Prepared by group 46

GridClash

A Lightweight UDP Protocol

25 December, 2025

Team Members

Ahmed Lotfy

22P0251

Philopater Mina

22P0250

Yousif Salah

22P0232

Hams Hassan

22P0253

Noha El Sayed

2201431

Adham Kandil

22P0217

Introduction

The primary objective of the project was to achieve low-latency state synchronization using an authoritative server architecture over UDP, without relying on TCP.

Protocol Architecture & Binary Header Design (GCUP)

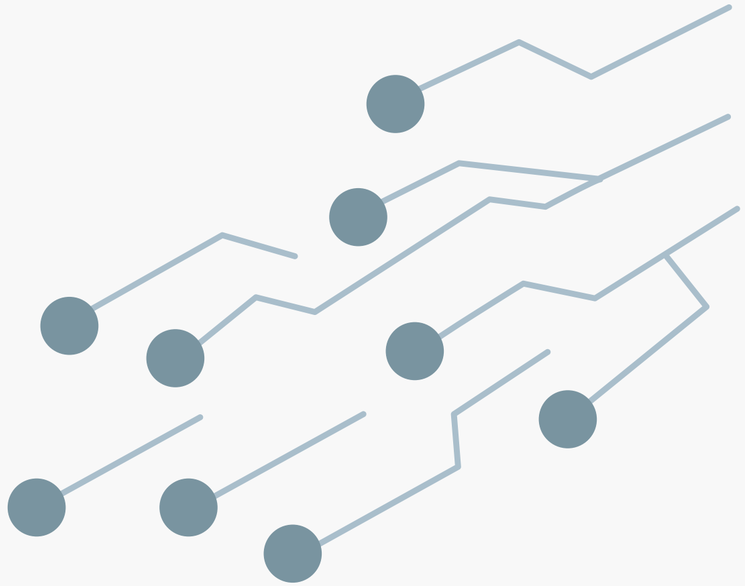
28-Byte Fixed Header Specification

Field	Description
protocol_id	4-byte ASCII identifier unique to your protocol
version	1 byte
msg_type	1 byte
snapshot_id	4 bytes
seq_num	4 bytes
server_timestamp	8 bytes
payload_len	2 bytes
checksum	4 bytes using CRC32

Client-Server Synchronization Logic

Authoritative Server Loop

The server broadcasts the full world state every ~50 ms, including the grid and player vectors, with each payload carrying both current position (x, y) and movement deltas (dx, dy), enabling clients to recover dropped intermediate states via delta reconstruction.



Optimistic Client Prediction & Reconciliation

Client-side prediction applies input events immediately for zero-latency feedback, while ACQUIRE_REQUESTs are verified using a custom ARQ timer (RTO = RTT + 4×Dev); ACKs confirm and clear predictions, and NACKs trigger an immediate rollback to the server-authoritative state.

Selective Reliability Design

Why Selective Reliability

- We are using UDP for low latency, but UDP does not guarantee delivery.
- Not all game messages need the same level of reliability.
- Player movement & snapshots → can tolerate loss.
- Cell acquire actions → MUST be reliable

Which Messages Are Reliable

- ACQUIRE_REQUEST (Client → Server)
- ACK / NACK (Server → Client)

These messages directly affect:

- Cell ownership
- Player score
- Game outcome

Snapshots are not reliable on purpose to avoid lag

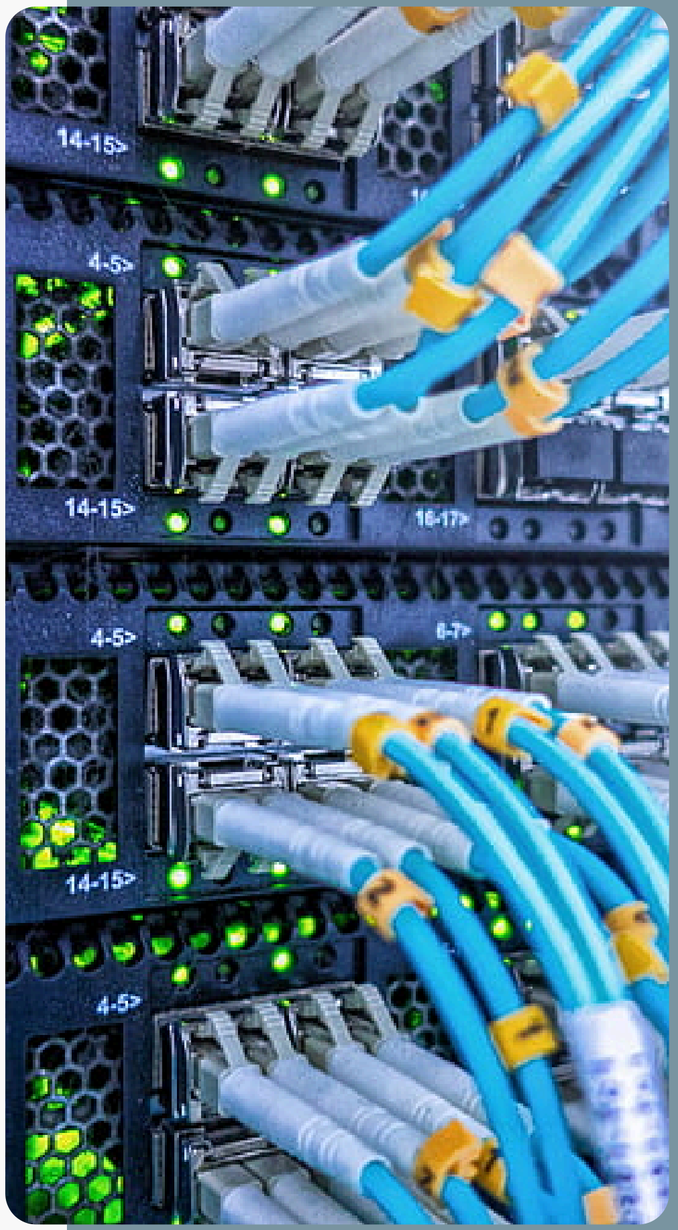
Core Idea

- Each ACQUIRE_REQUEST has a sequence number

Server:

- Detects duplicates
- Sends ACK if success
- Sends NACK if failed

Client can safely retry without breaking game logic



How Is it Implemented

Duplicate Suppression

- Server stores already handled sequence numbers per client: **processed_seqs = set()**
- Server does NOT re-apply it, Just re-sends the previous ACK

Prevents:

- Double scoring
- Double claiming cell

Reliable ACK / NACK System

- ACK = success

Cell was unclaimed OR already owned by the same player

- NACK = failure

Cell already owned by another player

- This allows the client to:
- Confirm success
- Retry safely if packet was lost

Testing Scenarios & Methodology

Goal: Evaluate server-client synchronization under network impairments

Scenario	Network Condition
Baseline	No Impairment
LAN-like loss	2% packet loss
WAN-like loss	5% packet loss
Delay 100ms	100 ms added latency

Metrics Collected

Latency (ms):

- time for snapshot to reach client

Jitter (ms):

- variability in snapshot delivery

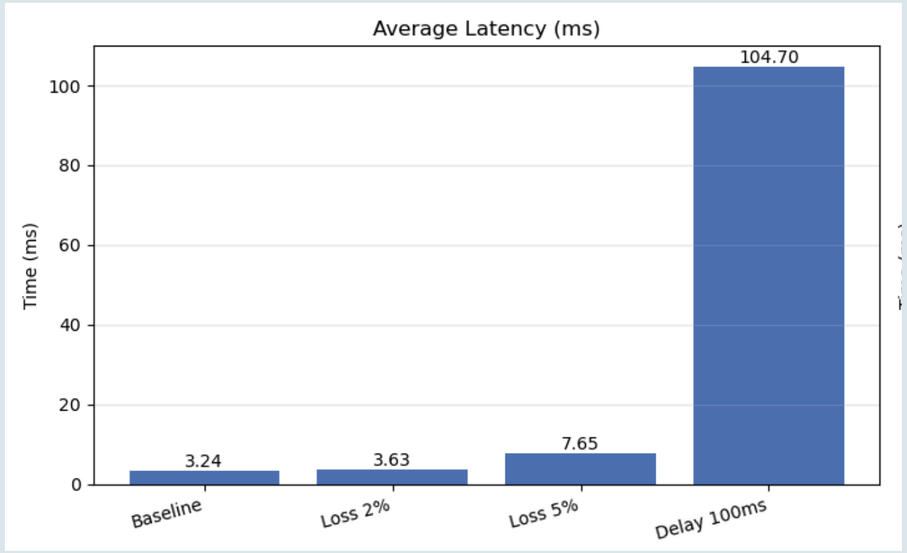
CPU (%)

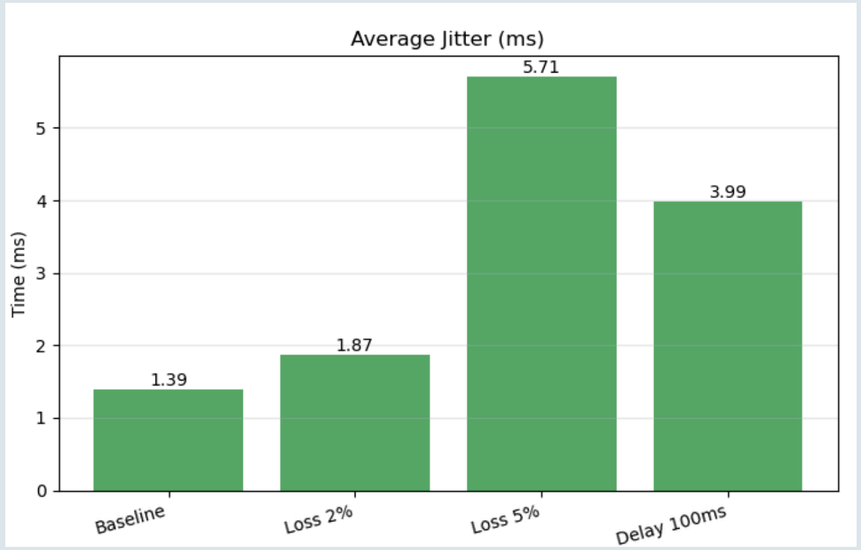
- server load

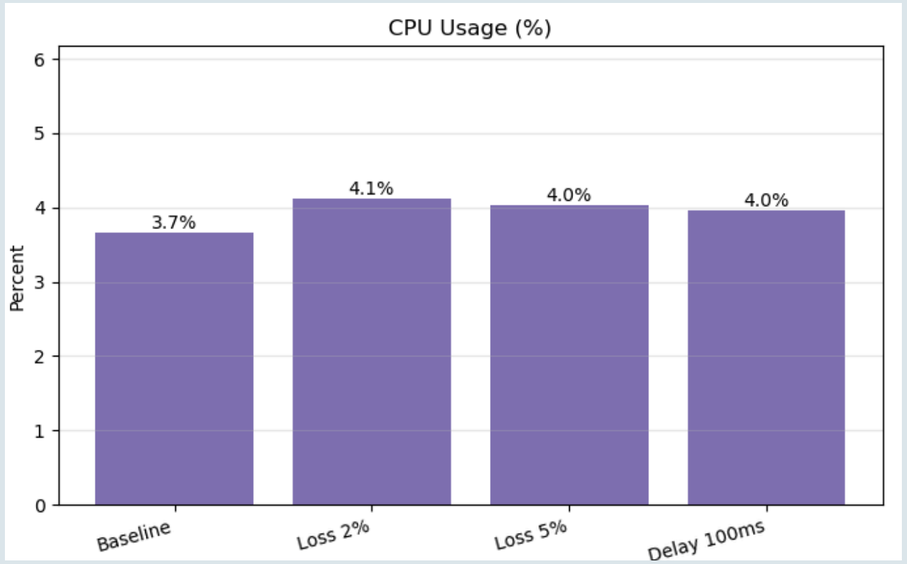
Perceived position error

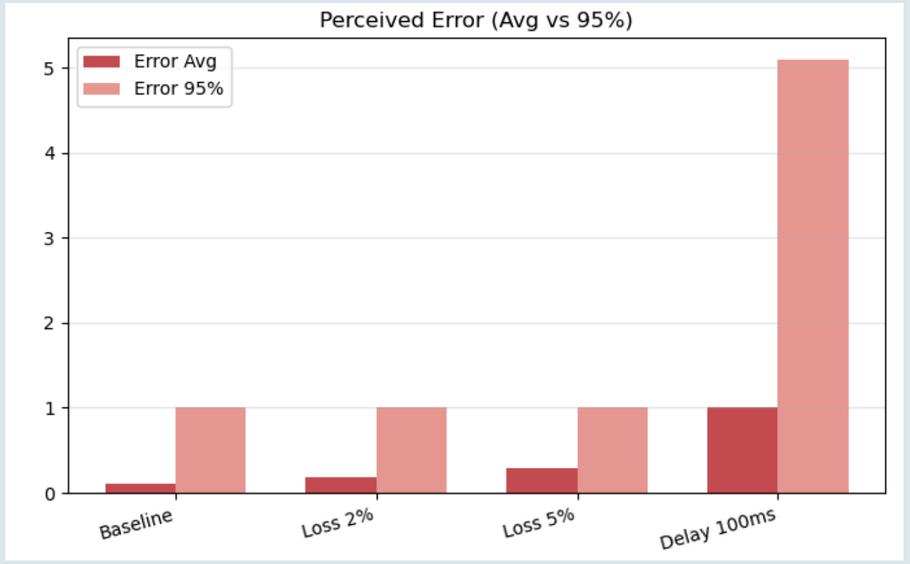
- Euclidean distance between server and client positions

Average results

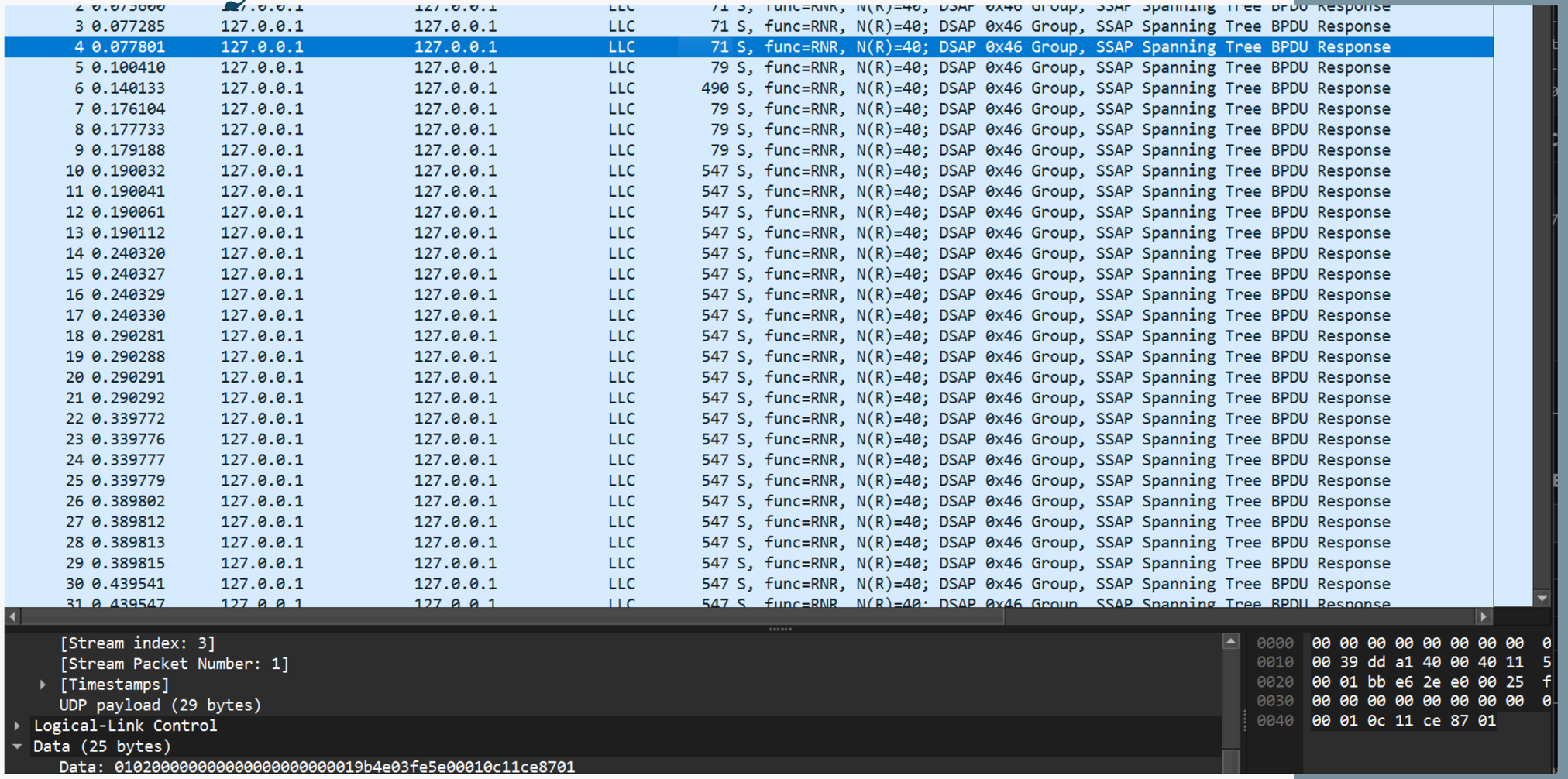








PCAP analysis



3	0.077285	127.0.0.1	127.0.0.1	LLC	71 S, func=RNR, N(R)=40; DSAP 0x46 Group, SSAP Spanning Tree BPDU Response
4	0.077801	127.0.0.1	127.0.0.1	LLC	71 S, func=RNR, N(R)=40; DSAP 0x46 Group, SSAP Spanning Tree BPDU Response
5	0.100410	127.0.0.1	127.0.0.1	LLC	79 S, func=RNR, N(R)=40; DSAP 0x46 Group, SSAP Spanning Tree BPDU Response
6	0.140133	127.0.0.1	127.0.0.1	LLC	490 S, func=RNR, N(R)=40; DSAP 0x46 Group, SSAP Spanning Tree BPDU Response
7	0.176104	127.0.0.1	127.0.0.1	LLC	79 S, func=RNR, N(R)=40; DSAP 0x46 Group, SSAP Spanning Tree BPDU Response
8	0.177733	127.0.0.1	127.0.0.1	LLC	79 S, func=RNR, N(R)=40; DSAP 0x46 Group, SSAP Spanning Tree BPDU Response
9	0.179188	127.0.0.1	127.0.0.1	LLC	79 S, func=RNR, N(R)=40; DSAP 0x46 Group, SSAP Spanning Tree BPDU Response
10	0.190032	127.0.0.1	127.0.0.1	LLC	547 S, func=RNR, N(R)=40; DSAP 0x46 Group, SSAP Spanning Tree BPDU Response
11	0.190041	127.0.0.1	127.0.0.1	LLC	547 S, func=RNR, N(R)=40; DSAP 0x46 Group, SSAP Spanning Tree BPDU Response
12	0.190061	127.0.0.1	127.0.0.1	LLC	547 S, func=RNR, N(R)=40; DSAP 0x46 Group, SSAP Spanning Tree BPDU Response
13	0.190112	127.0.0.1	127.0.0.1	LLC	547 S, func=RNR, N(R)=40; DSAP 0x46 Group, SSAP Spanning Tree BPDU Response
14	0.240320	127.0.0.1	127.0.0.1	LLC	547 S, func=RNR, N(R)=40; DSAP 0x46 Group, SSAP Spanning Tree BPDU Response
15	0.240327	127.0.0.1	127.0.0.1	LLC	547 S, func=RNR, N(R)=40; DSAP 0x46 Group, SSAP Spanning Tree BPDU Response
16	0.240329	127.0.0.1	127.0.0.1	LLC	547 S, func=RNR, N(R)=40; DSAP 0x46 Group, SSAP Spanning Tree BPDU Response
17	0.240330	127.0.0.1	127.0.0.1	LLC	547 S, func=RNR, N(R)=40; DSAP 0x46 Group, SSAP Spanning Tree BPDU Response
18	0.290281	127.0.0.1	127.0.0.1	LLC	547 S, func=RNR, N(R)=40; DSAP 0x46 Group, SSAP Spanning Tree BPDU Response
19	0.290288	127.0.0.1	127.0.0.1	LLC	547 S, func=RNR, N(R)=40; DSAP 0x46 Group, SSAP Spanning Tree BPDU Response
20	0.290291	127.0.0.1	127.0.0.1	LLC	547 S, func=RNR, N(R)=40; DSAP 0x46 Group, SSAP Spanning Tree BPDU Response
21	0.290292	127.0.0.1	127.0.0.1	LLC	547 S, func=RNR, N(R)=40; DSAP 0x46 Group, SSAP Spanning Tree BPDU Response
22	0.339772	127.0.0.1	127.0.0.1	LLC	547 S, func=RNR, N(R)=40; DSAP 0x46 Group, SSAP Spanning Tree BPDU Response
23	0.339776	127.0.0.1	127.0.0.1	LLC	547 S, func=RNR, N(R)=40; DSAP 0x46 Group, SSAP Spanning Tree BPDU Response
24	0.339777	127.0.0.1	127.0.0.1	LLC	547 S, func=RNR, N(R)=40; DSAP 0x46 Group, SSAP Spanning Tree BPDU Response
25	0.339779	127.0.0.1	127.0.0.1	LLC	547 S, func=RNR, N(R)=40; DSAP 0x46 Group, SSAP Spanning Tree BPDU Response
26	0.389802	127.0.0.1	127.0.0.1	LLC	547 S, func=RNR, N(R)=40; DSAP 0x46 Group, SSAP Spanning Tree BPDU Response
27	0.389812	127.0.0.1	127.0.0.1	LLC	547 S, func=RNR, N(R)=40; DSAP 0x46 Group, SSAP Spanning Tree BPDU Response
28	0.389813	127.0.0.1	127.0.0.1	LLC	547 S, func=RNR, N(R)=40; DSAP 0x46 Group, SSAP Spanning Tree BPDU Response
29	0.389815	127.0.0.1	127.0.0.1	LLC	547 S, func=RNR, N(R)=40; DSAP 0x46 Group, SSAP Spanning Tree BPDU Response
30	0.439541	127.0.0.1	127.0.0.1	LLC	547 S, func=RNR, N(R)=40; DSAP 0x46 Group, SSAP Spanning Tree BPDU Response

```
    [Stream index: 3]
    [Stream Packet Number: 1]
  ▸ [Timestamps]
    UDP payload (29 bytes)
▸ Logical-Link Control
▾ Data (25 bytes)
    Data: 0102000000000000000000000019b4e03fe5e00010c11ce8701
```

```
0000  00 00 00 00 00 00 00 00  0
0010  00 39 dd a1 40 00 40 11  5
0020  00 01 bb e6 2e e0 00 25  f
0030  00 00 00 00 00 00 00 00  0
0040  00 01 0c 11 ce 87 01
```

Packet structure: |GCUP| |v | |type| |snap_id| |seq_num| |timestamp| ||en| |crc32|

PCAP analysis

Key Observations

1. Handshake CLIENT_INIT → Client sends player ID to register

SERVER_HELLO ← Server confirms, sends grid size

2. Game Loop

SNAPSHOT ← Server broadcasts full state at 20Hz to all clients

3. Player Actions

ACQUIRE_REQ → Client requests to claim cell (row, col)

ACK ← Server confirms receipt

Next SNAPSHOT reflects the change

4. Keep-Alive

HEARTBEAT → Client sends periodically to stay connected

5. Integrity verification: CRC32 checksum on every packet

Limitations & Future Work

Scalability & Bandwidth Efficiency:

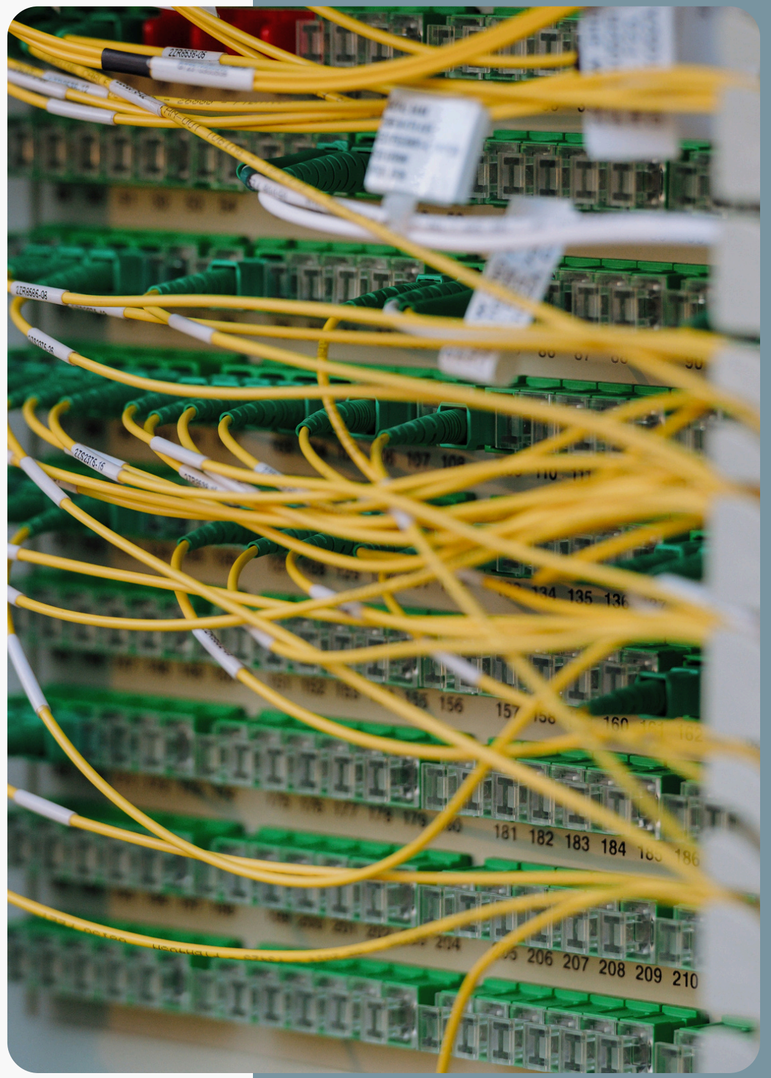
- **Issue:** Full-state broadcasts create linear bandwidth growth O(GridSize).
- **Constraint:** Large grids exceed the 1200-byte MTU limit.
- **Future Work:** Implement Delta Compression (send changes only) and Area of Interest (AoI) filtering.

Reliability & Congestion:

- **Issue:** Static 20Hz update rate does not adapt to network congestion.
- **Constraint:** No application-layer fragmentation for large states.
- **Future Work:** Adaptive Congestion Control (dynamic tick rate) and multi-packet reconstruction logic.

Security:

- **Issue:** Cleartext UDP allows spoofing and replay attacks.
- **Issue:** Minimal server-side sanity checks allow invalid moves.
- **Future Work:** DTLS encapsulation, server-side logical sanity checks.



Conclusion

Implemented GCUP with a custom header structure to handle sequencing, packet drops, and jitter.
System is fully operational, meeting all functional requirements with documented limits on bandwidth scaling.

Thank you