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)
- 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

<table>
<thead>
<tr>
<th>01</th>
<th>Background</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Virtual Reality</td>
</tr>
<tr>
<td></td>
<td>Oculus Quest 2 and Demo</td>
</tr>
<tr>
<td></td>
<td>Problem Introduction</td>
</tr>
<tr>
<td></td>
<td>Problem Approach</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>02</th>
<th>Game Development</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Unity Game Engine</td>
</tr>
<tr>
<td></td>
<td>Game Objects</td>
</tr>
<tr>
<td></td>
<td>XR Interactive Toolkit</td>
</tr>
<tr>
<td></td>
<td>Control Flow and Scripts</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>03</th>
<th>Singleplayer &amp; Multiplayer</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>04</th>
<th>Communication Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Message Structures</td>
</tr>
<tr>
<td></td>
<td>Protocol Buffers</td>
</tr>
<tr>
<td></td>
<td>Synchronous Delivery</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>05</th>
<th>Spines</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Introduction</td>
</tr>
<tr>
<td></td>
<td>Latency Graph &amp; Reconstruction</td>
</tr>
<tr>
<td></td>
<td>Soft Real-time vs Source Based</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>06</th>
<th>Demonstration</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Metrics &amp; Statistics</td>
</tr>
<tr>
<td></td>
<td>Limitations &amp; Improvements</td>
</tr>
</tbody>
</table>
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
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.
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
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 (Protobuf)
Client-Server Communication (Protobuf)

- Language independent
- Backwards compatibility/Implementation advantage
- Good performance
Client-Server Communication (Protobuf)

Protobuf (C#)

```csharp
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)

```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

- **Message**
  - Sender ID
  - Type
  - Data

- **RPC**
- **Syn Request**
- **Continuous**
  - Headset
  - Left Controller
  - Right Controller

- **Syn Response**
- **Presence**
- **World**
  - Avatars
  - Items
Multiplayer + Single Server

Diagram:

Client Request → Generator → Handler → State

1. Server Message
2. Read
3. Write
4. Server Response
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

**Rovolving 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 $\rightarrow$ Flooding
- 100 players in one game

Limited computing power $\rightarrow$ Efficiency matters
- 116 million people played Fortnite on iOS devices.

Updates not needed $\rightarrow$ 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 $\leq$ Child Keys
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](http://Spines.org) & [DSN Lab @ JHU](http://DSN_Lab@JHU)
Spines Overlay

- Link Protocols
  - UDP_LINKS
  - RELIABLE_LINKS
  - SOFT_REALTIME_LINKS
  - INTRUSION_TOL_LINKS
Spines Daemons
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>01</th>
<th>Background</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Virtual Reality</td>
</tr>
<tr>
<td></td>
<td>Oculus Quest 2 and Demo</td>
</tr>
<tr>
<td></td>
<td>Problem Introduction</td>
</tr>
<tr>
<td></td>
<td>Problem Approach</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>02</th>
<th>Game Development</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Unity Game Engine</td>
</tr>
<tr>
<td></td>
<td>Game Objects</td>
</tr>
<tr>
<td></td>
<td>XR Interactive Toolkit</td>
</tr>
<tr>
<td></td>
<td>Control Flow and Scripts</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>03</th>
<th>Singleplayer &amp; Multiplayer</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>04</th>
<th>Communication Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Message Structures</td>
</tr>
<tr>
<td></td>
<td>Protocol Buffers</td>
</tr>
<tr>
<td></td>
<td>Synchronous Delivery</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>05</th>
<th>Spines</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Introduction</td>
</tr>
<tr>
<td></td>
<td>Latency Graph &amp; Reconstruction</td>
</tr>
<tr>
<td></td>
<td>Soft Real-time vs Source Based</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>06</th>
<th>Demonstration</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Metrics &amp; Statistics</td>
</tr>
<tr>
<td></td>
<td>Limitations &amp; Improvements</td>
</tr>
</tbody>
</table>
## Demonstration

<table>
<thead>
<tr>
<th>User Interface</th>
<th>Statistics Panel</th>
<th>Movement</th>
<th>Revolving Sphere</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select name and location</td>
<td>View present players</td>
<td>Controlled by left joystick</td>
<td>Indicates if the server is running</td>
</tr>
<tr>
<td>Join lobby</td>
<td>View ping times with server</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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
<table>
<thead>
<tr>
<th>PLAYER</th>
<th>DATA CENTER</th>
<th>PING (ms)</th>
<th>STANDARD DEVIATION (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>JUNJIE</td>
<td>SJC</td>
<td>2.4</td>
<td>1.0</td>
</tr>
<tr>
<td>MELODY</td>
<td>DFW</td>
<td>2.1</td>
<td>0.8</td>
</tr>
<tr>
<td>BOHAN</td>
<td>ATL</td>
<td>2.3</td>
<td>0.9</td>
</tr>
<tr>
<td>BRANDON</td>
<td>NYC</td>
<td>2.3</td>
<td>1.4</td>
</tr>
</tbody>
</table>
## Receiving Player Messages (SJC)

<table>
<thead>
<tr>
<th>DATA CENTER</th>
<th>EXPECTED LATENCY (ms)</th>
<th>OBSERVED LATENCY (ms)</th>
<th>STANDARD DEVIATION (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SJC</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DFW</td>
<td>17</td>
<td>19.1</td>
<td>0.05</td>
</tr>
<tr>
<td>ATL</td>
<td>25.5</td>
<td>27.3</td>
<td>0.05</td>
</tr>
<tr>
<td>NYC</td>
<td>33</td>
<td>34.3</td>
<td>0.04</td>
</tr>
</tbody>
</table>

![Graph showing data](image-url)
Delivering Player Messages (SJC)
## Receiving Player Messages (DFW)

<table>
<thead>
<tr>
<th>DATA CENTER</th>
<th>EXPECTED LATENCY (ms)</th>
<th>OBSERVED LATENCY (ms)</th>
<th>STANDARD DEVIATION (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SJC</td>
<td>17</td>
<td>19.1</td>
<td>0.05</td>
</tr>
<tr>
<td>DFW</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ATL</td>
<td>8.5</td>
<td>8.1</td>
<td>0.04</td>
</tr>
<tr>
<td>NYC</td>
<td>18</td>
<td>20.0</td>
<td>0.10</td>
</tr>
</tbody>
</table>

![Graph showing receiving from Ingest Server DFW]
Delivering Player Messages (DFW)
## Receiving Player Messages (NYC)

<table>
<thead>
<tr>
<th>DATA CENTER</th>
<th>EXPECTED LATENCY (ms)</th>
<th>OBSERVED LATENCY (ms)</th>
<th>STANDARD DEVIATION (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SJC</td>
<td>33</td>
<td>35.0</td>
<td>0.09</td>
</tr>
<tr>
<td>DFW</td>
<td>18</td>
<td>20.0</td>
<td>0.10</td>
</tr>
<tr>
<td>ATL</td>
<td>9.5</td>
<td>11.0</td>
<td>0.10</td>
</tr>
<tr>
<td>NYC</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Receiving from Ingest Server NYC

![Graph showing receiving from Ingest Server NYC](image)

- ATL
- CHI
- DEN
- DFW
- LAX
- SJC
- WAS

---

67
Delivering Player Messages (NYC)
## Receiving Player Messages (ATL)

<table>
<thead>
<tr>
<th>DATA CENTER</th>
<th>EXPECTED LATENCY (ms)</th>
<th>OBSERVED LATENCY (ms)</th>
<th>STANDARD DEVIATION (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SJC</td>
<td>25.5</td>
<td>27.3</td>
<td>0.05</td>
</tr>
<tr>
<td>DFW</td>
<td>8.5</td>
<td>8.1</td>
<td>0.04</td>
</tr>
<tr>
<td>ATL</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>NYC</td>
<td>9.5</td>
<td>11.0</td>
<td>0.09</td>
</tr>
</tbody>
</table>

![Graph showing receiving from Ingest Server ATL](image)
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 and Orange out of sync for 8 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?