming to a Project

1) After having the project created and running we have to add render streaming to the project

2) Select Window > Package Manager in the menu bar.

3) Check Package Manager window, Click **+** button and select Add package by name. If you use Unity 2020.3, select Add package from git URL.

4) Input the string below to the input field.

```
com.unity.renderstreaming
```

5) The Render Streaming Wizard window is opened automatically after installing the package. Select Fix All.

6) Select Play on Unity Editor

Launching Web App

1) Select Download latest version web app on the Render Streaming Wizard window.

2) Open the “web app” downloaded and a console will appear. It will give a list of all the IPs you can use to see the streaming.

3) Once you are in the web address click on receiver sample and you should be able to control the game.

References

https://unity.com/solutions/digital-twin-definition

Web RTC

https://www.youtube.com/watch?v=qyS3grRfiLA&ab_channel=DevoxxUK

https://www.youtube.com/watch?v=FExZvpVvYxA&ab_channel=HusseinNasser

Render Streaming Setup

https://hackmd.io/@shinn716/H1gBahLKc

https://docs.unity3d.com/Packages/com.unity.renderstreaming@3.1/manual/index.html

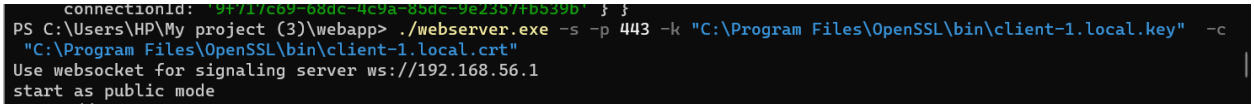
https://assetstore.unity.com/packages/essentials/starter-assets-third-person-character-controller-urp-196526

https://docs.unity3d.com/Packages/com.unity.renderstreaming@3.1/manual/settings.html#signaling-settings

https://docs.unity.cn/Packages/com.unity.renderstreaming@1.2/manual/en/overview.html

Render Streaming execution

https://www.youtube.com/watch?v=zsDHTDAXaE0&ab_channel=FusedVR



https://medium.com/@eclipsek24/dev-log-132-getting-started-with-unity-render-streaming-webrtc-939729ddac1e

https://docs.unity.cn/Packages/com.unity.renderstreaming@1.2/manual/en/tutorial.html

Troubleshooting

https://www.youtube.com/watch?v=nB0r0c-SIVg&ab_channel=UGuruz

https://chat.openai.com/share/08b242f0-0aa9-4ca7-9f9b-88484daa738e

Unreal Pixel Streaming

https://www.wowza.com/community/t/unreal-engine-pixel-streaming-webrtc-to-rtsp/94961

https://docs.unrealengine.com/5.2/en-US/overview-of-pixel-streaming-in-unreal-engine/