Real-Time Collaborative Virtual Reality Across the Continent

Bohan Wu | Brandon Fremin | Junjie Lei | Melody Hsu

Virtual Reality

- Computer generated simulated experience or environment
- Fully immersive through artificially constructed images and sounds
- Uses equipment such as a headset and controllers fitted with sensors
- Applications in business, education, art, entertainment, etc.

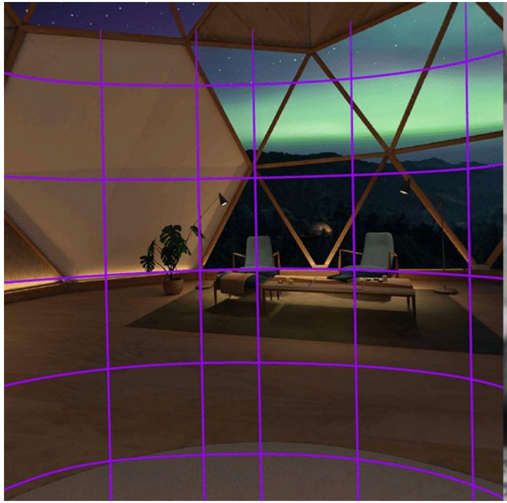
Oculus Specifications

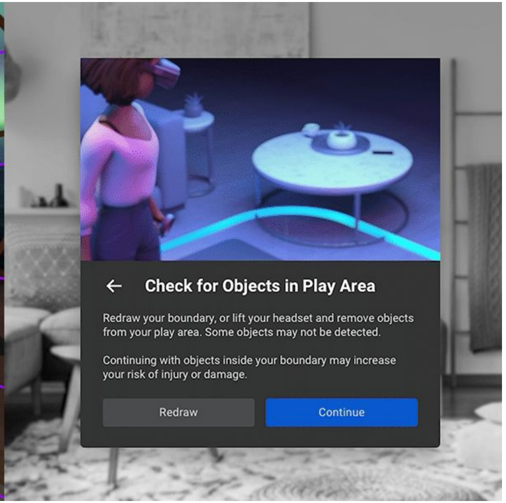
- 2 controllers + 1 headset
- 72 Hz frame refresh rate
- Must be connected with a Facebook account
- Local storage of apps and games that can be downloaded/uploaded
- Connects to Wi-Fi
    - Limitation: unable to connect to Wi-Fi networks that require 2-factor authentication

Oculus Game Demonstration

Components (Controllers, Cameras, Processing)

- Tracks user movement (controllers)
- Tracks surrounding play area (4 headset cameras)





Check for Objects in Play Area

Redraw your boundary, or lift your headset and remove objects from your play area. Some objects may not be detected.

Continuing with objects inside your boundary may increase your risk of injury or damage.

- Qualcomm Snapdragon XR2 Platform (little endian)

Problems

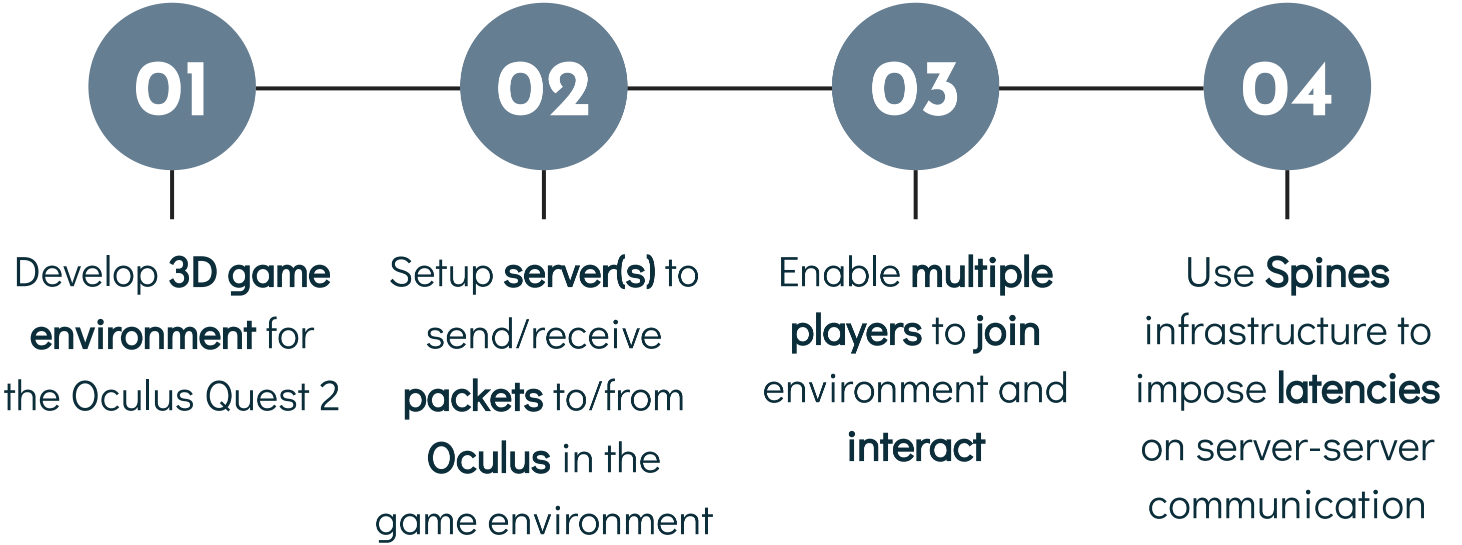
How do we ensure that users in the **same virtual space** are **experiencing** events and **interacting** at the **same time**?

How do we deal with **conflicting** updates from **different clients**?

Project Goals

- Develop simple multiplayer app for Oculus Quest 2 in which players can interact in real-time (<65 ms latency) from any two locations in the continental United States.
- All players see a consistent state of the world
- App is extensible to generic VR Headset use cases

Approach



01

Develop **3D game environment** for the Oculus Quest 2

02

Setup **server(s)** to send/receive **packets** to/from **Oculus** in the game environment

03

Enable **multiple players** to **join** environment and **interact**

04

Use **Spines** infrastructure to impose **latencies** on server-server communication

TABLE OF CONTENTS

01 Background

Virtual Reality
Oculus Quest 2 and Demo
Problem Introduction
Problem Approach

02 Game Development

Unity Game Engine
Game Objects
XR Interactive Toolkit
Control Flow and Scripts

03 Singleplayer & Multiplayer

04 Communication Architecture

Message Structures
Protocol Buffers
Synchronous Delivery

05 Spines

Introduction
Latency Graph & Reconstruction
Soft Real-time vs Source Based

06 Demonstration

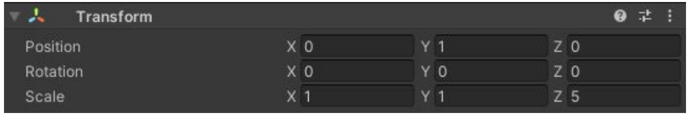
Metrics & Statistics
Limitations & Improvements

Unity Game Engine

- Cross platform game engine
- Supports desktop, mobile, console, and virtual reality platforms
- Game development for iOS and Android
  - Inclusive of 2D and 3D games, simulations, and experiences
- Scripting API in C# language
- Deployed as Android File (.apk)

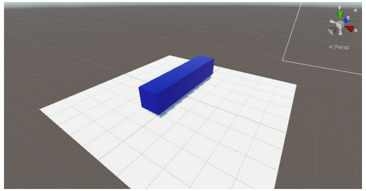
Unity GameObject

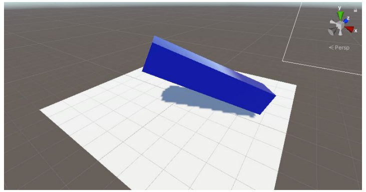
- GameObjects: Components in UnityEngine
  - Transform = (Position/Rotation)
    - Represented as 7 floats

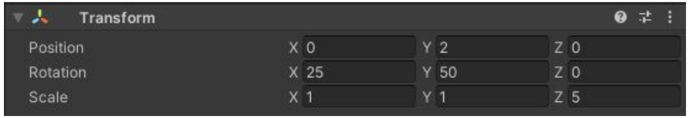


Transform

	X	Y	Z
Position	0	1	0
Rotation	0	0	0
Scale	1	1	5

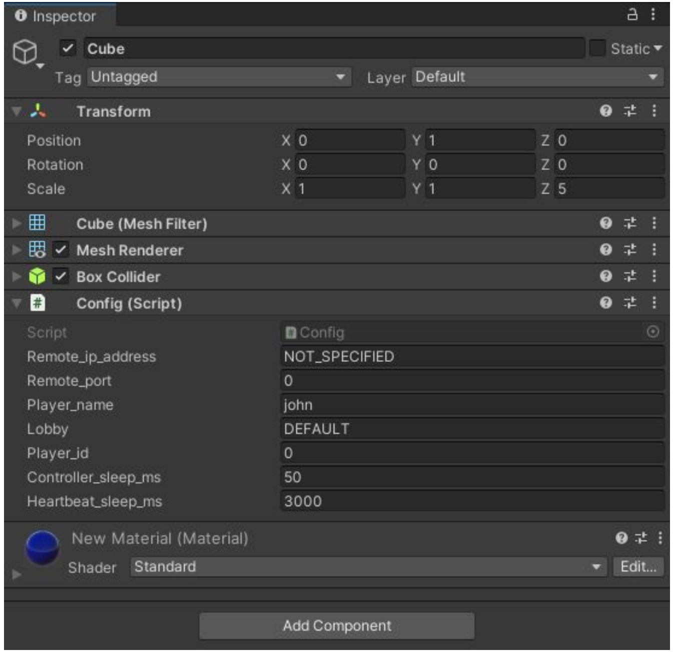






Transform

	X	Y	Z
Position	0	2	0
Rotation	25	50	0
Scale	1	1	5



Inspector

☑ Cube ☐ Static

Tag Untagged Layer Default

Transform

	X	Y	Z
Position	0	1	0
Rotation	0	0	0
Scale	1	1	5

Cube (Mesh Filter)

☑ Mesh Renderer

☑ Box Collider

Config (Script)

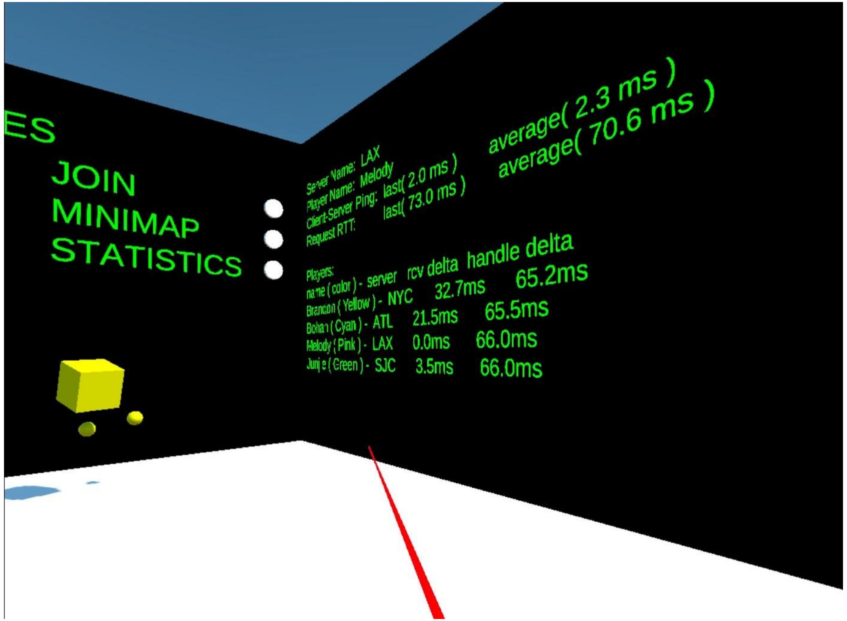
Script	Config
Remote_ip_address	NOT_SPECIFIED
Remote_port	0
Player_name	john
Lobby	DEFAULT
Player_id	0
Controller_sleep_ms	50
Heartbeat_sleep_ms	3000

New Material (Material)

Shader Standard

Extended Reality – XR Toolkit

- Camera rig
  - Track the user’s head movement to render the camera view.
- Controller
- Locomotion system
- Ray interactors
- Debugger UI Canvas
  - UI overlay used to output log onto the camera.



```
Server Name:  LAX
Player Name:  Melody
Client-Server Ping:  last( 2.0 ms )    average( 2.3 ms )
Request RTT:         last( 73.0 ms )   average( 70.6 ms )

Players:
name ( color ) - server   rcv delta   handle delta
Brandon ( Yellow ) - NYC    32.7ms     65.2ms
Bohan ( Cyan ) -  ATL       21.5ms     65.5ms
Melody ( Pink ) -  LAX      0.0ms      66.0ms
Junie ( Green ) -  SJC      3.5ms      66.0ms
```

Unity Engine (Lifecycle Control Flow)

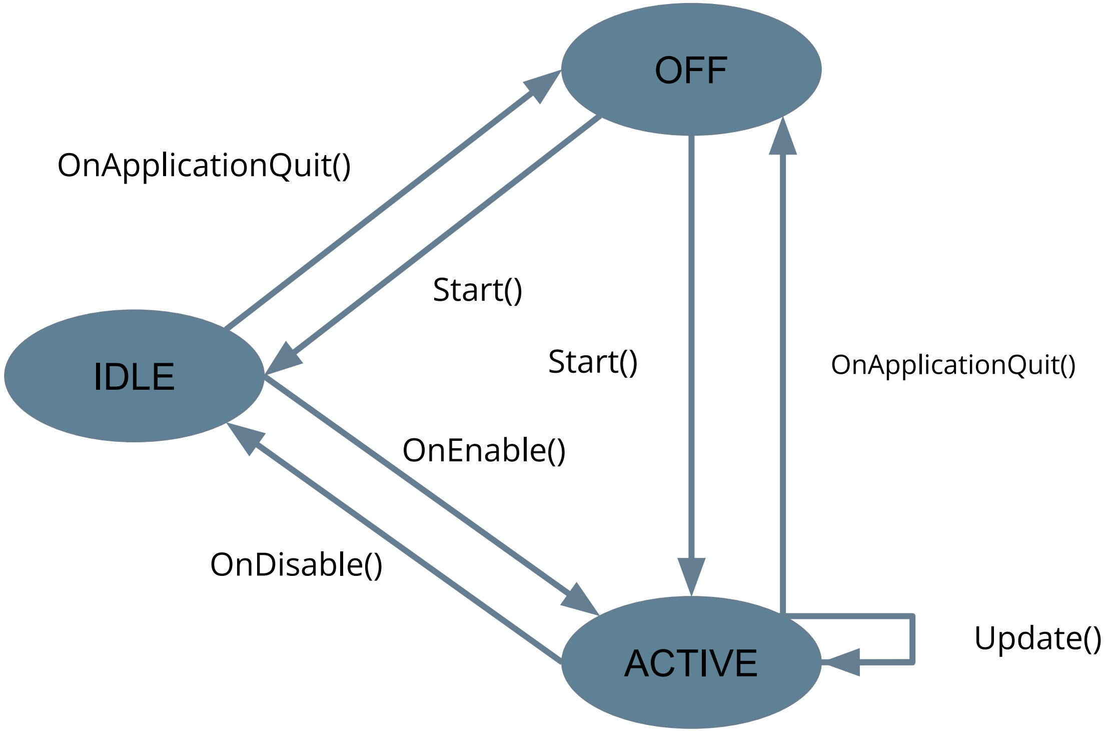


TABLE OF CONTENTS

01 Background

Virtual Reality
Oculus Quest 2 and Demo
Problem Introduction
Problem Approach

02 Game Development

Unity Game Engine
Game Objects
XR Interactive Toolkit
Control Flow and Scripts

03 Singleplayer & Multiplayer

04 Communication Architecture

Message Structures
Protocol Buffers
Synchronous Delivery

05 Spines

Introduction
Latency Graph & Reconstruction
Soft Real-time vs Source Based

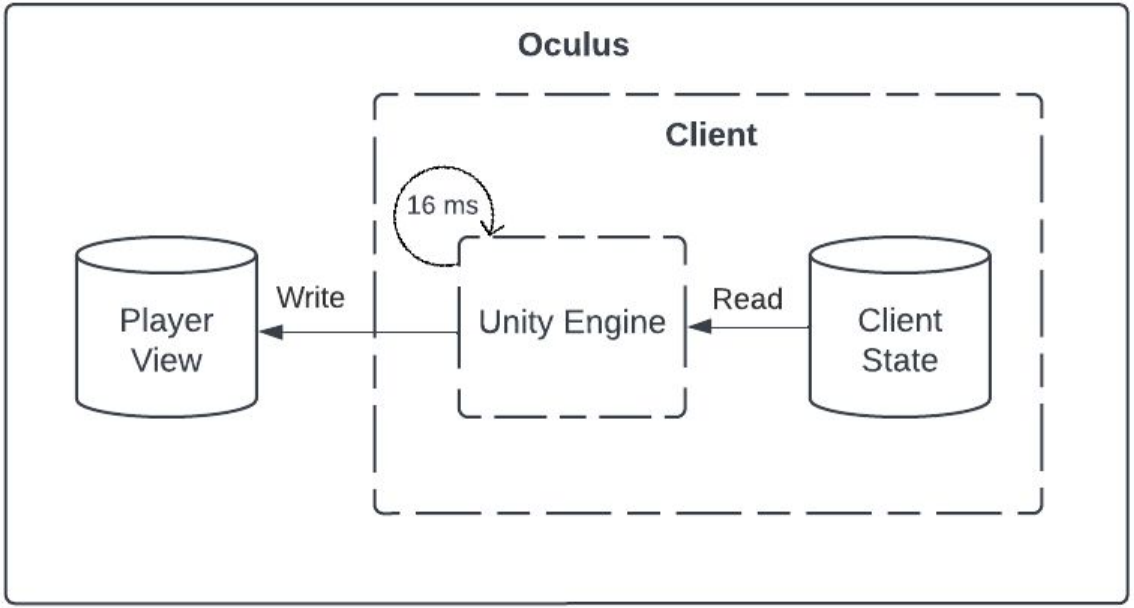
06 Demonstration

Metrics & Statistics
Limitations & Improvements

Single Player

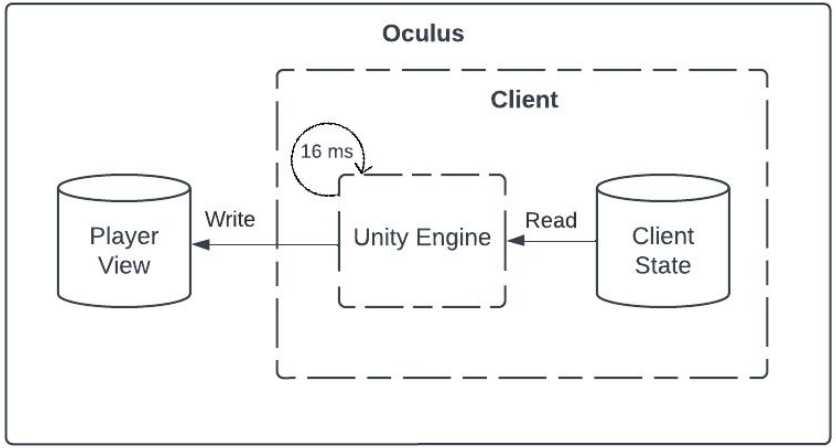
All code runs within the Oculus headset

Single Player (Video Only)

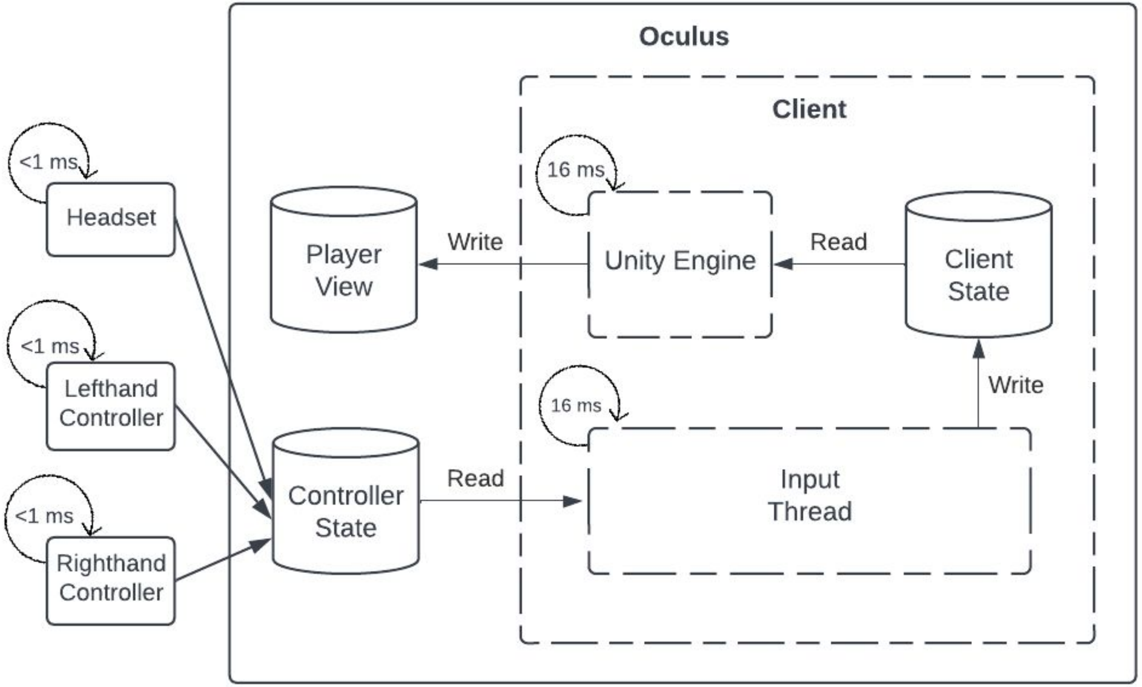


Unity Engine Rendering Frames

- Unity Engine's main loop takes care of rendering
- Frames rendered every 16 ms (60 frames/sec)
  - Clock starts upon app startup
  - Read from local state
  - Rendered for every user

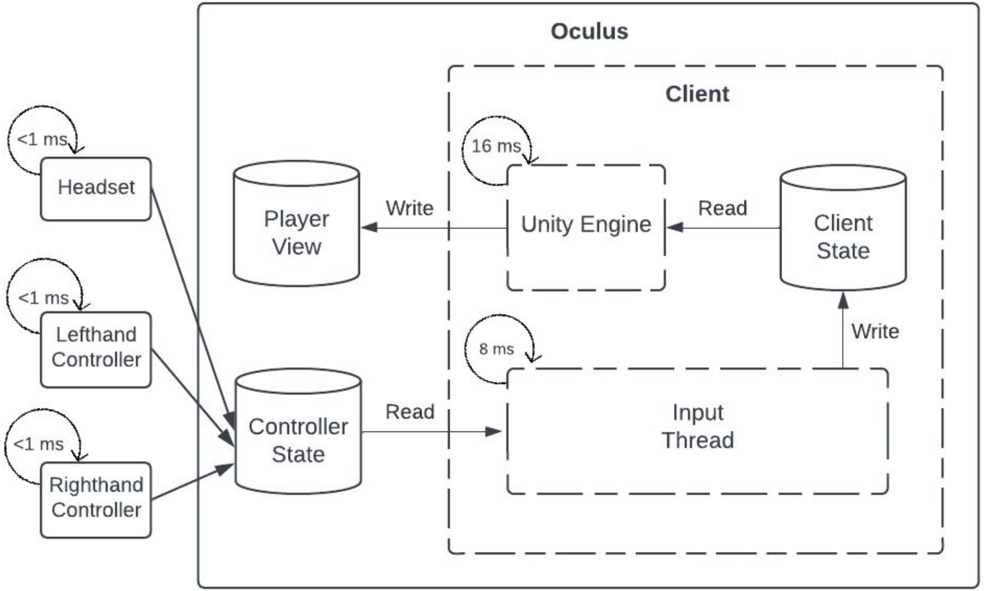


Single Player (Controller Inputs)

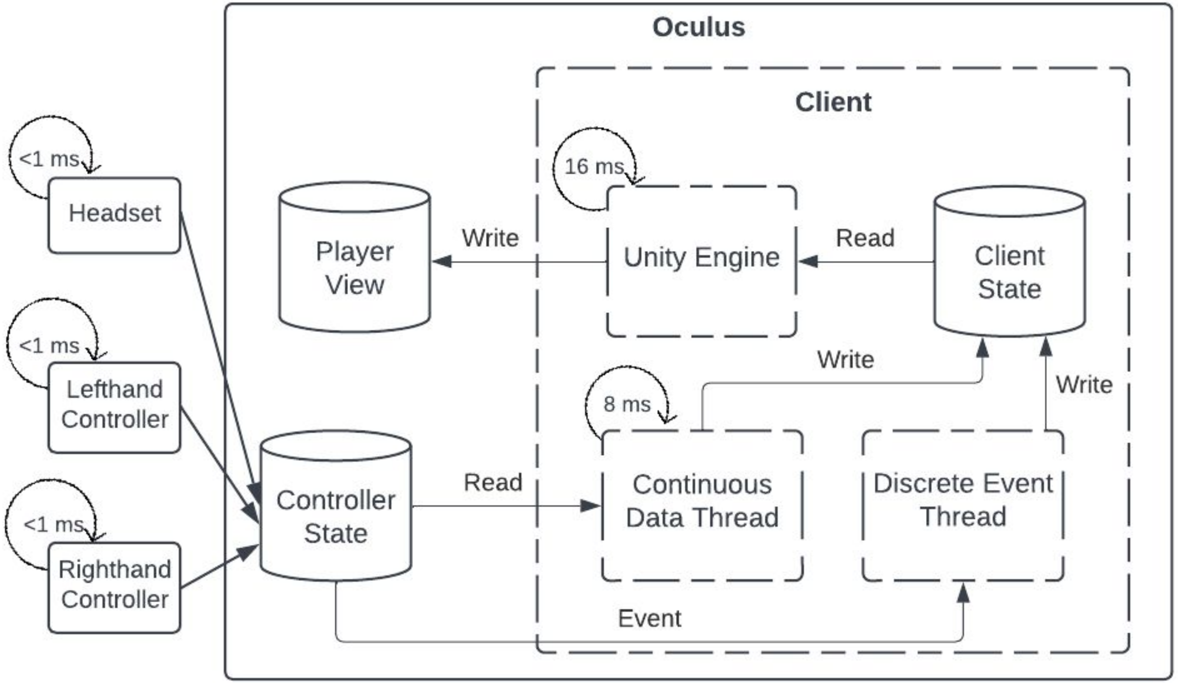


Headset, Left Controller, Right Controller

- Headset
  - Transform (7 floats)
  - Buttons (3 x 1 bools)
- Left/Right Controller
  - Transform (7 floats)
  - Joystick (2 floats)
  - Triggers (2 x 1 floats)
  - Buttons (6 x 1 bools)
- Controller data is sampled at a rate greater than 1 kHz
- Unity handles read/write atomicity
- Input thread can handle controller sampling errors, estimate velocity/acceleration, smooth out reading



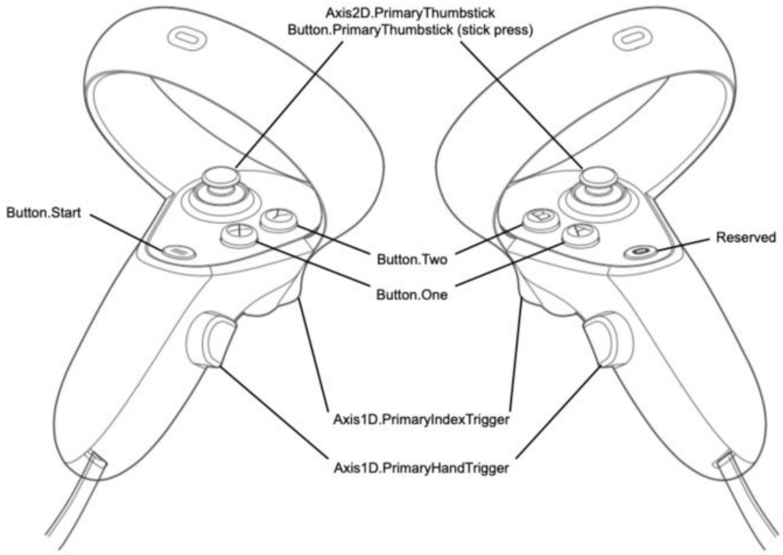
Single Player (Continuous and Discrete Inputs)



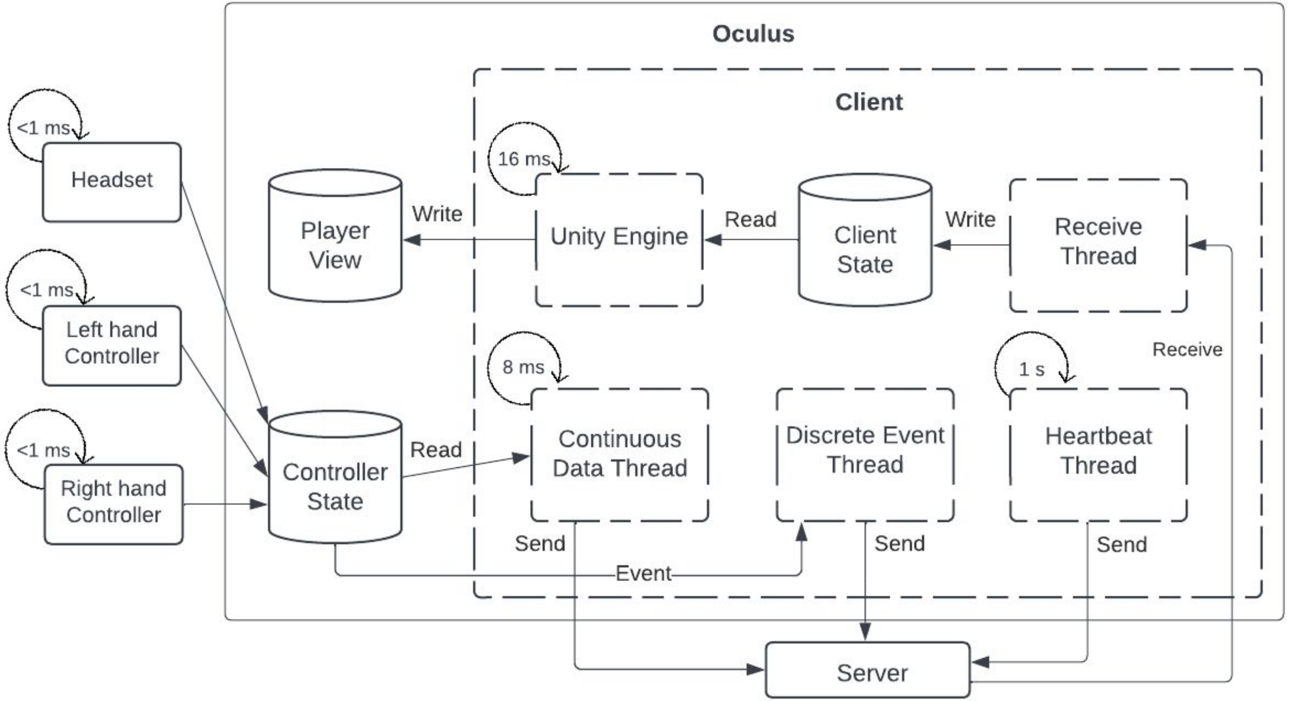
Single Player (Discrete Inputs)

- Buttons have boolean values
  - 0 → unpressed
  - 1 → pressed

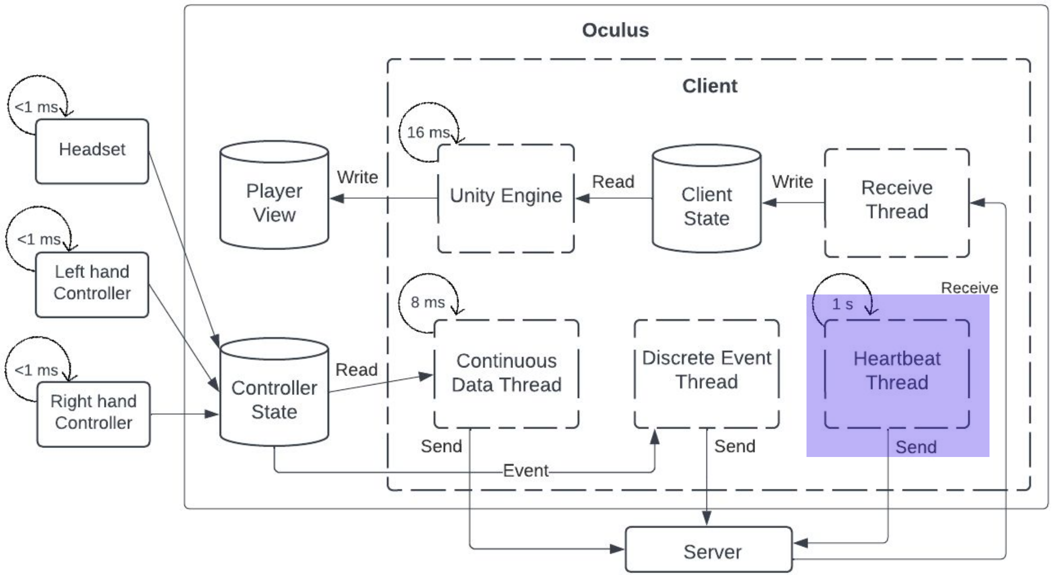
- Many ways to press a button
  - onUp
  - onDown
  - onPressAndHold
  - onDoubleClick


Axis2D.PrimaryThumbstick
Button.PrimaryThumbstick (stick press)
Button.Start
Reserved
Button.Two
Button.One
Axis1D.PrimaryIndexTrigger
Axis1D.PrimaryHandTrigger

Multiplayer (Single Server)



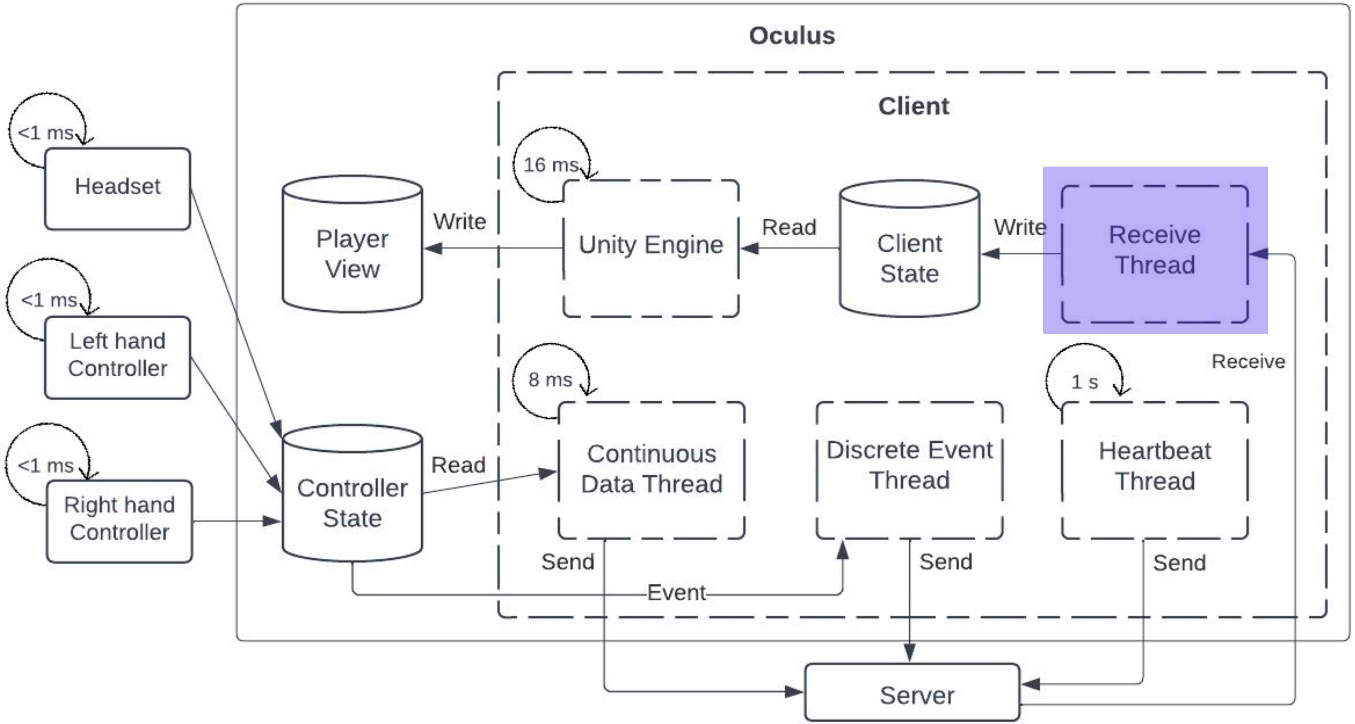
Heartbeat Thread



- Heartbeat Messages (Client-Server Ping)
- Metrics Messages

Receive Thread

- World Messages
- Heartbeat Messages
- Syn Messages
- Presence Messages
- RPC Messages
- Statistics Messages



Multiplayer Write Conflicts

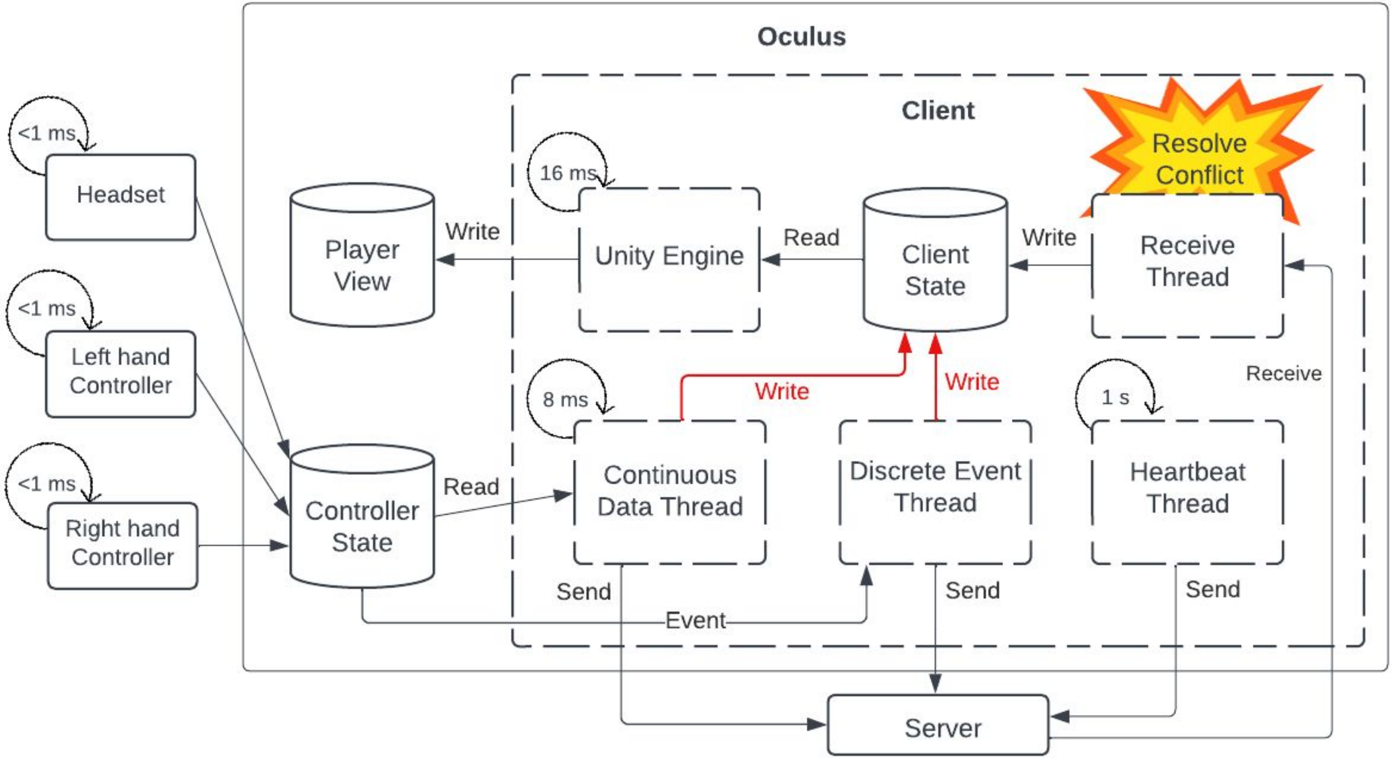


TABLE OF CONTENTS

01 Background

Virtual Reality
Oculus Quest 2 and Demo
Problem Introduction
Problem Approach

02 Game Development

Unity Game Engine
Game Objects
XR Interactive Toolkit
Control Flow and Scripts

03 Singleplayer & Multiplayer

04 Communication Architecture

Message Structures
Protocol Buffers
Synchronous Delivery

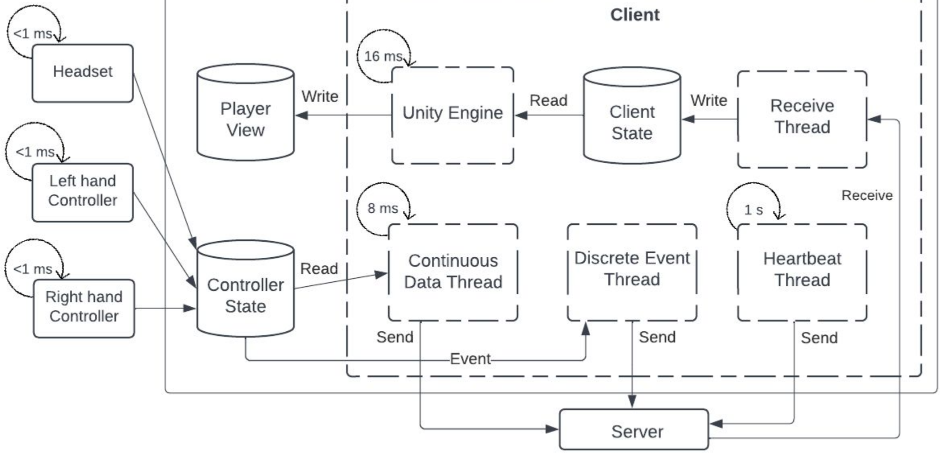
05 Spines

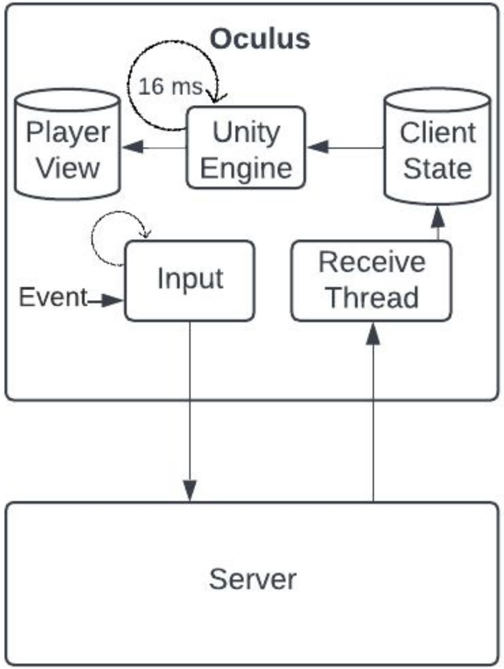
Introduction
Latency Graph & Reconstruction
Soft Real-time vs Source Based

06 Demonstration

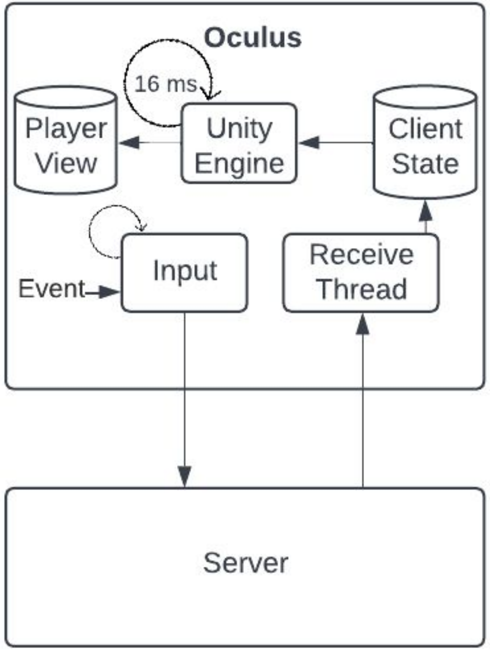
Metrics & Statistics
Limitations & Improvements

Client-Server Communication

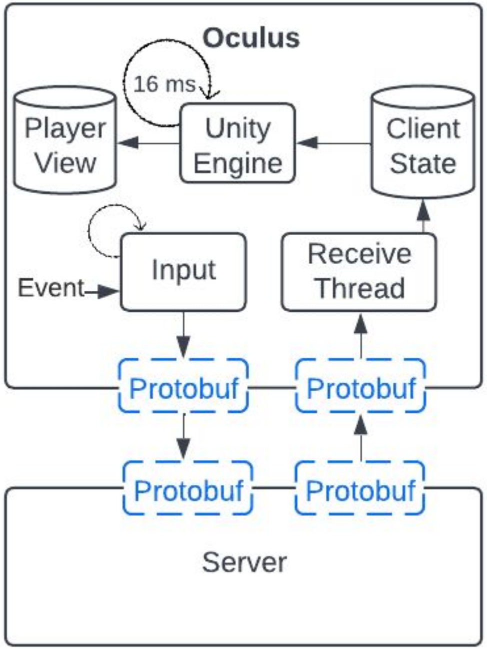




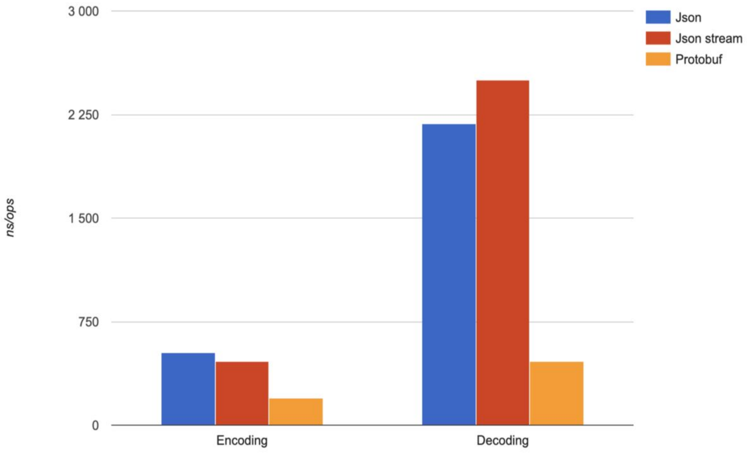
Client-Server Communication

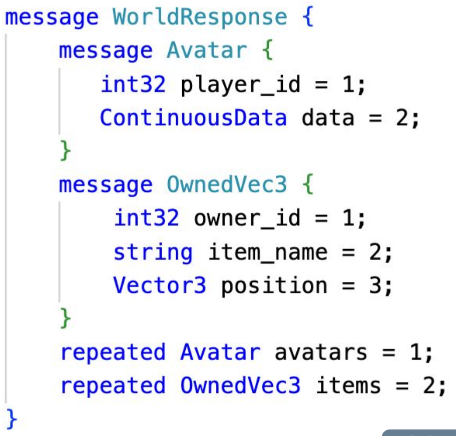


Client-Server Communication (Protobuf)



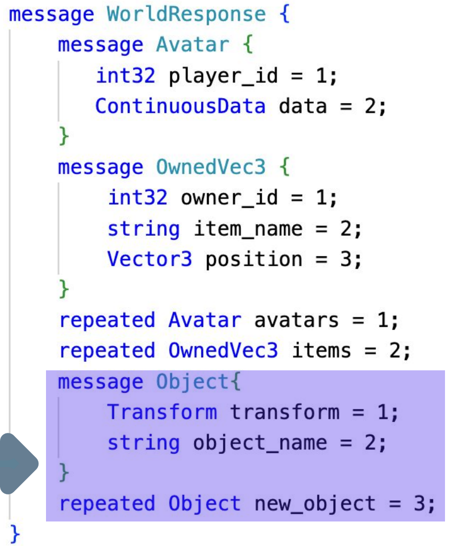
Client-Server Communication (Protobuf)





```
message WorldResponse {
    message Avatar {
        int32 player_id = 1;
        ContinuousData data = 2;
    }
    message OwnedVec3 {
         int32 owner_id = 1;
         string item_name = 2;
         Vector3 position = 3;
    }
    repeated Avatar avatars = 1;
    repeated OwnedVec3 items = 2;
}
```

New field

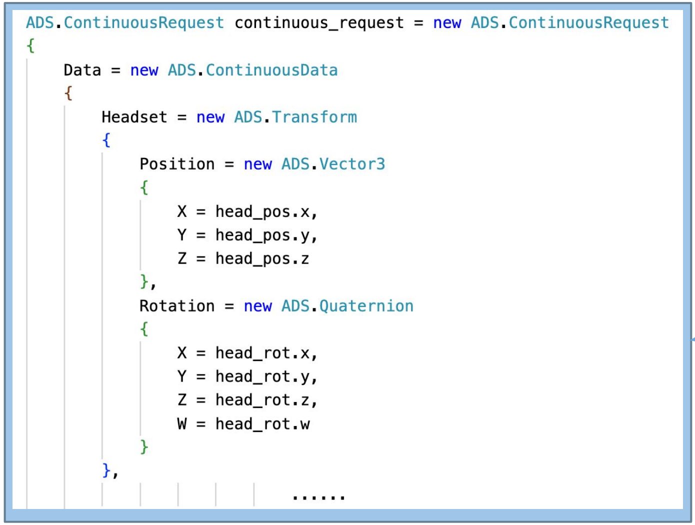


```
message WorldResponse {
    message Avatar {
        int32 player_id = 1;
        ContinuousData data = 2;
    }
    message OwnedVec3 {
         int32 owner_id = 1;
         string item_name = 2;
         Vector3 position = 3;
    }
    repeated Avatar avatars = 1;
    repeated OwnedVec3 items = 2;
    message Object{
         Transform transform = 1;
         string object_name = 2;
    }
    repeated Object new_object = 3;
}
```

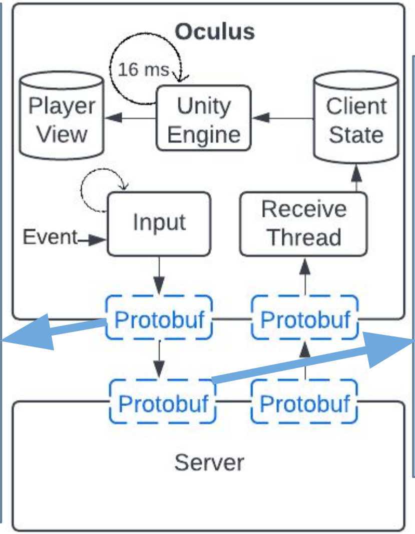
- Language independent
- Backwards compatibility/Implementation advantage
- Good performance

Client-Server Communication (Protobuf)

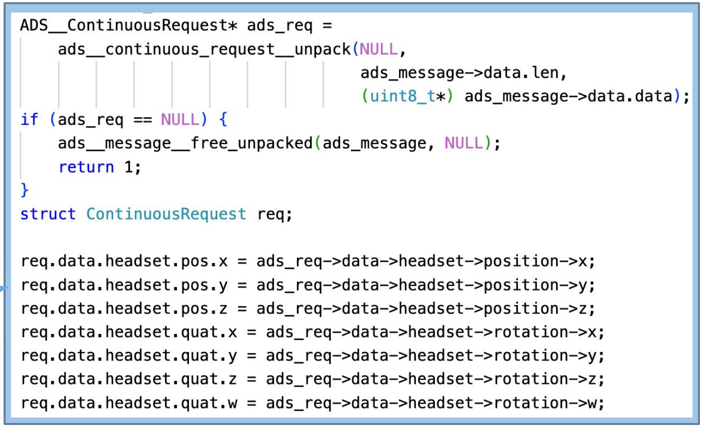
Protobuf (C#)



```
ADS.ContinuousRequest continuous_request = new ADS.ContinuousRequest
{
    Data = new ADS.ContinuousData
    {
        Headset = new ADS.Transform
        {
            Position = new ADS.Vector3
            {
                X = head_pos.x,
                Y = head_pos.y,
                Z = head_pos.z
            },
            Rotation = new ADS.Quaternion
            {
                X = head_rot.x,
                Y = head_rot.y,
                Z = head_rot.z,
                W = head_rot.w
            }
        },
                        ......
```



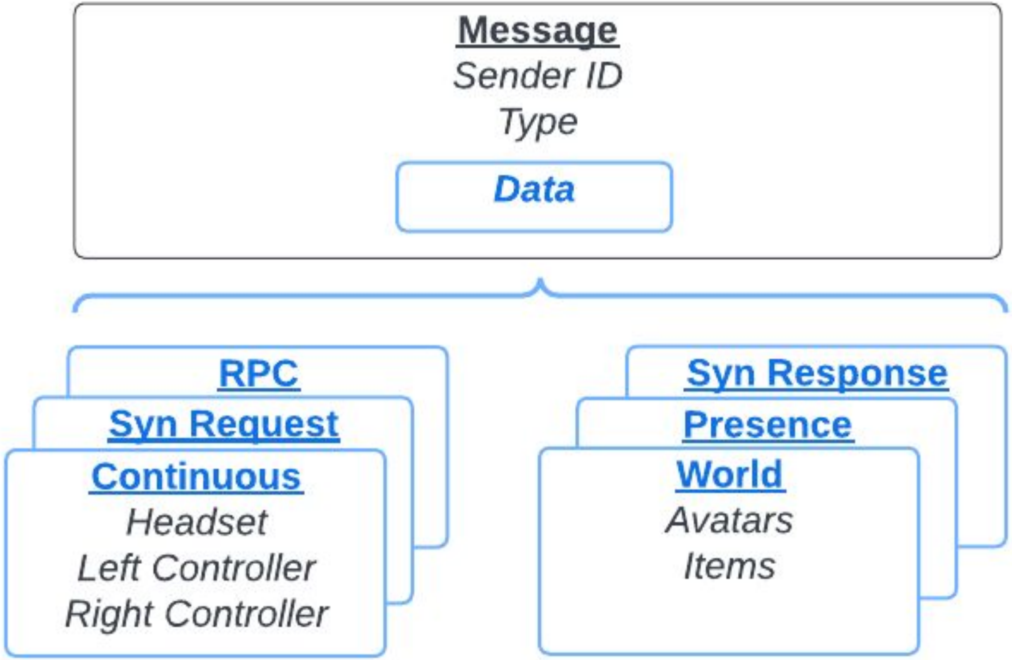
Protobuf (C)



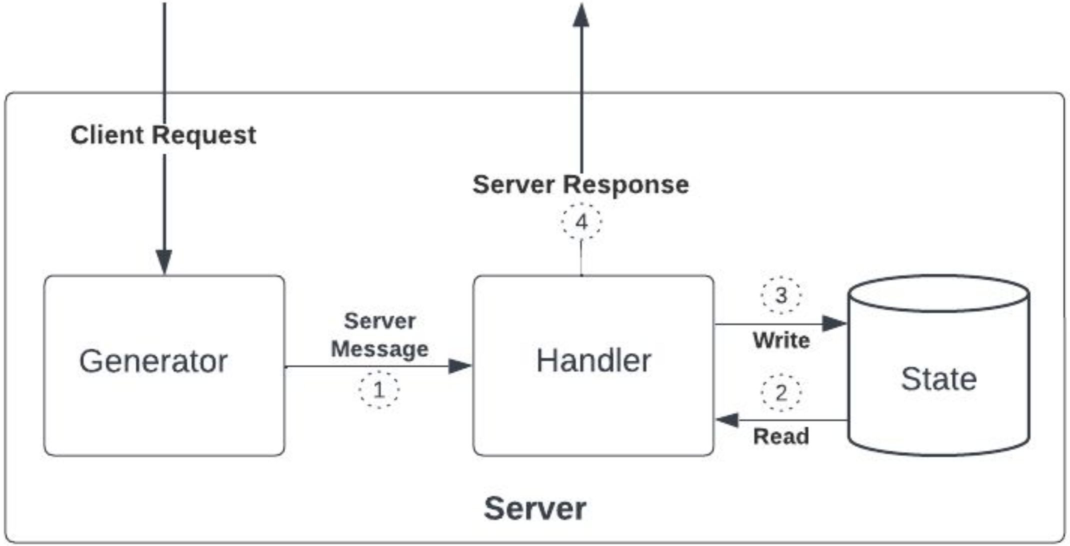
```
ADS__ContinuousRequest* ads_req =
    ads__continuous_request__unpack(NULL,
                                ads_message->data.len,
                                (uint8_t*) ads_message->data.data);
if (ads_req == NULL) {
    ads__message__free_unpacked(ads_message, NULL);
    return 1;
}
struct ContinuousRequest req;

req.data.headset.pos.x = ads_req->data->headset->position->x;
req.data.headset.pos.y = ads_req->data->headset->position->y;
req.data.headset.pos.z = ads_req->data->headset->position->z;
req.data.headset.quat.x = ads_req->data->headset->rotation->x;
req.data.headset.quat.y = ads_req->data->headset->rotation->y;
req.data.headset.quat.z = ads_req->data->headset->rotation->z;
req.data.headset.quat.w = ads_req->data->headset->rotation->w;
```

Message Structure

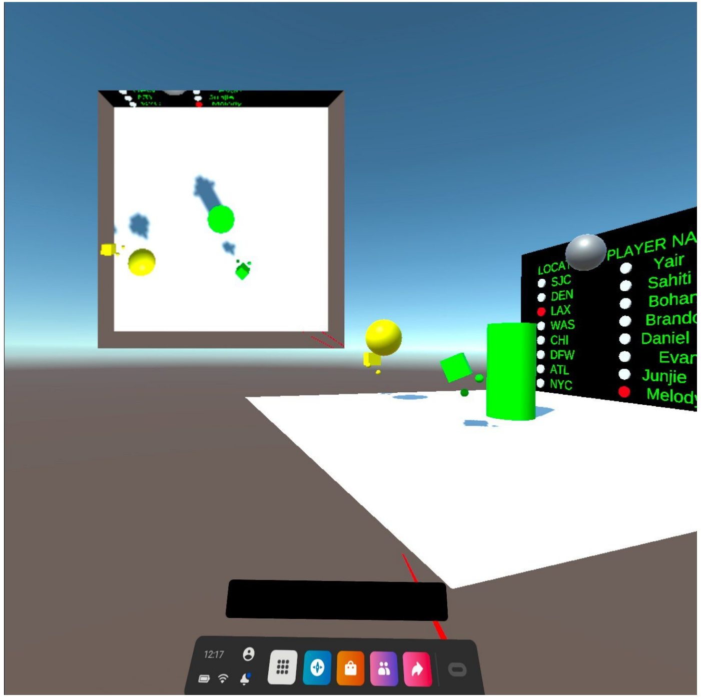


Multiplayer + Single Server



Server State

- Players
  - Logistical information: ID, Name, IP Address/Port, Ingest Server
  - Pose Information
    - Headset Transform
    - Left Controller Transform
    - Right Controller Transform
    - Offset
  - Movement Information
    - Body Velocity (Left Joystick)
- Items
  - Ownership
  - Item Transform
  - Item Velocity (Right Joystick)

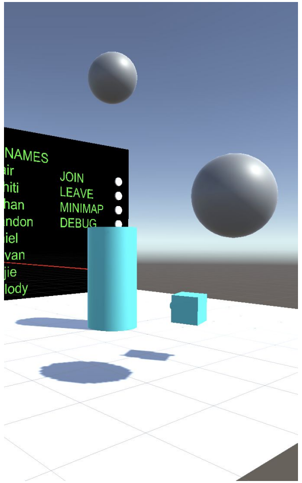


Interactivity

HAPTIC FEEDBACK

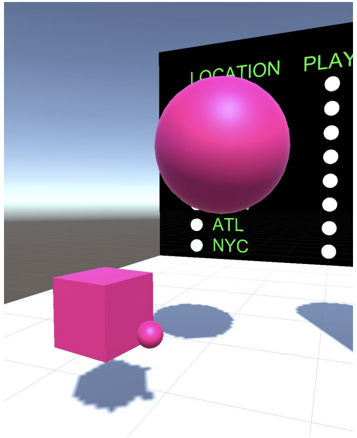
cylinder changes color to match the avatar of player who sent request

all players synchronously feel controller rumbles



REVOLVING SPHERE

motion indicates that server is active

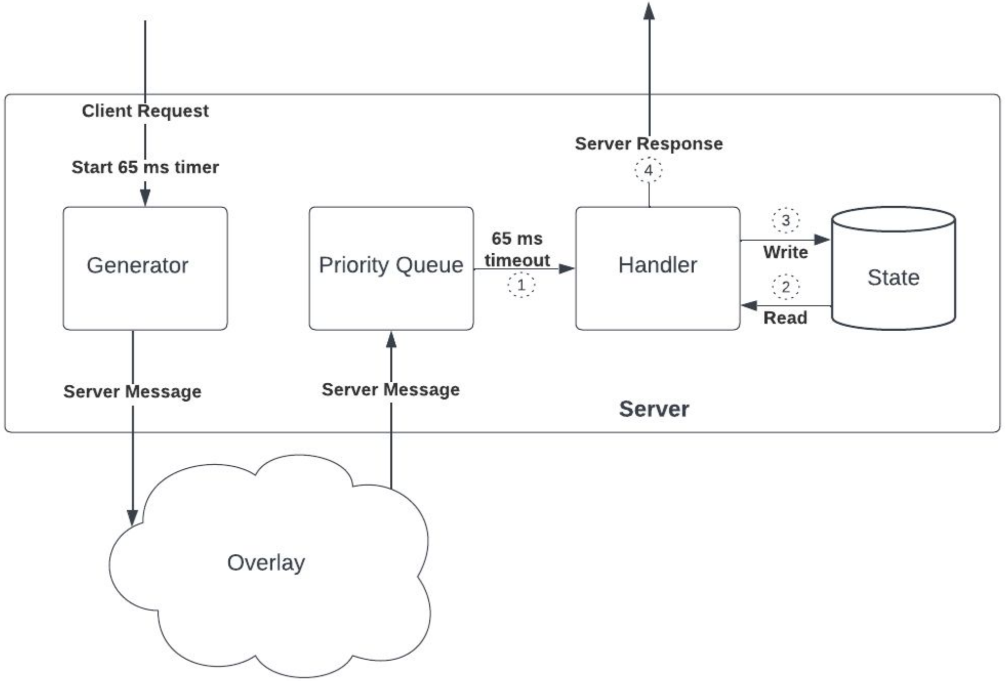


INTERACTABLE SPHERE

claim possession of a common object and change its position

object “owner” alone can move it – all others see its position changing

Multiplayer + Multiple Servers



Multiplayer (Fortnite)

A lot of updates → Flooding

- 100 players in one game

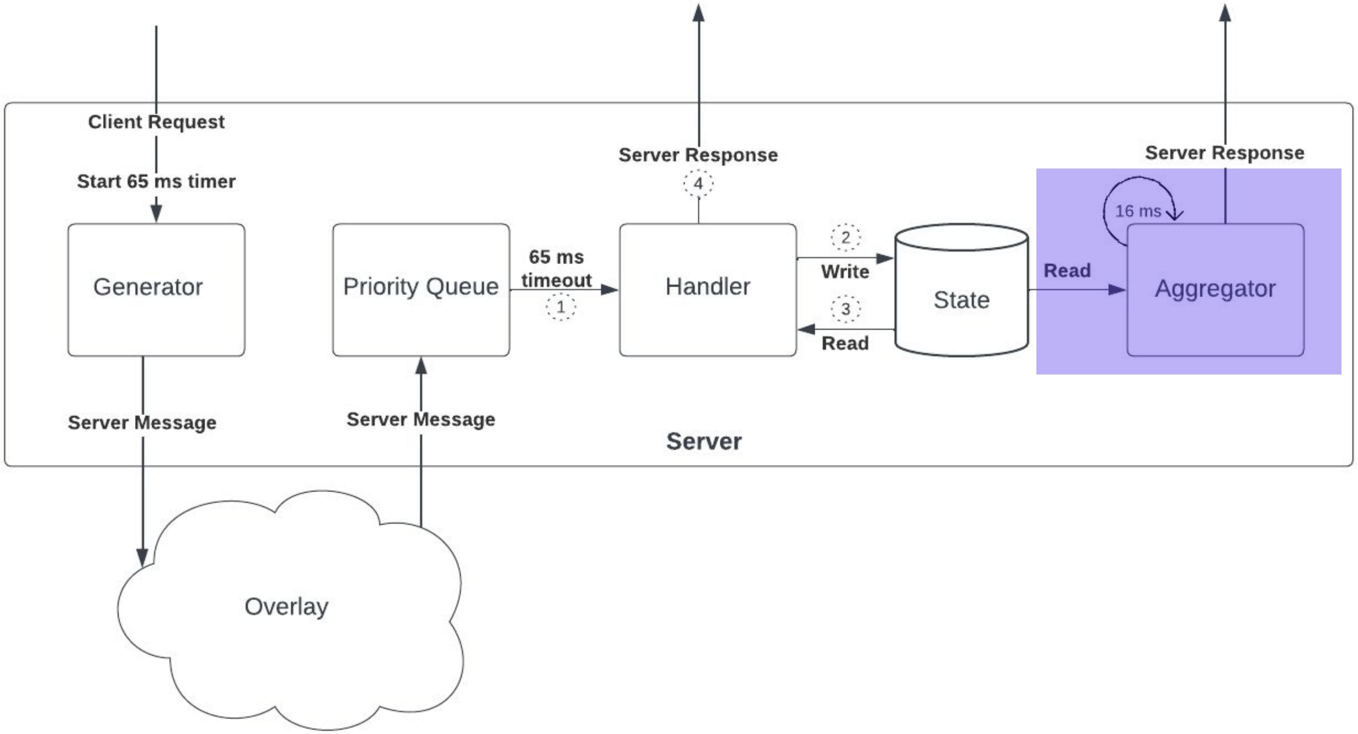
Limited computing power → Efficiency matters

- 116 million people played Fortnite on iOS devices.

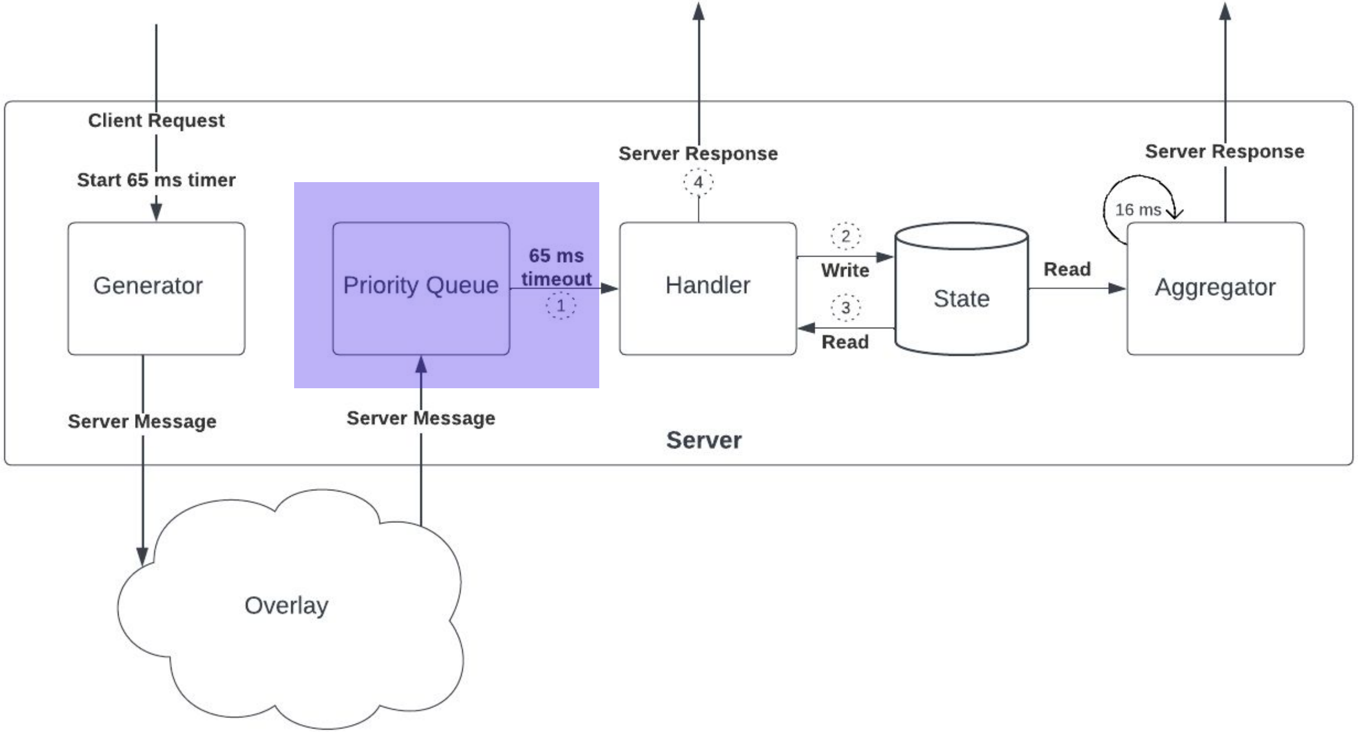
Updates not needed → Send cumulative updates

- Client render every 16 ms for 60Hz refresh rate

Multiplayer + Multiple Servers + Aggregator

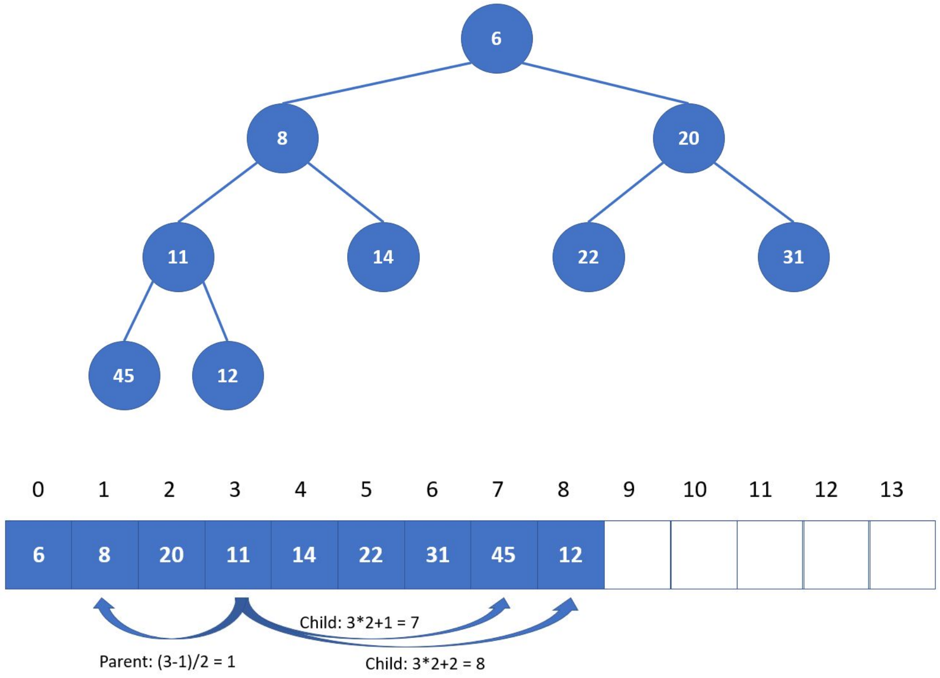


Priority Queue



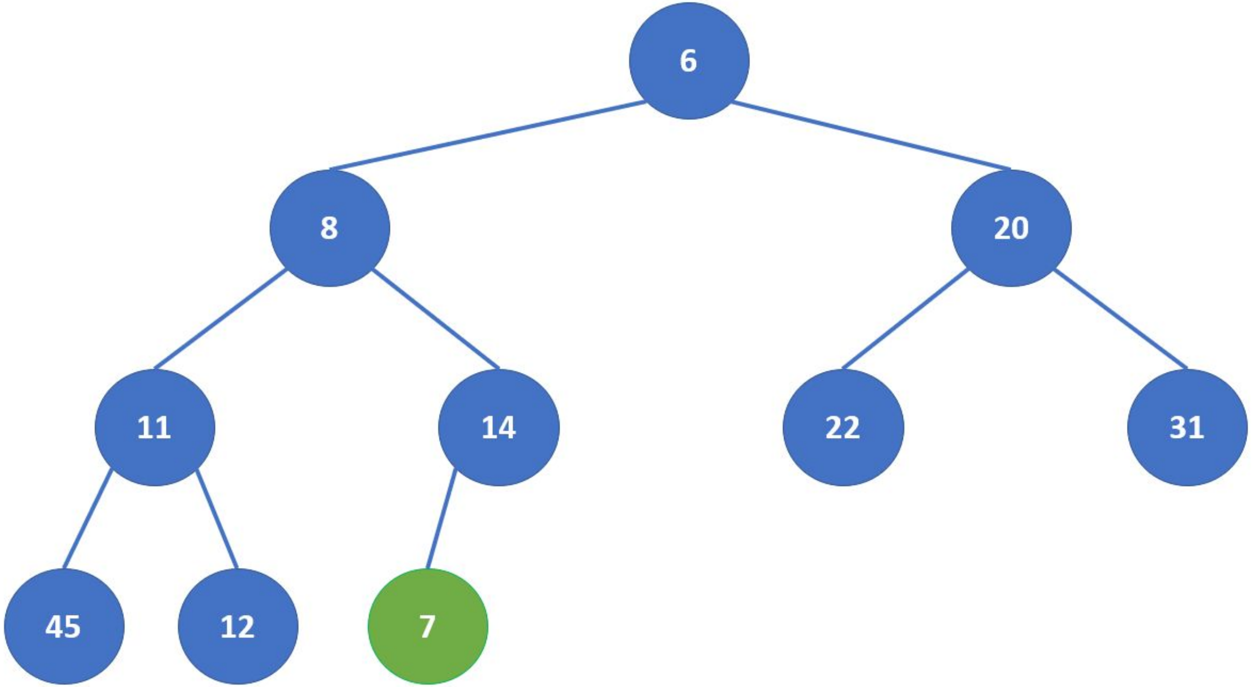
Minimum Priority Queue

- Minimum priority queue is used for ordering stamped messages.
- Binary heap data structure:
  - O(1) find-min, O(log(n)) insert, O(log(n)) remove
  - Complete binary tree
  - Parent Key <= Child Keys

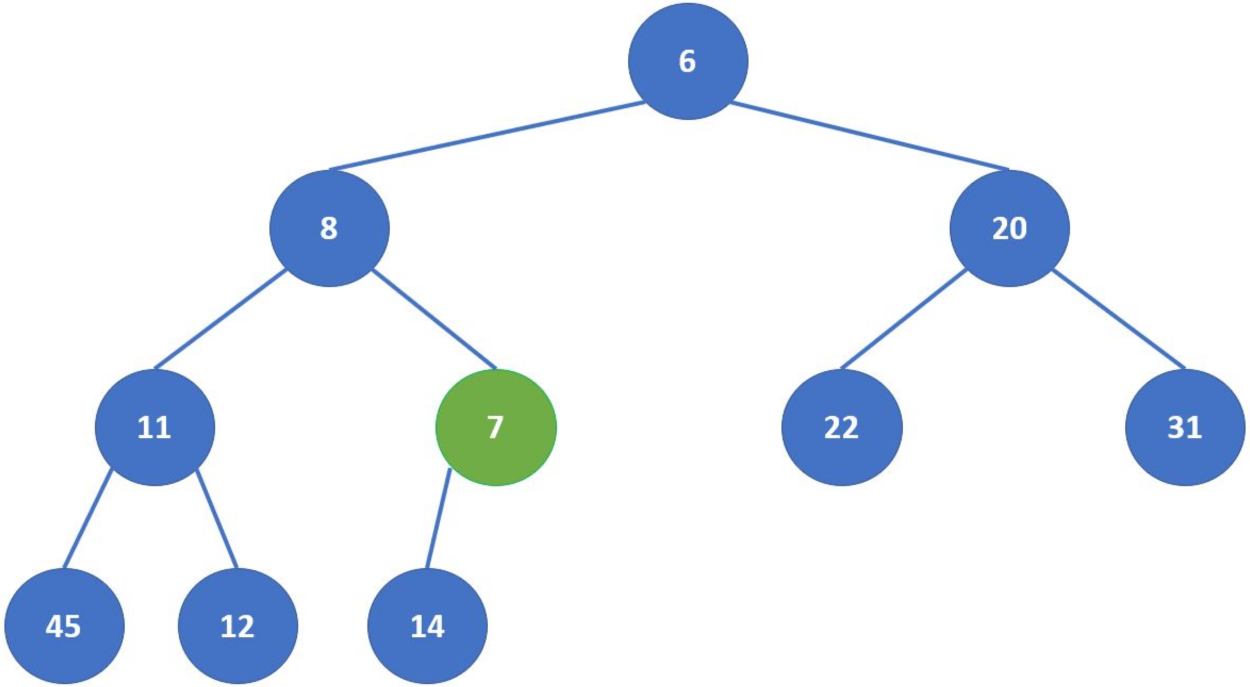


0	1	2	3	4	5	6	7	8	9	10	11	12	13
6	8	20	11	14	22	31	45	12					

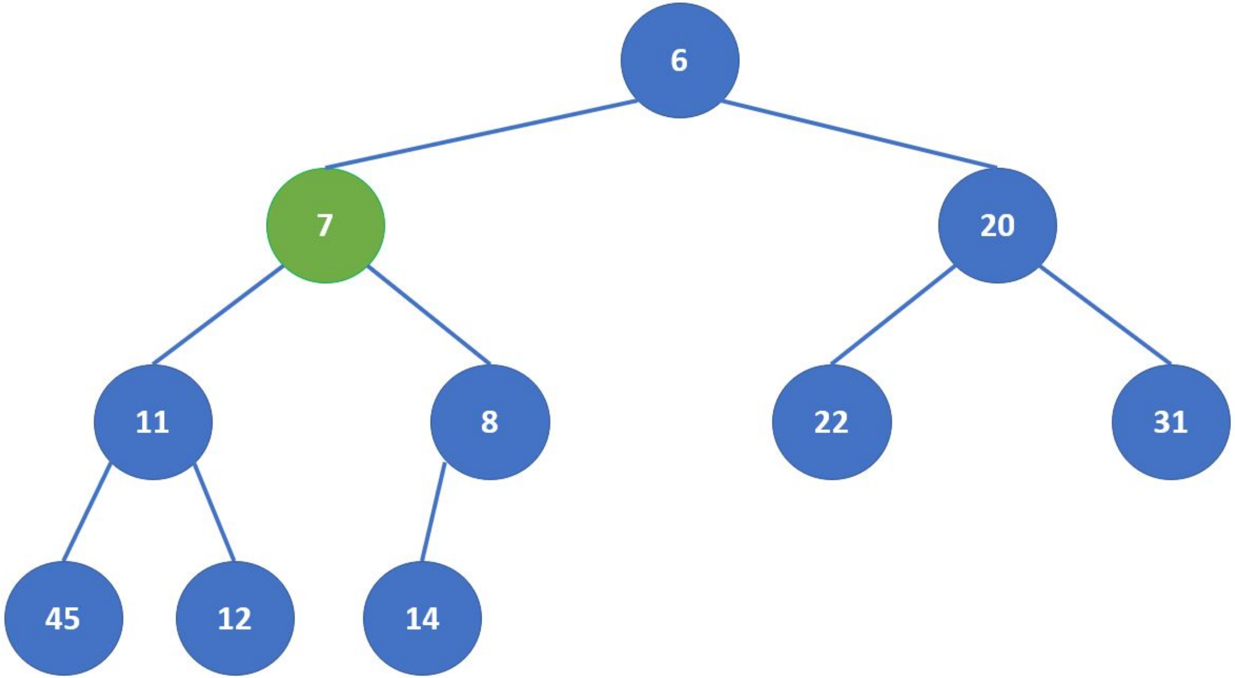
Minimum Priority Queue (Insert)



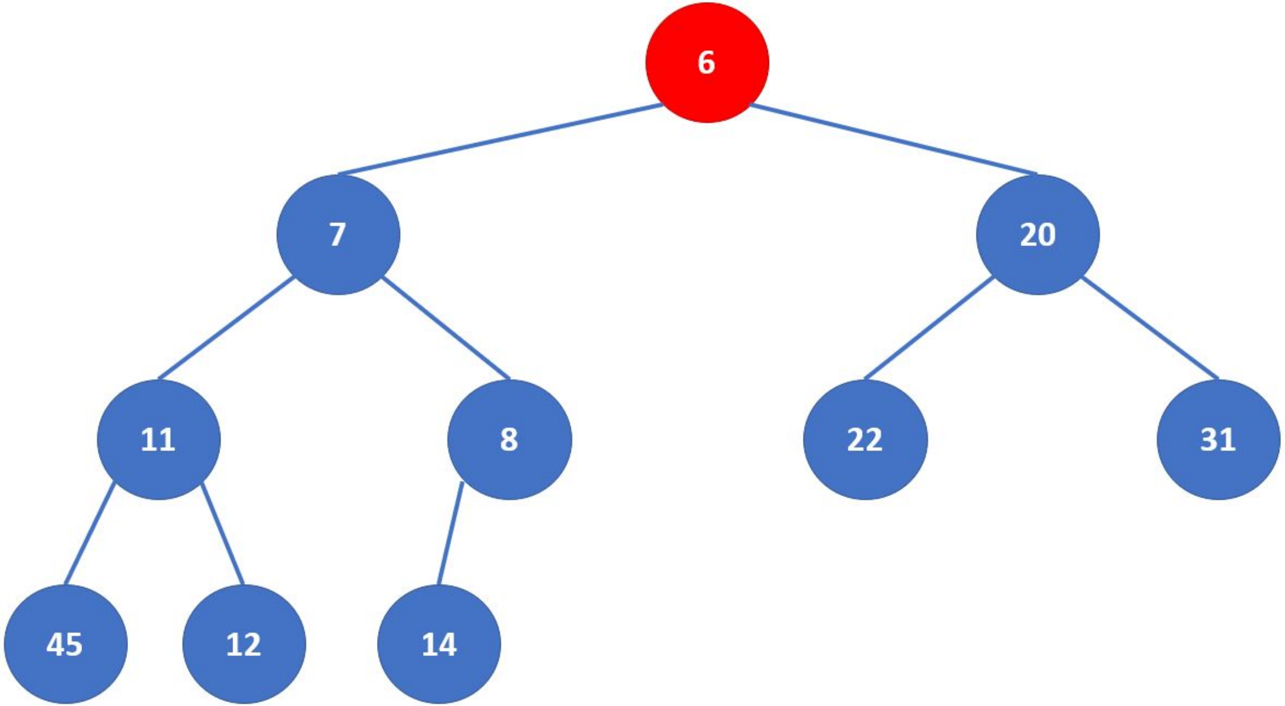
Minimum Priority Queue (Insert)



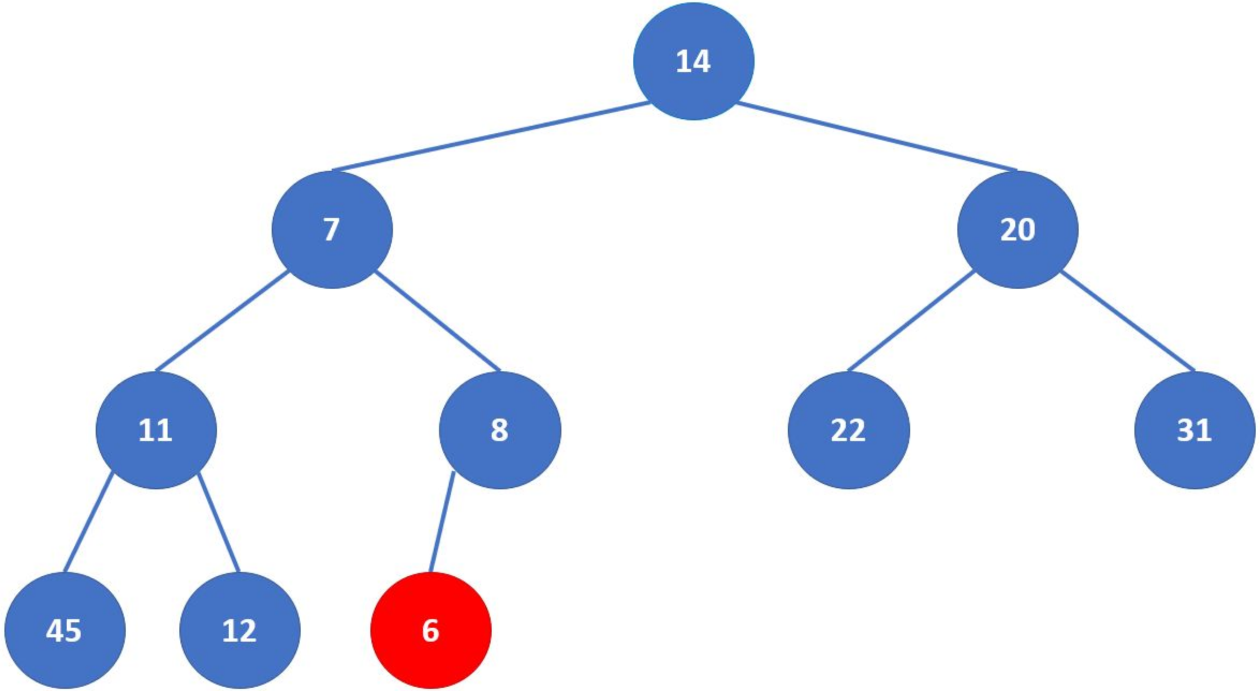
Minimum Priority Queue (Insert)



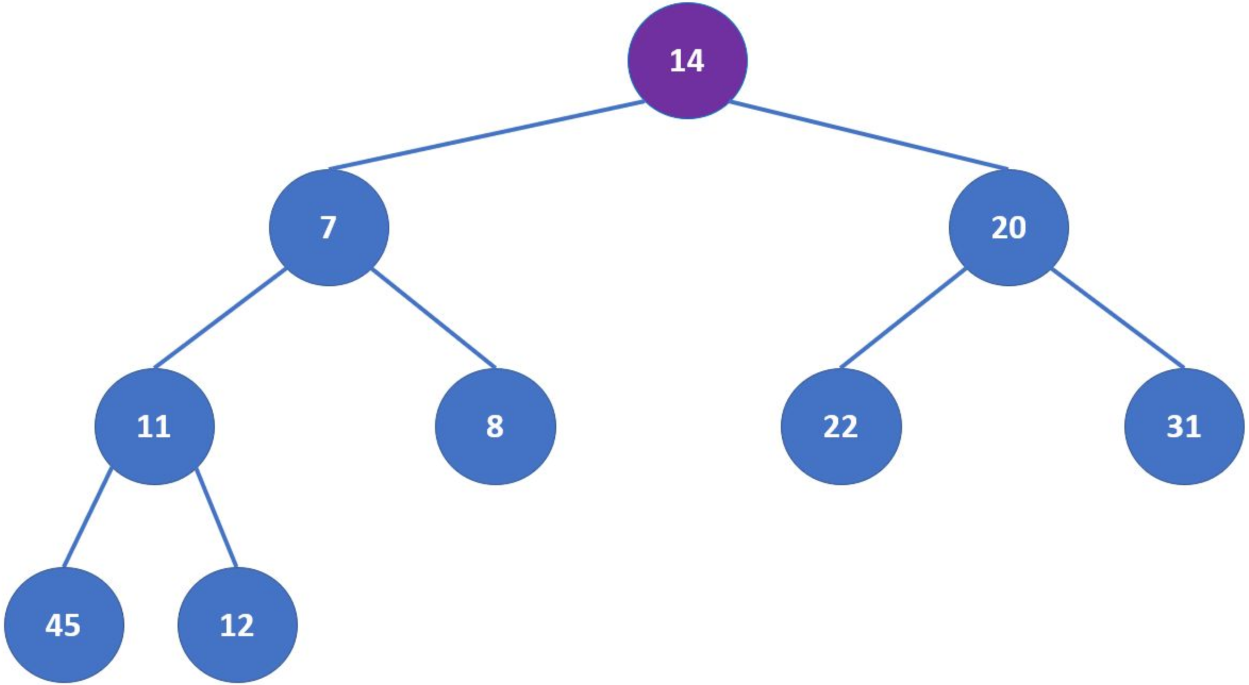
Minimum Priority Queue (Remove)



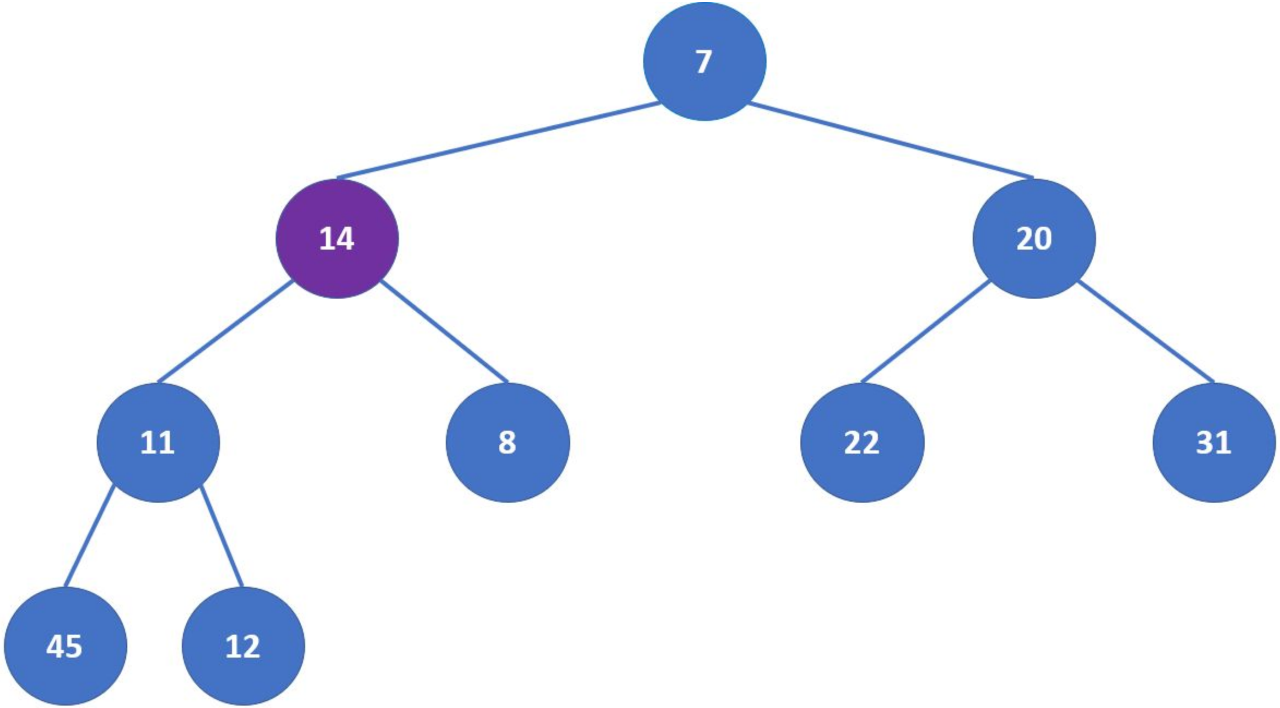
Minimum Priority Queue (Remove)



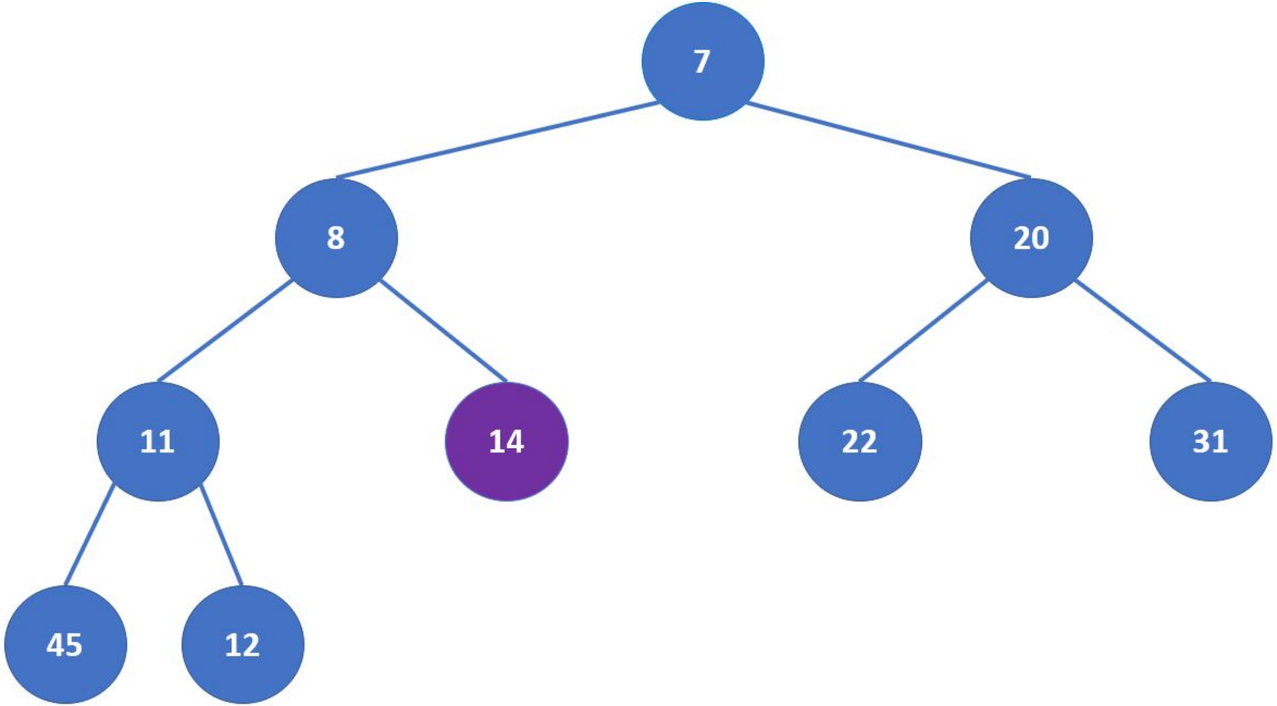
Minimum Priority Queue (Remove)



Minimum Priority Queue (Remove)

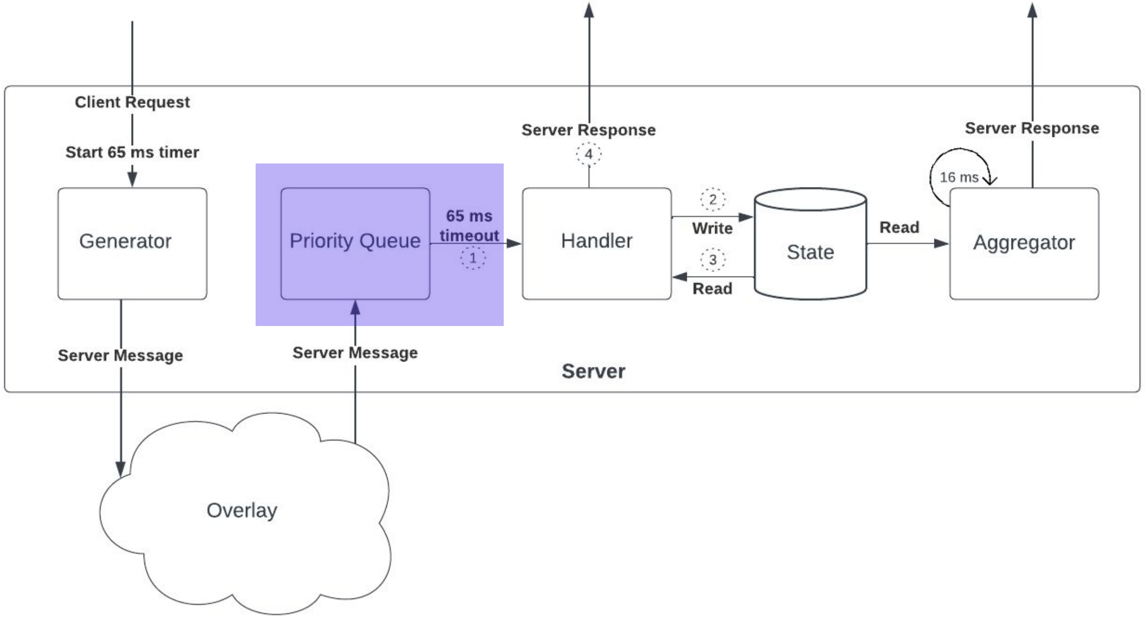


Minimum Priority Queue (Remove)

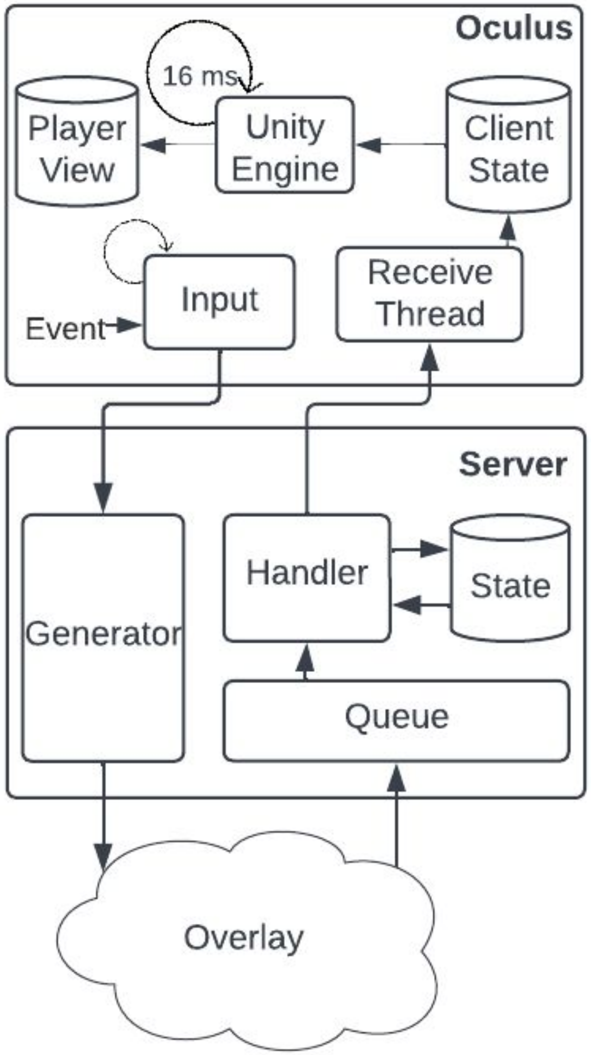


Synchronous Delivery

- 1 ms loop checks priority queue for new requests to be processed
    - All messages with timestamps older than 65 ms are handled
- Queue messages are ordered by:
    1) timestamp (us) given at ingest server
    2) message digest
    3) message size
    4) literal message bytes



Client, Server, Overlay



Client, Server, Spines

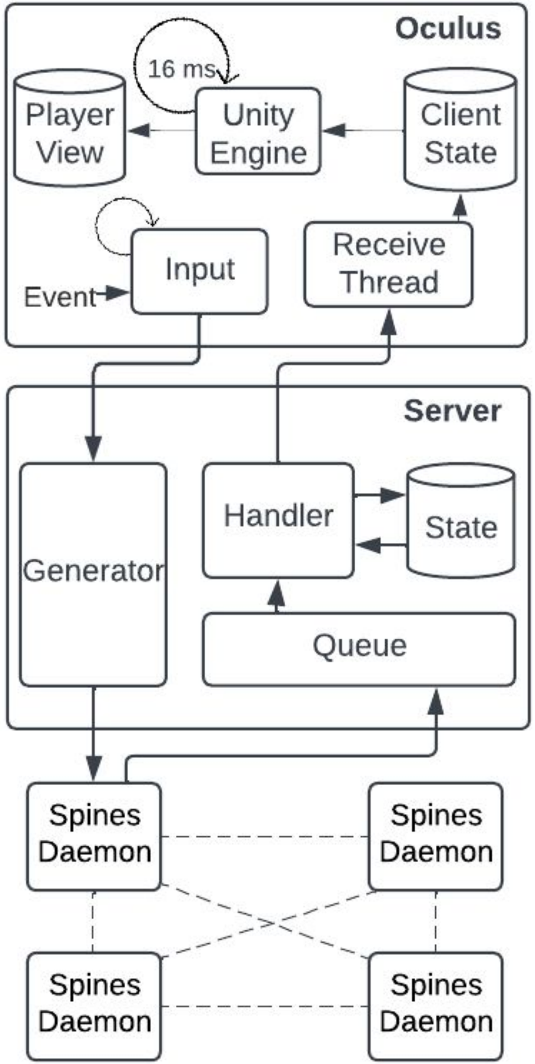


TABLE OF CONTENTS

01 Background

Virtual Reality
Oculus Quest 2 and Demo
Problem Introduction
Problem Approach

02 Game Development

Unity Game Engine
Game Objects
XR Interactive Toolkit
Control Flow and Scripts

03 Singleplayer & Multiplayer

04 Communication Architecture

Message Structures
Protocol Buffers
Synchronous Delivery

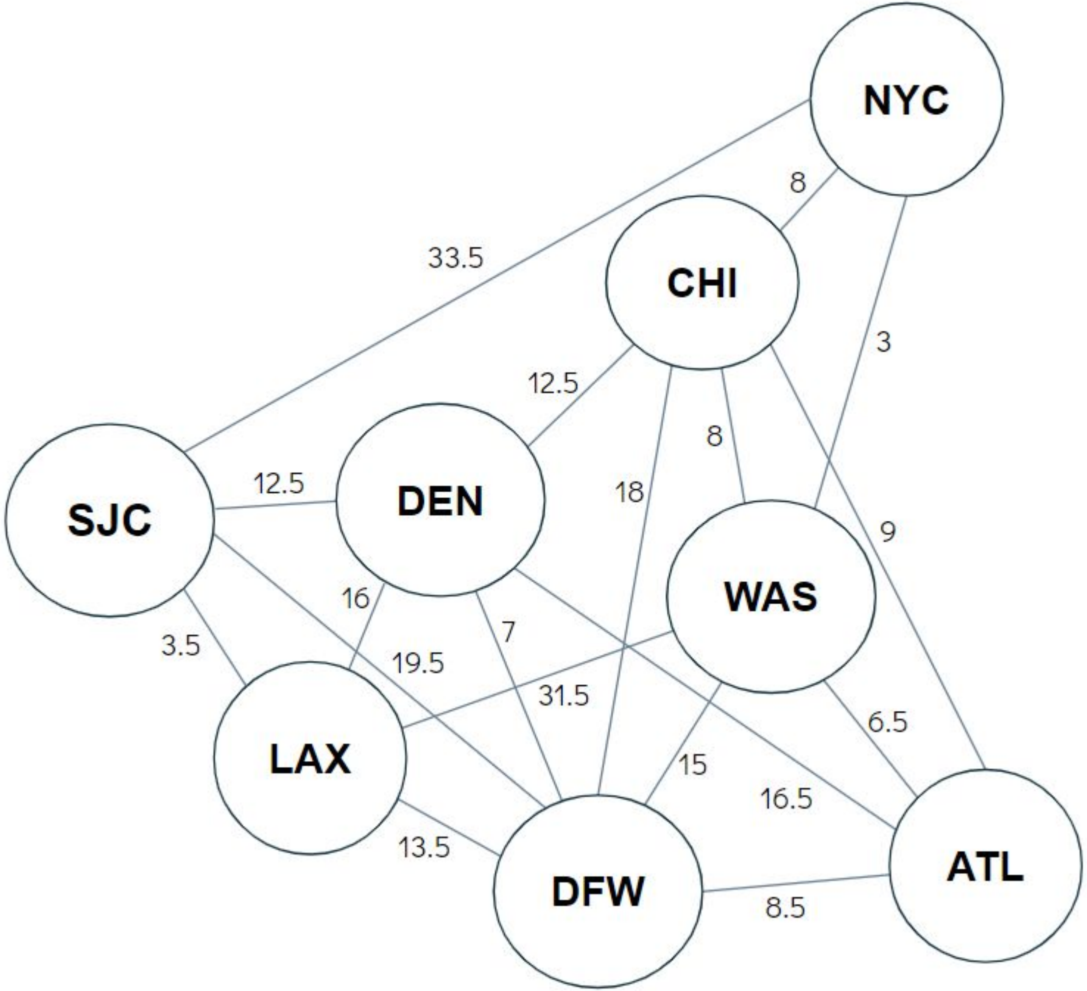
05 Spines

Introduction
Latency Graph & Reconstruction
Soft Real-time vs Source Based

06 Demonstration

Metrics & Statistics
Limitations & Improvements

Emulating Continental United States



About Spines

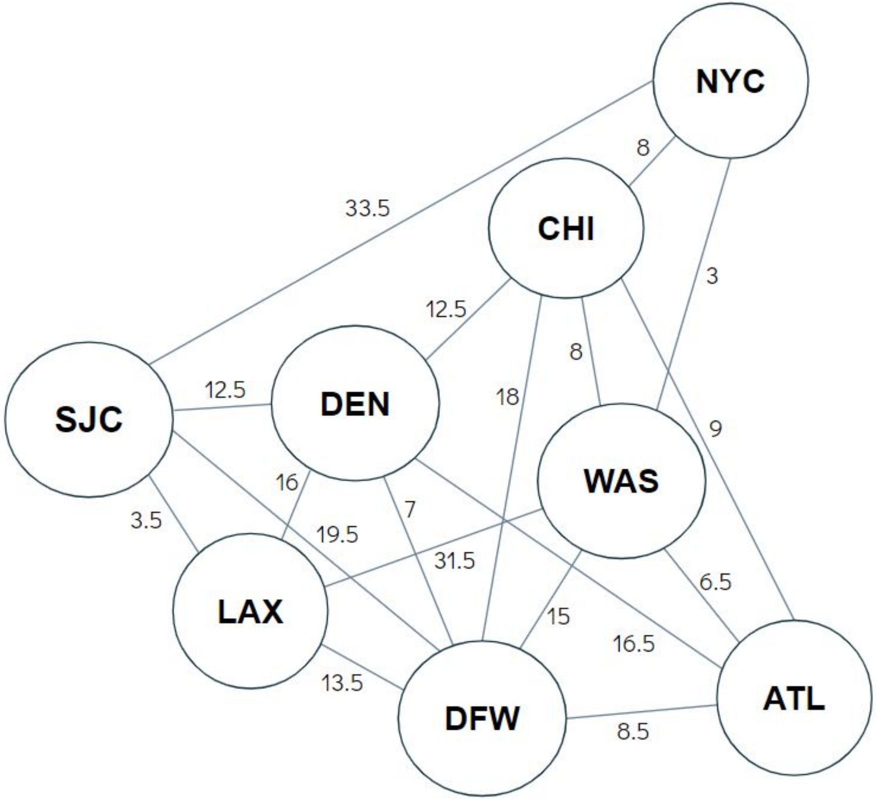
- Generic Infrastructure for dynamic, multi-hop network
    - Unicast & Multicast & Anycast
    - Automatic reconfiguration

- Instantiate network topology
    - Initialize each node and tell its direct neighbors
    - Set bi-directional links between neighbors with bandwidth, latency, loss rate, and burst rate information
    - Spines will compose the latency graph and learn the best routes from each node to any other nodes

More about Spines Infrastructure at Spines.org & DSN_Lab @ JHU

Spines Overlay

- Link Protocols
  - `UDP_LINKS`
  - `RELIABLE_LINKS`
  - `SOFT_REALTIME_LINKS`
  - `INTRUSION_TOL_LINKS`



Spines Daemons

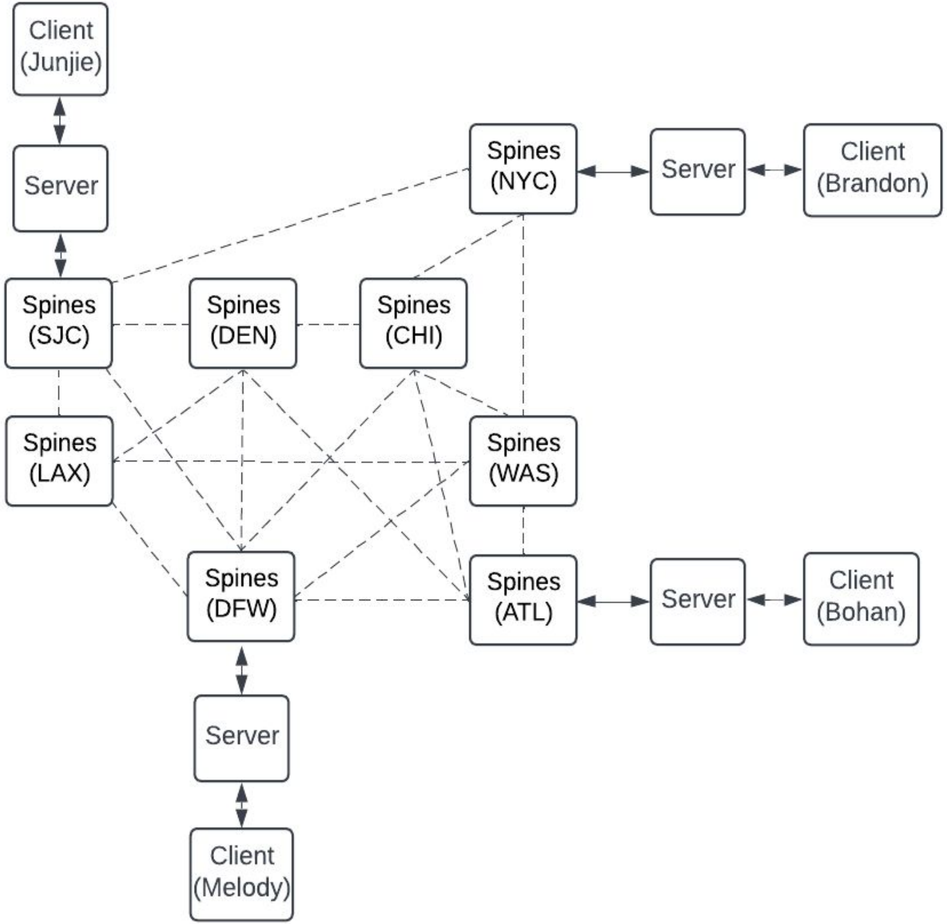


TABLE OF CONTENTS

01 Background

Virtual Reality
Oculus Quest 2 and Demo
Problem Introduction
Problem Approach

02 Game Development

Unity Game Engine
Game Objects
XR Interactive Toolkit
Control Flow and Scripts

03 Singleplayer & Multiplayer

04 Communication Architecture

Message Structures
Protocol Buffers
Synchronous Delivery

05 Spines

Introduction
Latency Graph & Reconstruction
Soft Real-time vs Source Based

06 Demonstration

Metrics & Statistics
Limitations & Improvements

Demonstration

User Interface

Select name and location

Join lobby

Statistics Panel

View present players

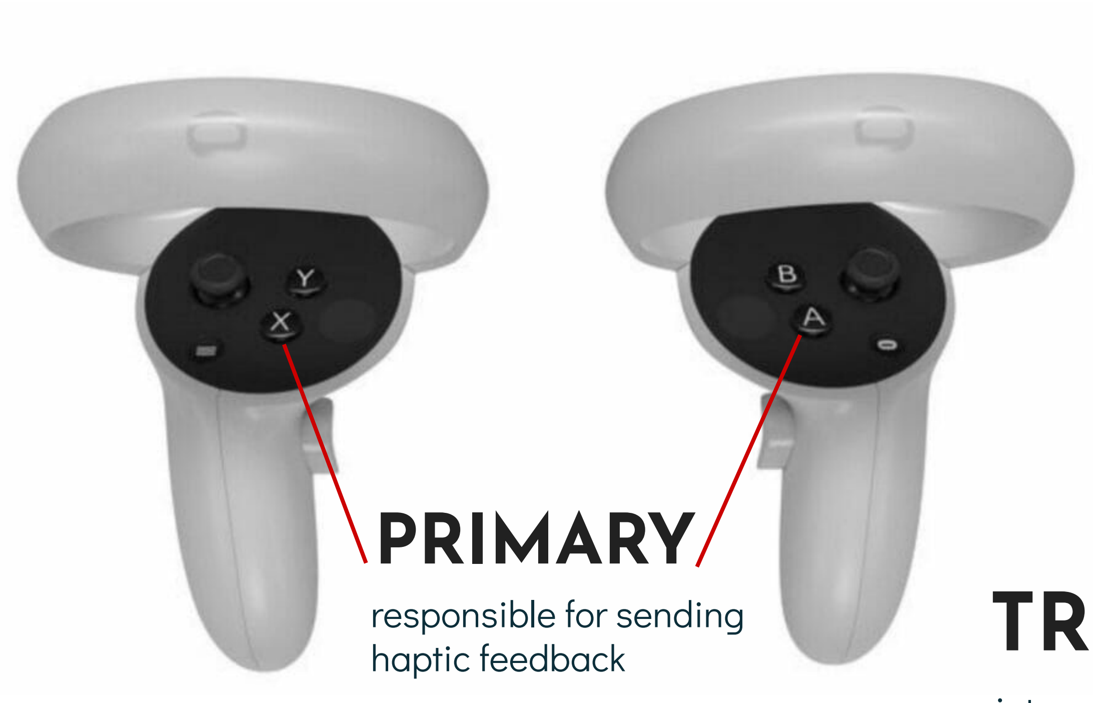
View ping times with server

Movement

Controlled by left joystick

Revolving Sphere

Indicates if the server is running



PRIMARY

responsible for sending haptic feedback

RIGHT sends to all players in the lobby

TRIGGER

interact with buttons and objects within range of raycasters

Haptics Cylinder

Right primary button sends haptic request

Right controller rumbles locally immediately after sending request

All players' left controllers rumble in synchrony 65 ms after any haptic request

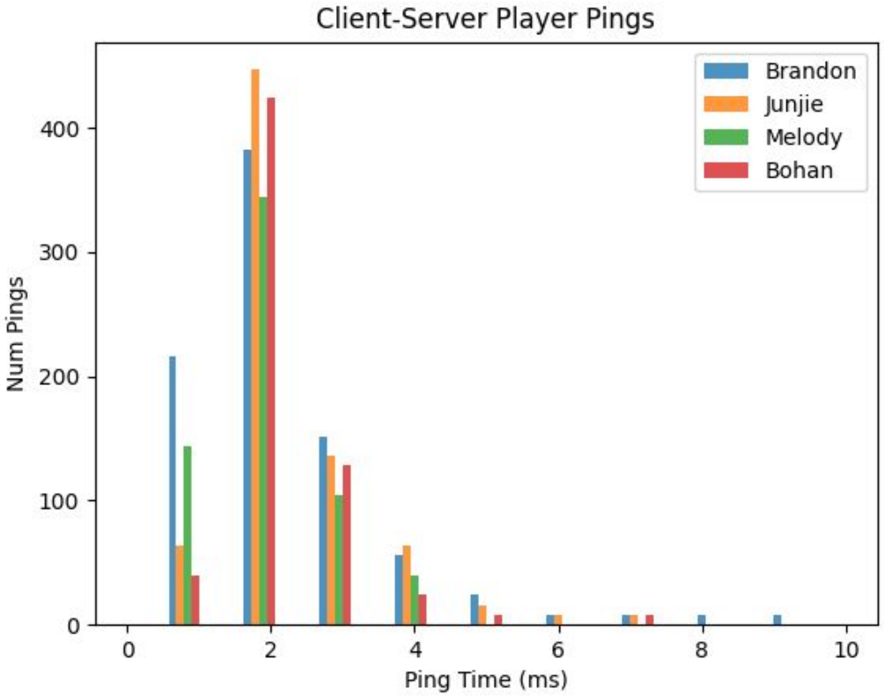
Interactable Sphere

Right trigger button claims possession

Right joystick moves claimed sphere forward/backward

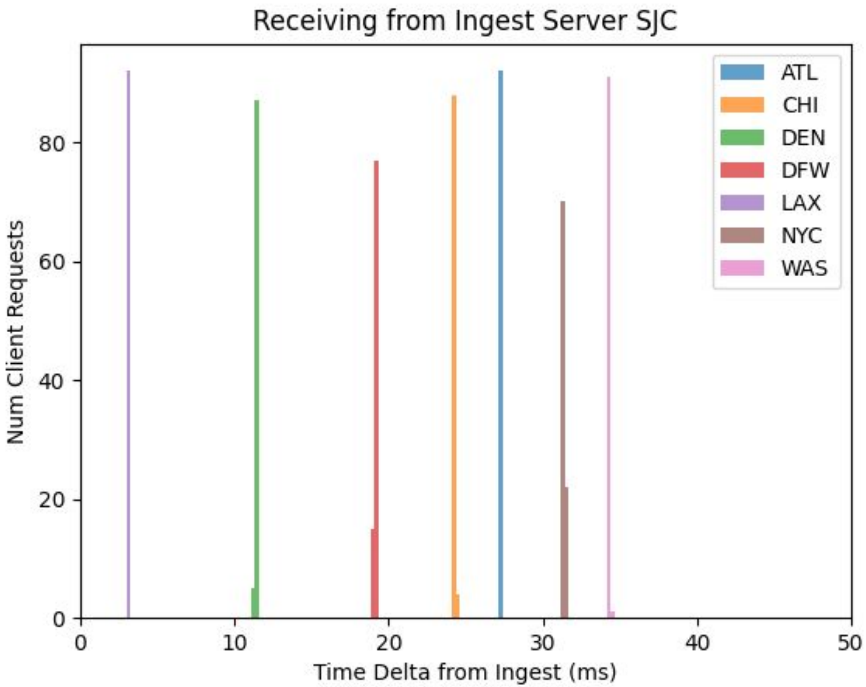
Client-Server Ping Times

PLAYER	DATA CENTER	PING (ms)	STANDARD DEVIATION (ms)
JUNJIE	SJC	2.4	1.0
MELODY	DFW	2.1	0.8
BOHAN	ATL	2.3	0.9
BRANDON	NYC	2.3	1.4

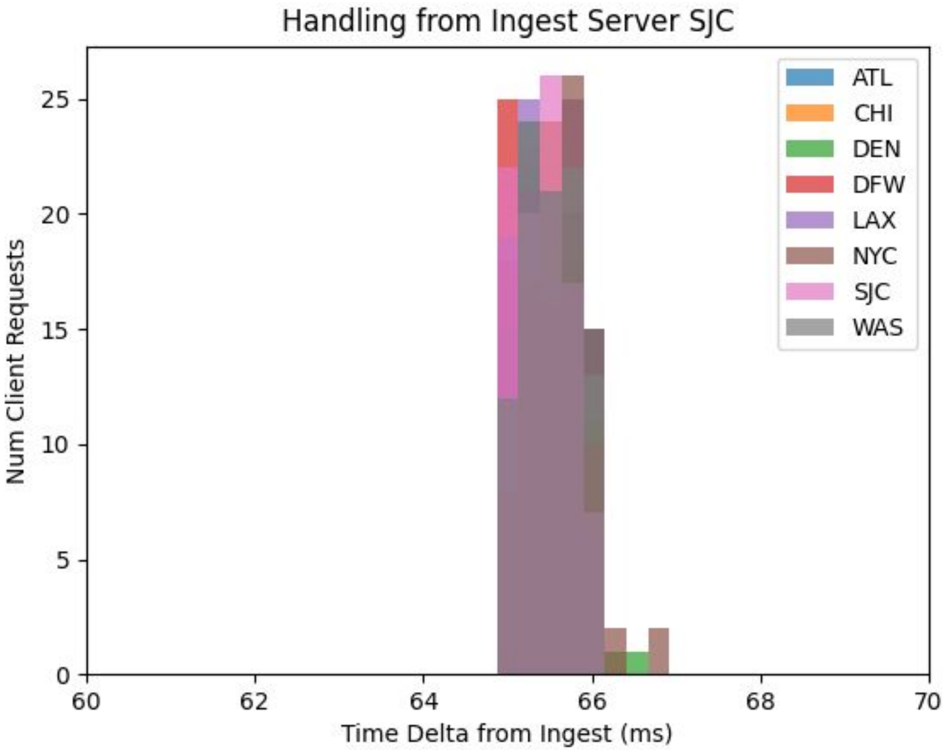


Receiving Player Messages (SJC)

DATA CENTER	EXPECTED LATENCY (ms)	OBSERVED LATENCY (ms)	STANDARD DEVIATION (ms)
SJC	0	**0**	0
DFW	17	**19.1**	0.05
ATL	25.5	**27.3**	0.05
NYC	33	**34.3**	0.04

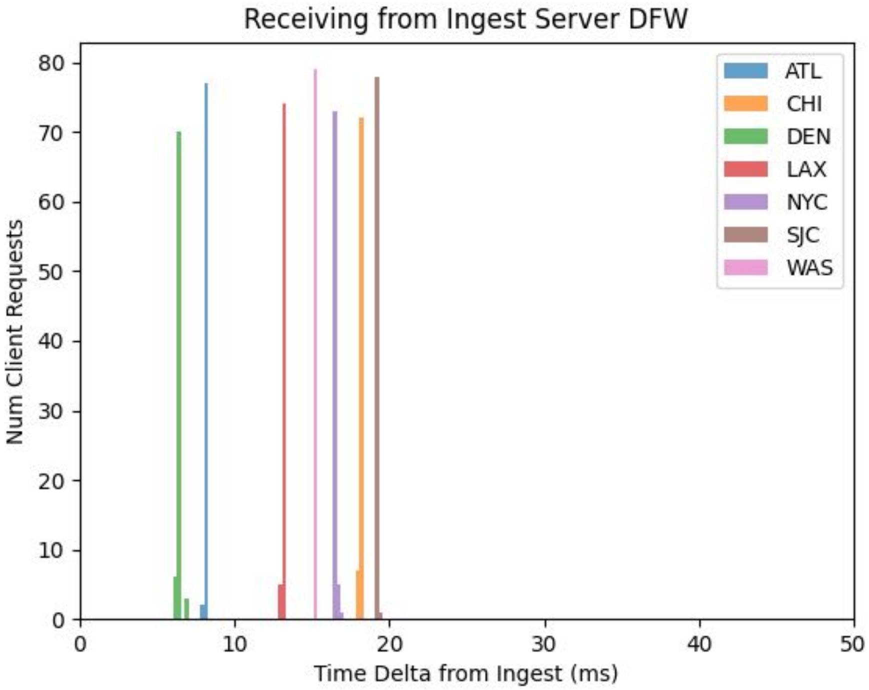


Delivering Player Messages (SJC)

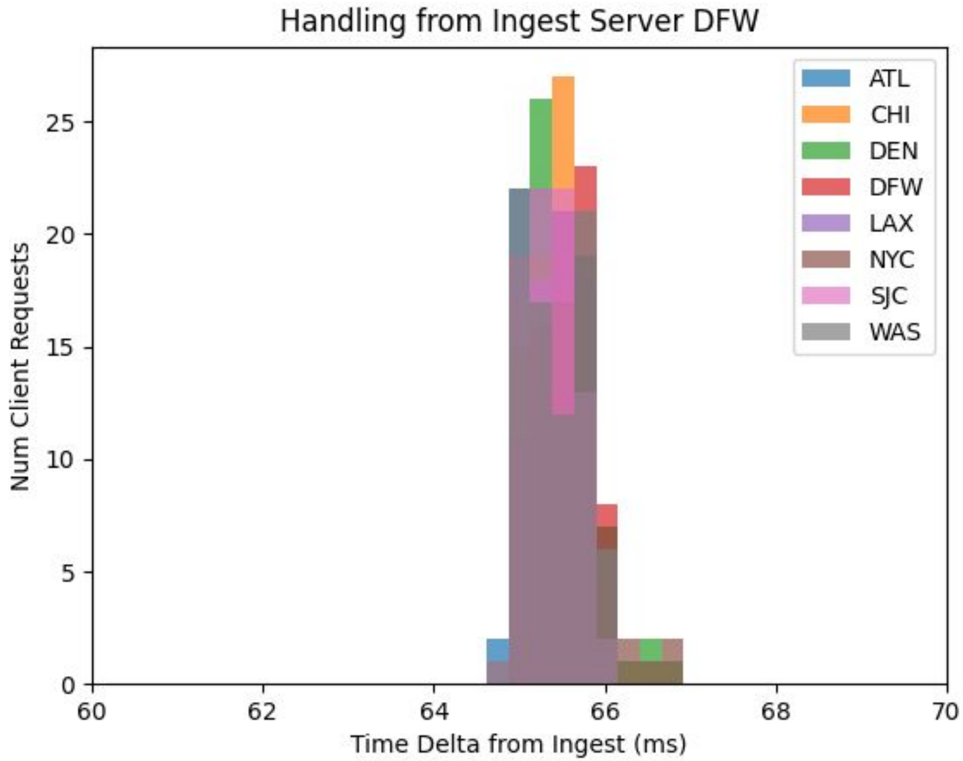


Receiving Player Messages (DFW)

DATA CENTER	EXPECTED LATENCY (ms)	OBSERVED LATENCY (ms)	STANDARD DEVIATION (ms)
SJC	17	**19.1**	0.05
DFW	0	**0**	0
ATL	8.5	**8.1**	0.04
NYC	18	**20.0**	0.10

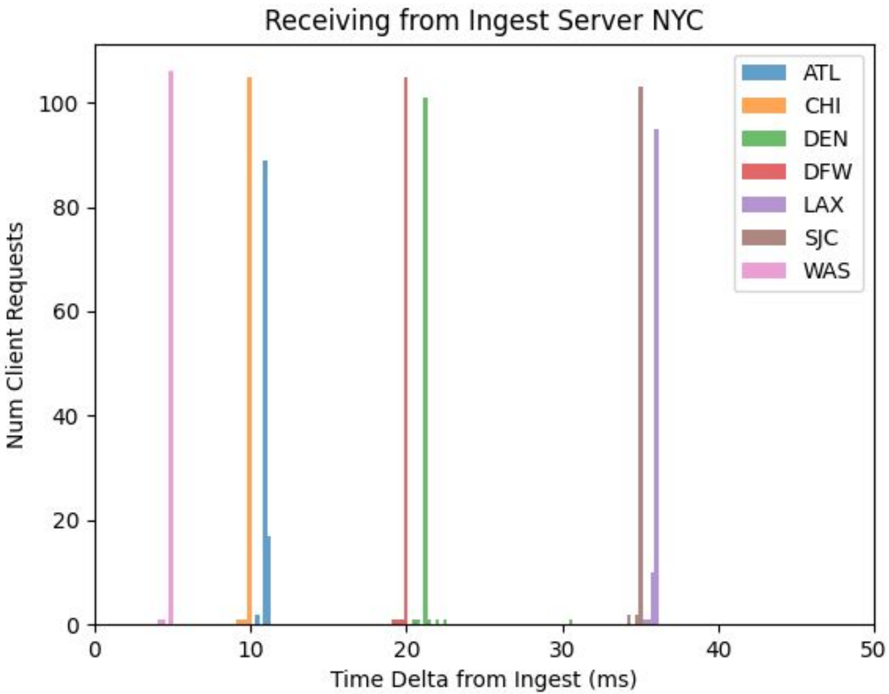


Delivering Player Messages (DFW)

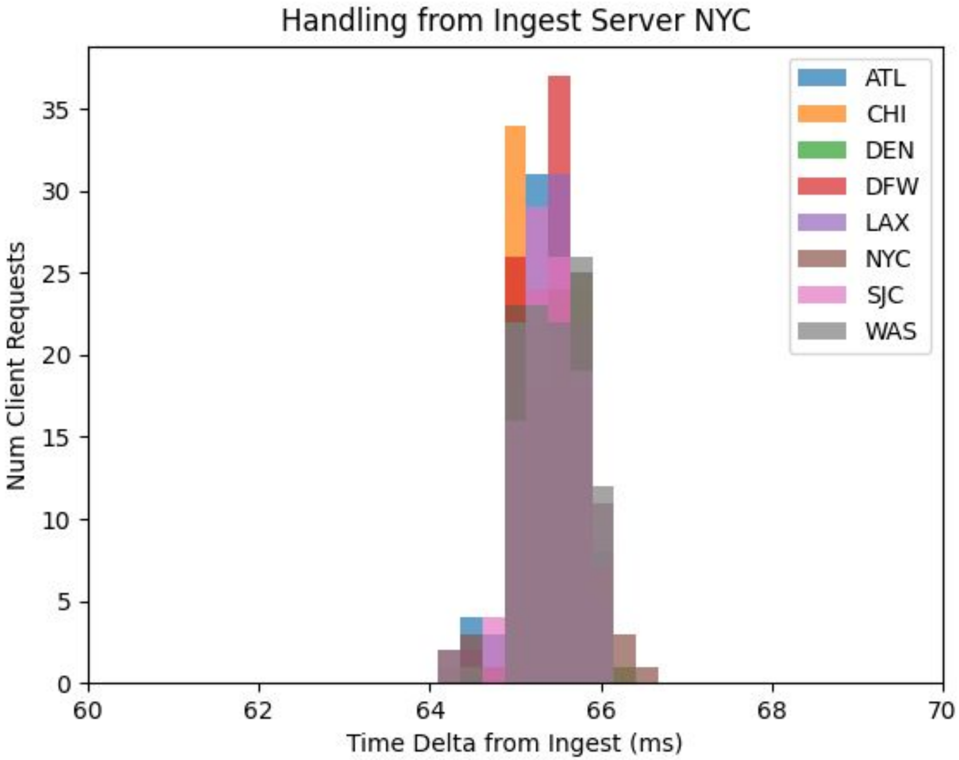


Receiving Player Messages (NYC)

DATA CENTER	EXPECTED LATENCY (ms)	OBSERVED LATENCY (ms)	STANDARD DEVIATION (ms)
SJC	33	**35.0**	0.09
DFW	18	**20.0**	0.10
ATL	9.5	**11.0**	0.10
NYC	0	**0**	0

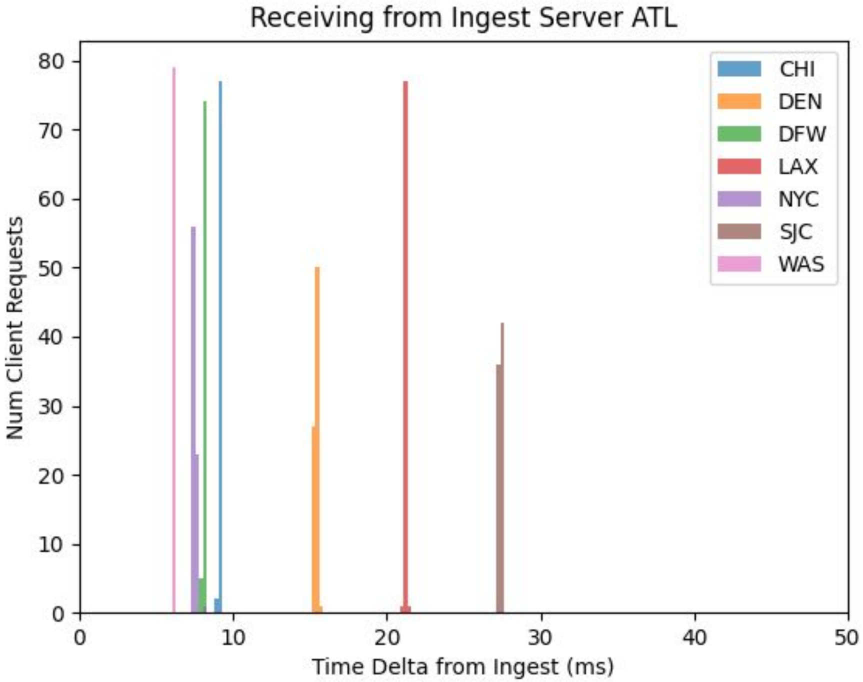


Delivering Player Messages (NYC)

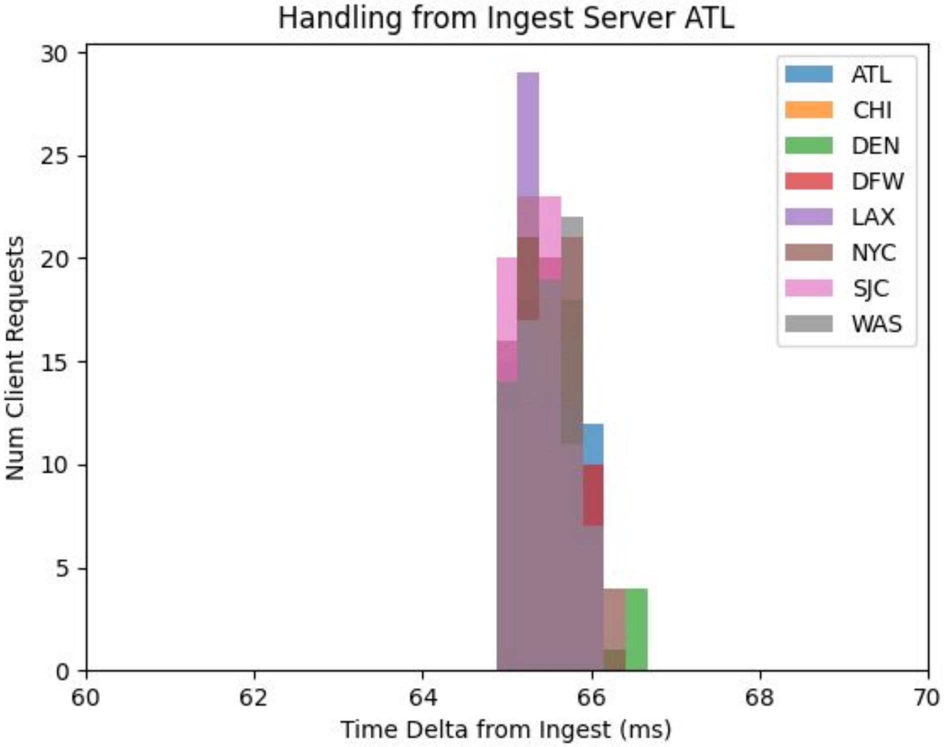


Receiving Player Messages (ATL)

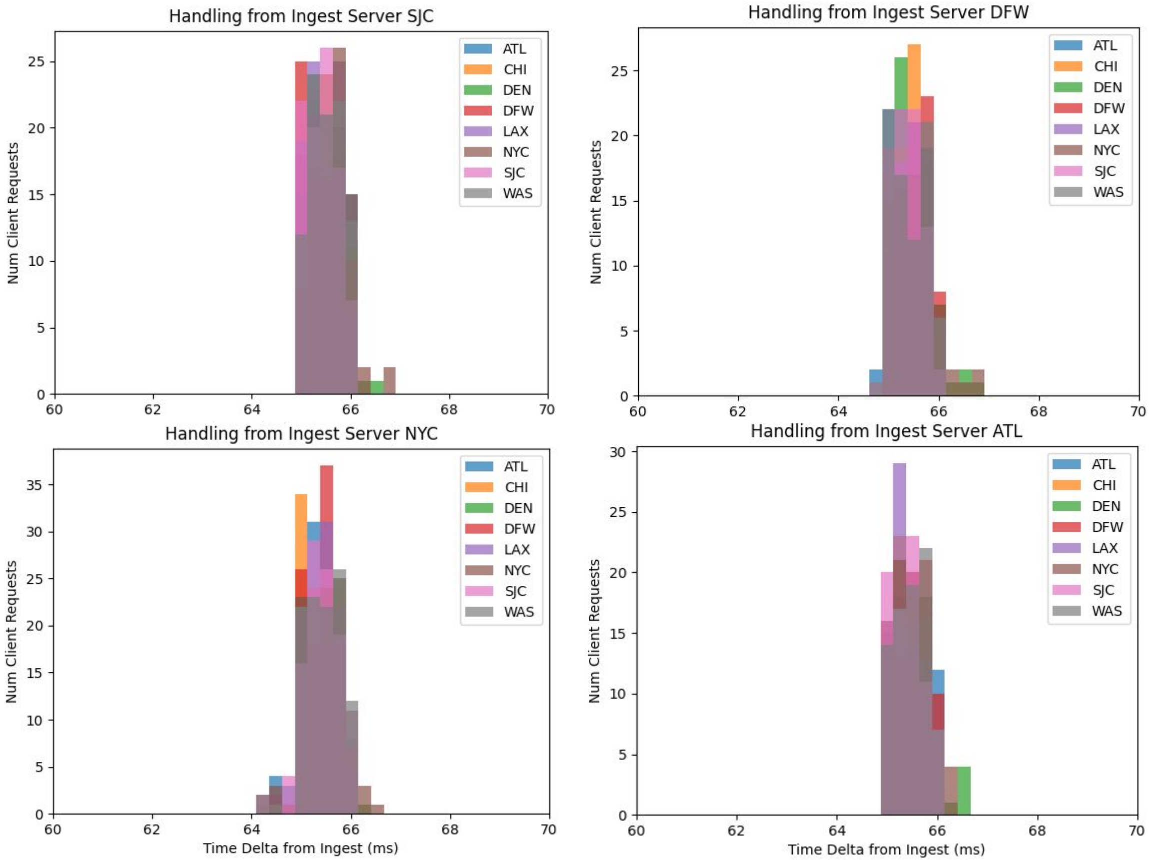
DATA CENTER	EXPECTED LATENCY (ms)	OBSERVED LATENCY (ms)	STANDARD DEVIATION (ms)
SJC	25.5	**27.3**	0.05
DFW	8.5	**8.1**	0.04
ATL	0	**0**	0
NYC	9.5	**11.0**	0.09



Delivering Player Messages (ATL)



Delivering Player Messages (All Servers)



LIMITATION

Existing delay between server delivery and client delivery

IMPROVEMENT

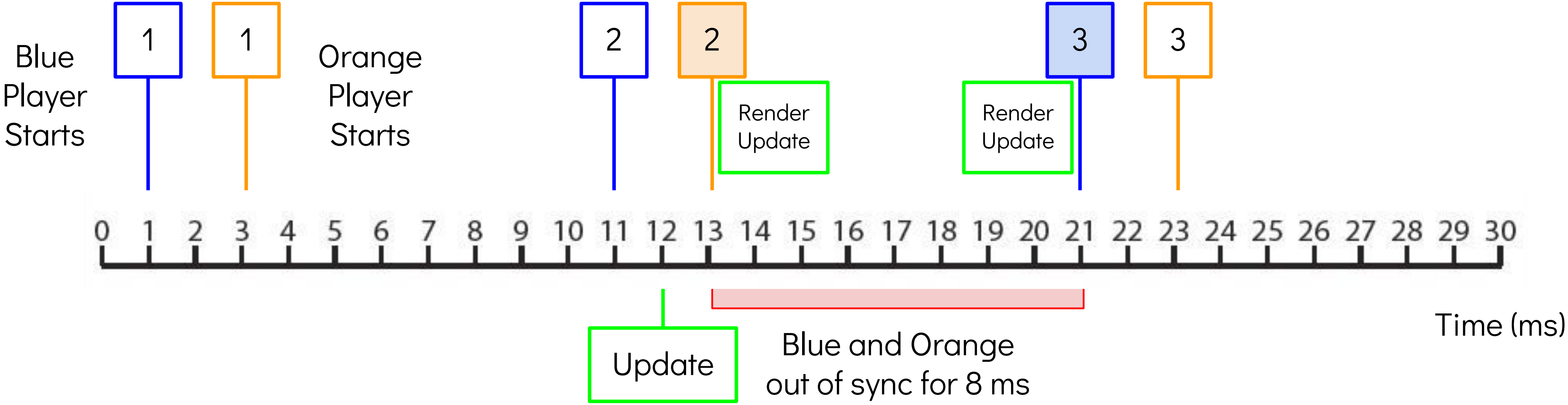
Implement the server to deliver messages immediately while the client handles the synchronization delay. All clients would run a clock synchronization algorithm

LIMITATION

Clients render at different offset times

IMPROVEMENT

Force the Oculus to skip a frame in order to synchronize frame rendering



Blue Player Starts

Orange Player Starts

Render Update

Render Update

Update

Blue and Orange out of sync for 8 ms

Time (ms)

LIMITATION

Minimum priority queue data structure runs in *log(n)*

IMPROVEMENT

Use a bucketed array of size 1000 instead, which would have *O(1)* insert and lookup times.

LIMITATION

Packet losses are not handled

IMPROVEMENT

Implement server state reconciliation, in which servers periodically send states to one another about the players in the lobby and the state of the world objects

Questions?