Game Architecture

Networking

Today’s Agenda

- The Internet
- Network Topology Choices
- Writing a Game Network Protocol
- Synchronizing Gameplay

The Internet

- Bandwidth
- Latency
- Speed of Light
- Observed Latency
- Implications

The Internet

- **Bandwidth**
- Latency
- Speed of Light
- Observed Latency
- Implications

The maximum **rate** at which we can send/receive data.

Measured in *amount of data* per *time*

Usually

- megabits per second (Mb/s)
- kilobits per second (kb/s)

Sometimes

- megabytes per second (MB/s)
- kilobytes per second (kB/s)

The Internet

- Bandwidth
- **Latency**
- Speed of Light
- Observed Latency
- Implications

“Latency is the term used to indicate any kind of **delay** that happens in data communication over a network.

Latency is physically a consequence of the limited velocity with which any physical interaction can propagate. This velocity is always lower than or equal to the **speed of light**.”

Latency is a *time interval*

We measure latency in **milliseconds**.

The Internet

- Bandwidth
- Latency
- **Speed of Light**
- Observed Latency
- Implications

$C \approx 300{,}000{,}000$ m/s

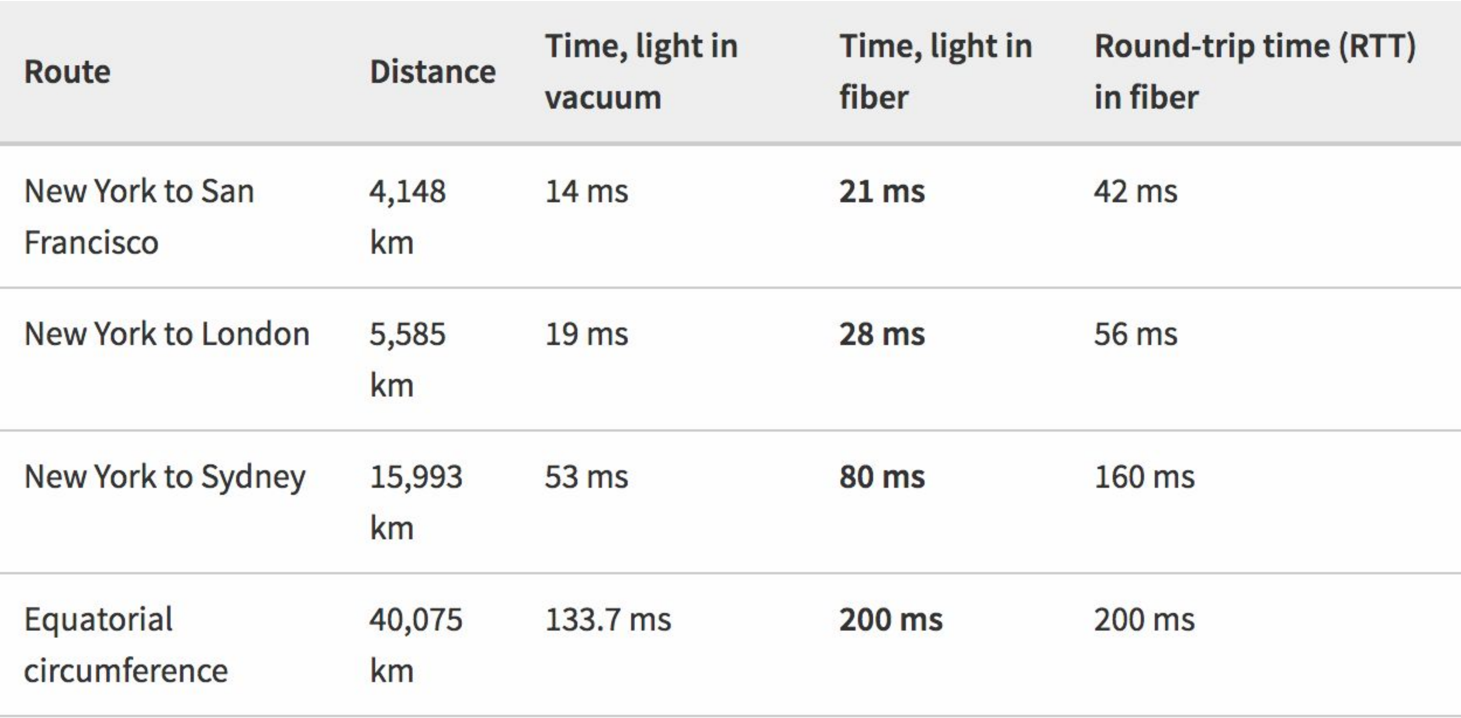
The Internet

- Bandwidth
- Latency
- **Speed of Light**
- Observed Latency
- Implications

$C \approx 300{,}000{,}000$ m/s

$C_f \approx 200{,}000{,}000$ m/s

The Internet - Observed Latency



Route	Distance	Time, light in vacuum	Time, light in fiber	Round-trip time (RTT) in fiber
New York to San Francisco	4,148 km	14 ms	**21 ms**	42 ms
New York to London	5,585 km	19 ms	**28 ms**	56 ms
New York to Sydney	15,993 km	53 ms	**80 ms**	160 ms
Equatorial circumference	40,075 km	133.7 ms	**200 ms**	200 ms

https://hpbn.co/primer-on-latency-and-bandwidth/#speed-of-light-and-propagation-latency

The Internet

- Bandwidth
- Latency
- Speed of Light
- **Observed Latency**
- Implications

Distance	RTT
Wired LAN	< 1 ms
Same City	~10-20 ms
Inter-City	20-50 ms
NY to SF	80 ms
NY to London	80 ms
LA to Tokyo	120 ms
NY to Sydney	200 ms

http://ithare.com/mmog-rtt-input-lag-and-how-to-mitigate-them/

The Internet

- Bandwidth
- Latency
- Speed of Light
- Observed Latency
- **Implications**

NY to SF = 80 ms

NY to SF = 5 frames @ 60 fps

Render lag = 4 frames (say)

Total lag from socket send, to seeing a rendered result on screen:

= RTT + Render Lag

= 80 ms + 67 ms

= **147 ms**

The Internet

- Bandwidth
- Latency
- Speed of Light
- Observed Latency
- **Implications**

Always ask:

Where am I hiding the lag?

http://www.gdcvault.com/play/1014345/I-Shot-You-First-Networking

I Shot You First: Networking the Gameplay of HALO: REACH
David Aldridge, Bungie

The Internet

- Bandwidth
- Latency
- Speed of Light
- Observed Latency
- **Implications**

You need to simulate lag, because there is none on your LAN.

Wrap your sockets in a debug socket and add:

Latency	Fixed wait time before send
Jitter	+/- Latency variance
Packet loss	% Chance to drop a packet
Packet Duplication	% Chance to send a duplicated packet
Packet Order	Send packets out-of-order

Today’s Agenda

- The Internet
- **Network Topology Choices**
- Writing a Game Network Protocol
- Synchronizing Gameplay

Network Topology Choices

- Definition
- Input Based Peer-to-Peer
- Strict Client-Server
- State Based Peer-to-Peer

Network Topology Choices

- **Definition**
- Input Based Peer-to-Peer
- Strict Client-Server
- State Based Peer-to-Peer

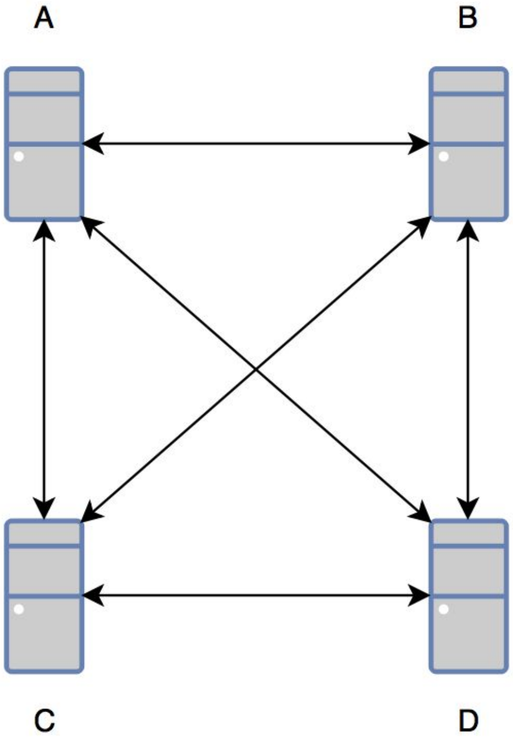
A network topology specifies:

- The pattern in which nodes connect to each other
- The roles and responsibilities of each node

Network Topology Choices

- **Definition**
- Input Based Peer-to-Peer
- Strict Client-Server
- State Based Peer-to-Peer

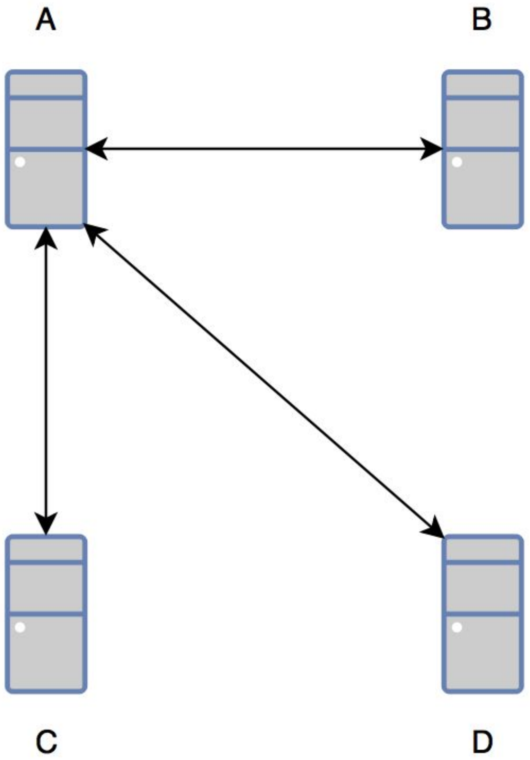
Peer-to-Peer



Network Topology Choices

- **Definition**
- Input Based Peer-to-Peer
- Strict Client-Server
- State Based Peer-to-Peer

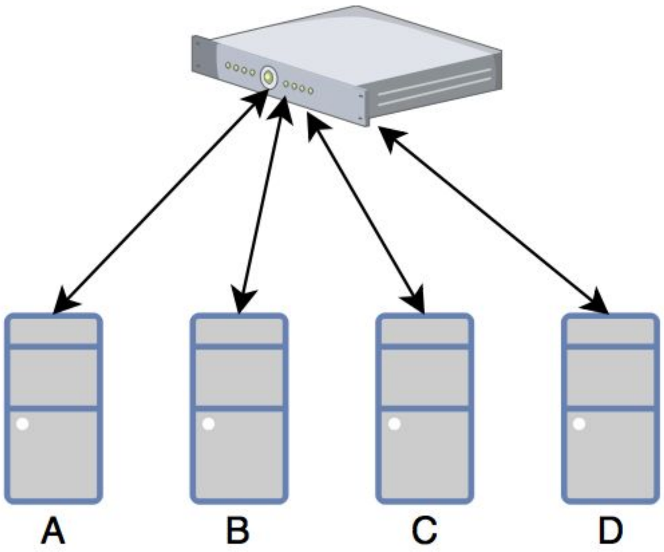
Client-Server (listen server)



Network Topology Choices

- **Definition**
- Input Based Peer-to-Peer
- Strict Client-Server
- State Based Peer-to-Peer

Client-Server (dedicated server)



Network Topology Choices

- Definition
- **Input Based Peer-to-Peer**
- Strict Client-Server
- State Based Peer-to-Peer

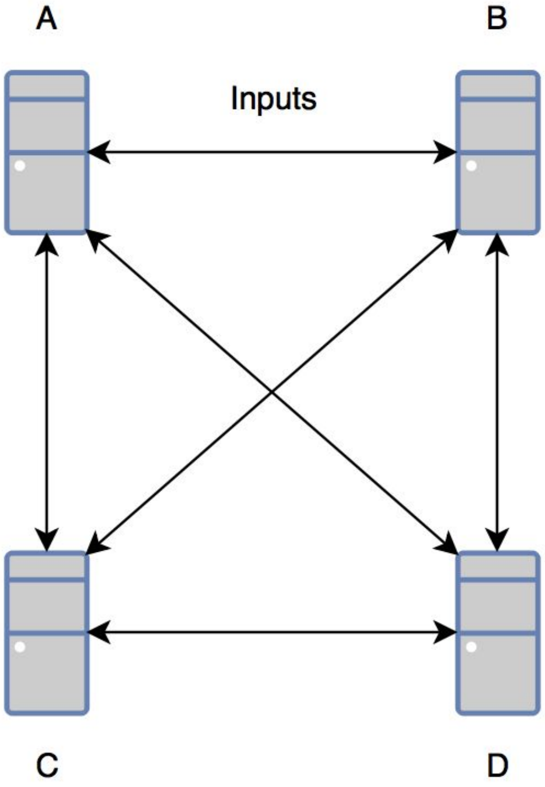
Also known as:

- Synchronous P2P
- Lockstep P2P
- Deterministic Lockstep P2P
- Key Sharing

Network Topology Choices - Lockstep P2P

Operation

1. Everyone sends controller **inputs** to everyone else
2. Everyone waits for everyone
3. Tick frame
4. Repeat



Network Topology Choices - Lockstep P2P

Pros

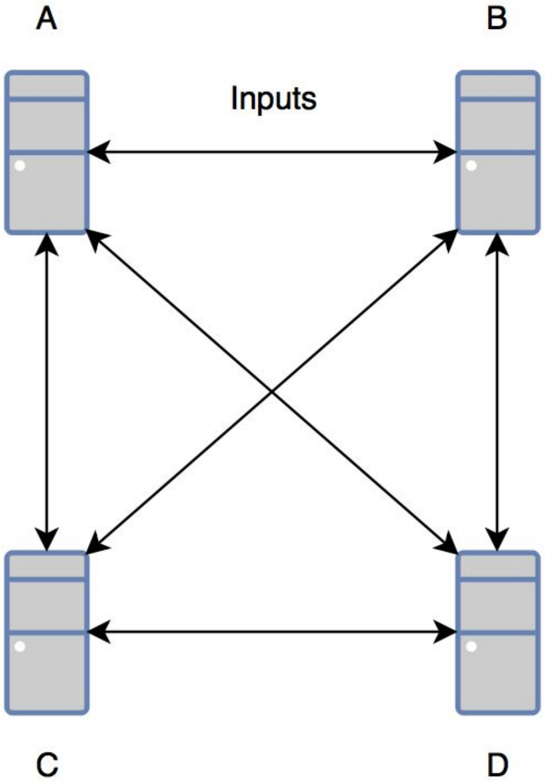
- It “just works”
- No impact on game engine
- Great for turn based games

Examples

- Starcraft
- Halo campaign

Cons

- Requires determinism
- Laggy input client-side
- Latency of system = latency of slowest peer
- Doesn’t scale beyond 2-4 players
- Requires fully connected mesh (or TURN server)
- Join-in-progress is impossible



Network Address Translation (NAT)

Aside:

- No. devices connecting to the internet > No. IPv4 addresses
- This is why IPv6 exists
- The solution: NAT
- Map a single public IP to multiple private IPs/Ports
- **Games: I need your public IP address in order to connect to you**

Example:

Local IP/Port:

192.168.1.129:3074

Mapped to Public IP/Port:

23.254.202.80:3074

Or (more likely)

23.254.202.80:65324

Network Address Translation (NAT) (cont.)

STUN

- Algorithm to determine NAT type
- https://en.wikipedia.org/wiki/STUN

Three Classifications:

- **Open** (static internal/external port mappings)
- **Moderate** (ports mapped on first use)
- **Strict** (random external port mapping)

	O	**M**	**S**
O	✓	✓	✓
M	✓	✓	✕
S	✓	✕	✕

Network Topology Choices - Lockstep P2P

Pros

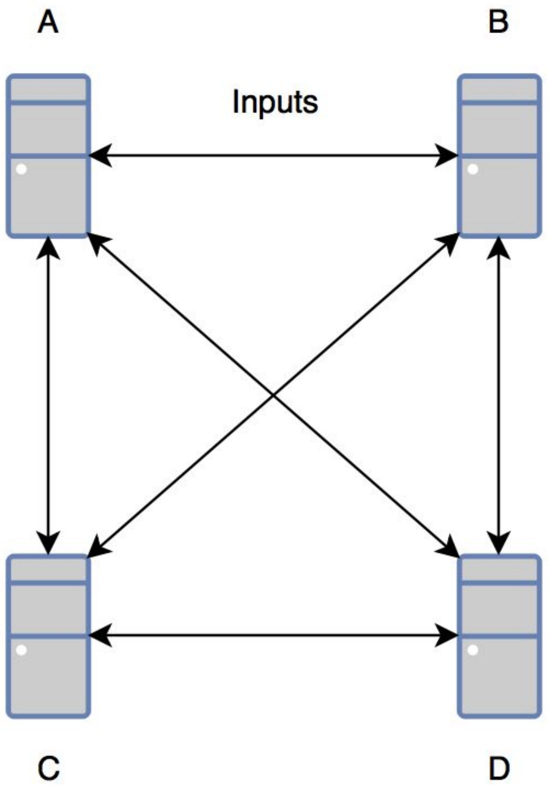
- It “just works”
- No impact on game engine
- Great for turn based games

Examples

- Starcraft
- Halo campaign

Cons

- Requires determinism
- Laggy input client-side
- Latency of system = latency of slowest peer
- Doesn’t scale beyond 2-4 players
- Requires fully connected mesh (or TURN server)
- Join-in-progress is impossible
- **Near impossible for clan matches**



Network Topology Choices

- Definition
- Input Based Peer-to-Peer
- **Strict Client-Server**
- State Based Peer-to-Peer

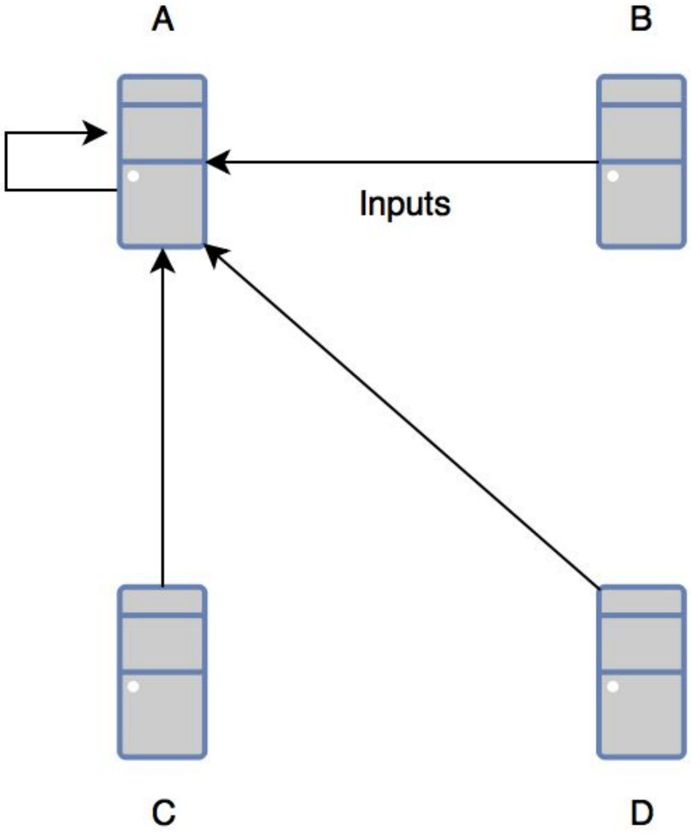
Also known as:

- Client-Server

Network Topology Choices - Strict Client-Server

Operation

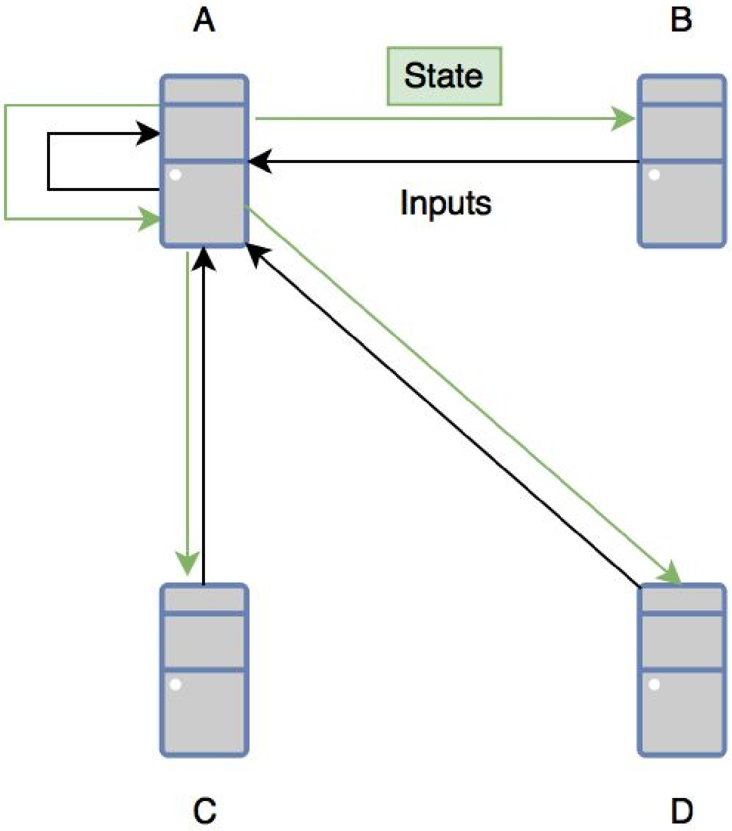
1. Clients send controller **inputs** to the Server at 60 Hz
2. Server runs the full game simulation



Network Topology Choices - Strict Client-Server

Operation

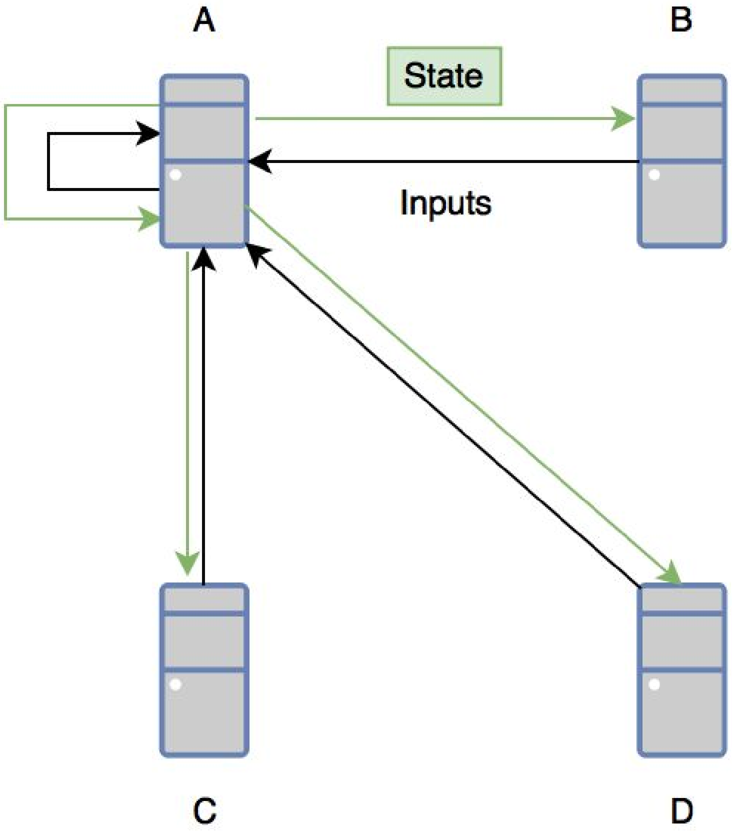
1. Clients send controller **inputs** to the Server at 60 Hz
2. Server runs the full game simulation
3. Server sends world **state** to all clients at ~20 Hz (aka *snapshots*)



Network Topology Choices - Strict Client-Server

Important Notes

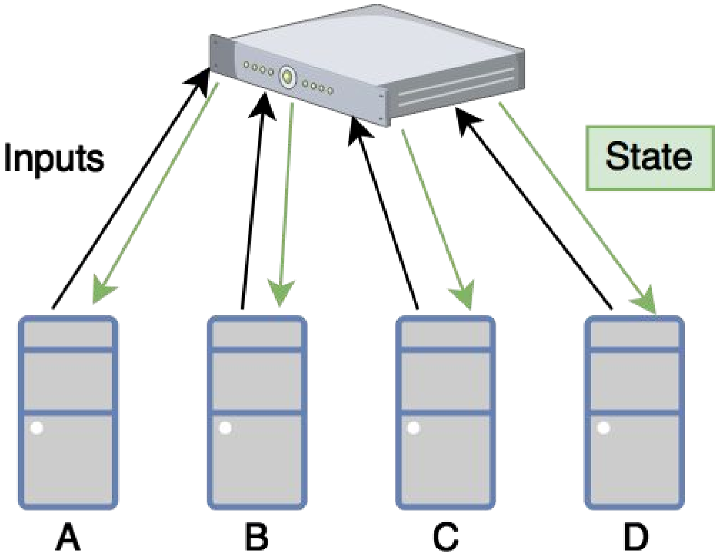
- Server is the *only* node that runs the full game simulation
  - //game/code/server/... vs. //game/code/client/...
- Clients are **not** authoritative over their state
- Corollary: Clients must **predict** their own movement, otherwise the game would feel laggy because the inputs wouldn't affect local state until the next snapshot
- We call this **Client Side Prediction**
- Clients must accept corrections from Server (more on this later)



Network Topology Choices - Strict Client-Server

Important Notes

- Server is the *only* node that runs the full game simulation
    - //game/code/server/... vs. //game/code/client/...
- Clients are **not** authoritative over their state
- Corollary: Clients must **predict** their own movement, otherwise the game would feel laggy because the inputs wouldn't affect local state until the next snapshot
- We call this **Client Side Prediction**
- Clients must accept corrections from Server (more on this later)



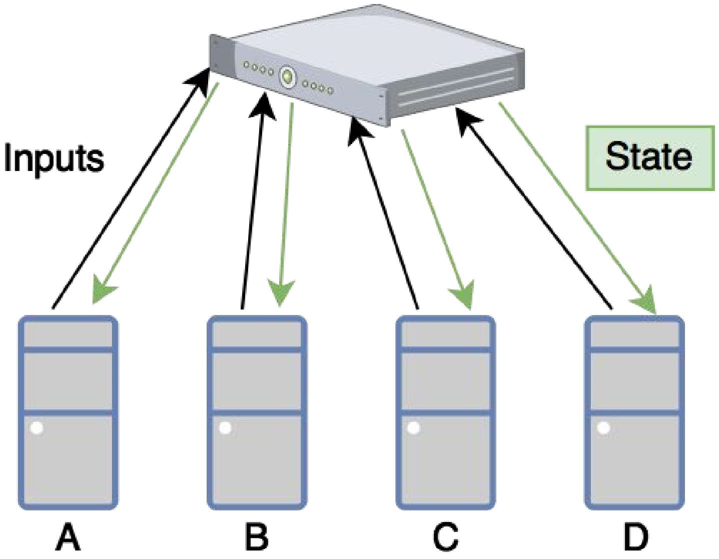
Network Topology Choices - Strict Client-Server

Pros

- Security: Single authority, can be hosted in cloud
- NAT problems disappear (Listen Servers are chosen with open NAT)
- High player counts
- Dedicated servers mean no need for host migration
- Join-in-progress mostly works

Cons

- Server needs high upstream bandwidth
- Listen Server uses more CPU
- Dedicated servers need to be very well distributed to give players good pings
- Host migration is difficult with a Listen Server (clients do not have full state)



Examples

- Quake, Call of Duty, Overwatch

Network Topology Choices

- Definition
- Input Based Peer-to-Peer
- Strict Client-Server
- **State Based Peer-to-Peer**

Also known as:

- Peer-to-Peer

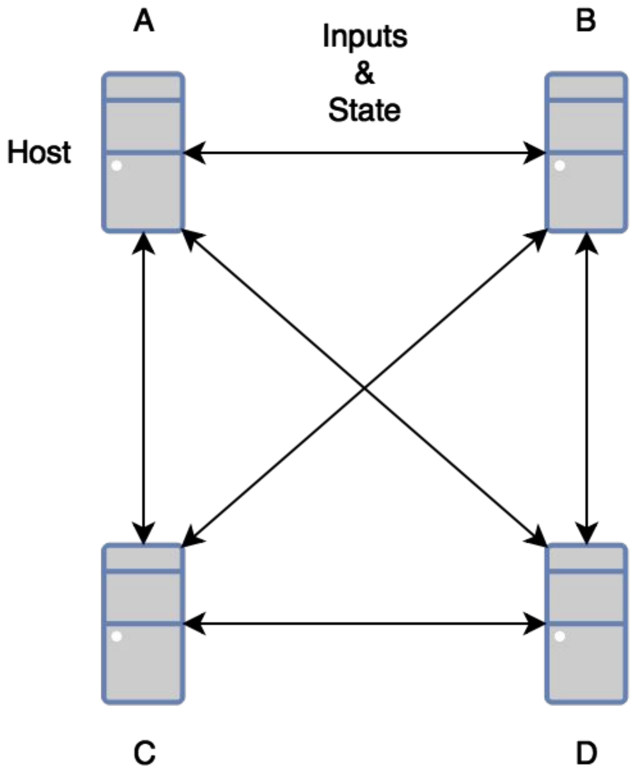
Often modified to:

- Client-Server with distributed authority
- “Generalized Client-Server”

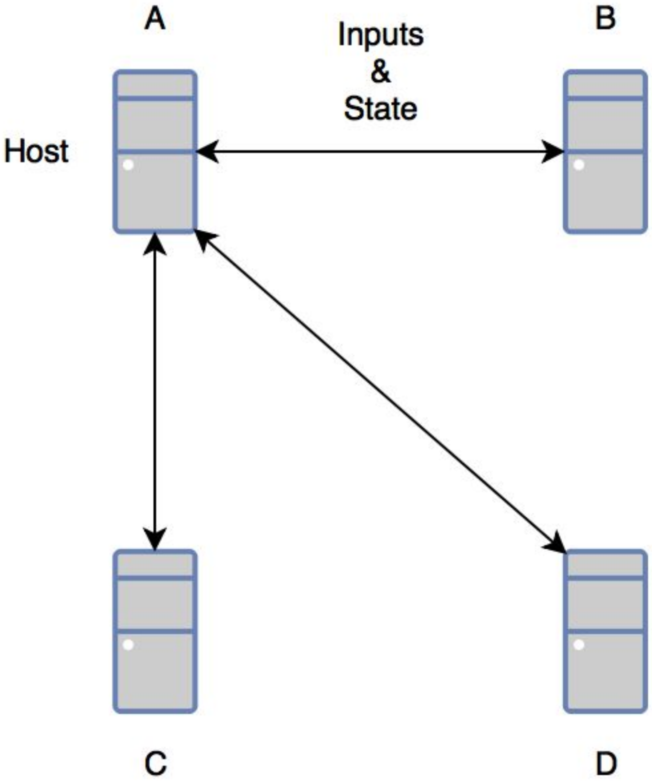
Network Topology Choices - State Based P2P

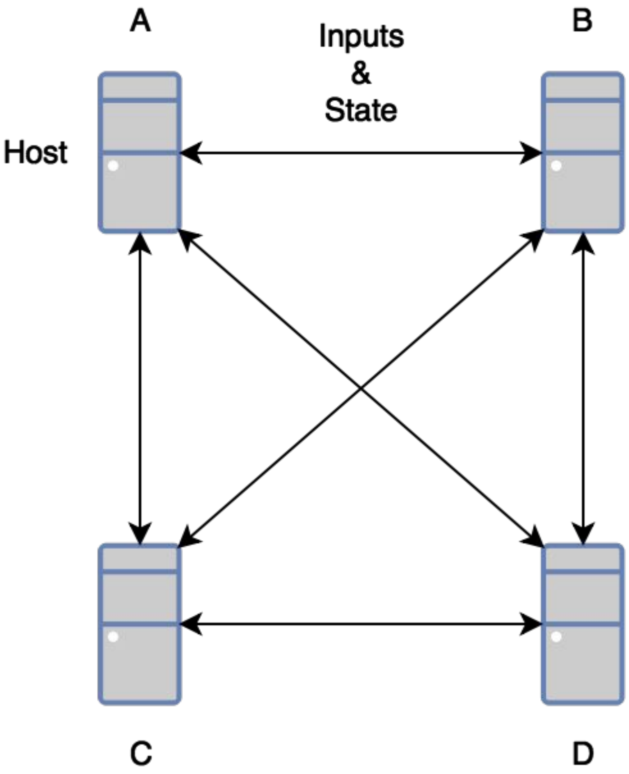
Operation

1. Peers are authoritative over their own player
2. Host is authoritative over everything else by default
3. Everyone applies inputs locally
4. Everyone runs full simulation
5. I send you state about my player
6. You send me state about your player
7. Host sends state about everything else



Network Topology Choices - State Based P2P

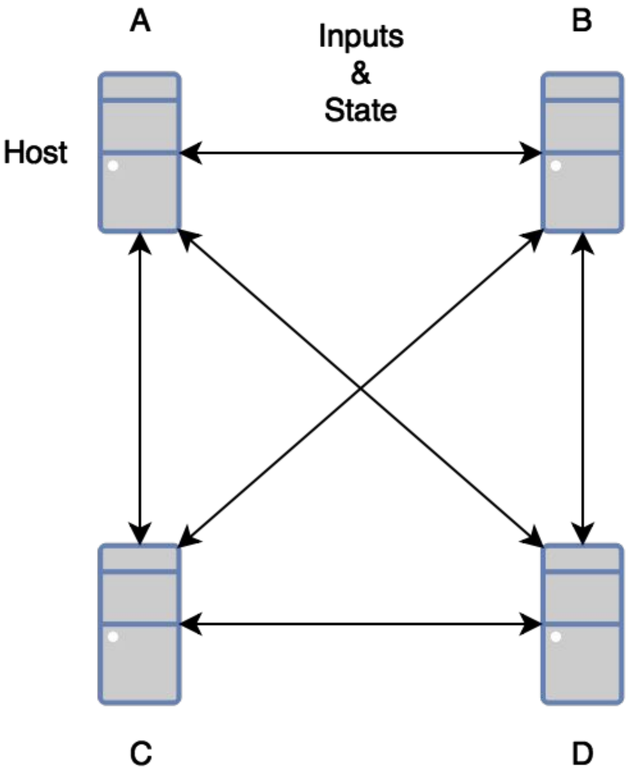




Network Topology Choices - State Based P2P

Important Notes

- This is a common architecture, it's the easiest way to integrate online into an engine without rewriting everything
- Code is sprinkled with `if (HasAuthority())`
- Need for Client-Side Prediction is obviated
- Authority migration becomes a feature
  - This feels laggy - take authority over it!
- Request/Response to Host from Peers for permission to manipulate entities with Host authority
  - Peer: `ServerPickUpCollectible();`



Network Topology Choices - State Based P2P

Pros

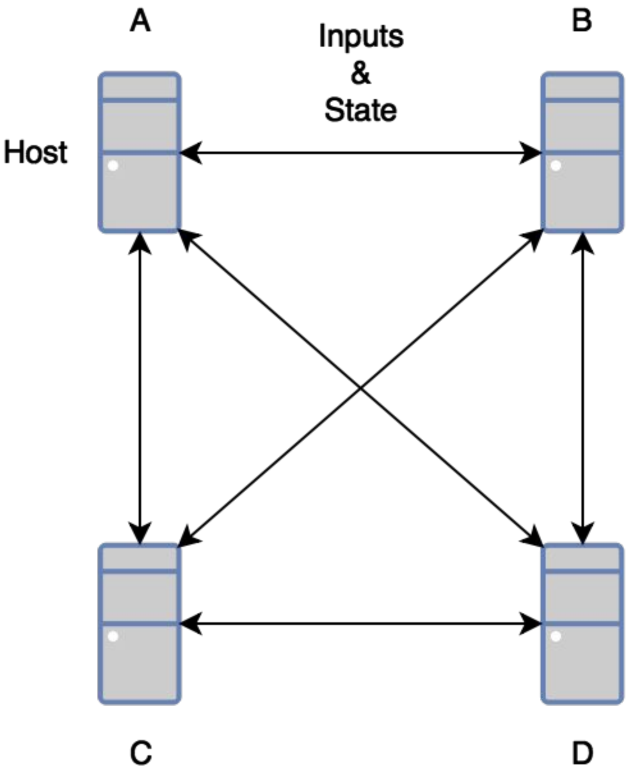
- Can support large worlds with lots of entities assuming relevancy checks (allow clients to simulate sub-sections)
- Get up and running quicker than Strict Client-Server
- Can massage single-player code to work online
- Join-in-progress is doable
- Host migration is doable

Cons

- Not great for PvP - who arbitrates damage, victim or attacker?
- Client authority means hacking (The Division)
- Can massage single-player code to work online (bugs!)

Examples

- Skylanders, Ubisoft, Unreal Engine, Destiny



Network Topology Choices - Summary

Phew… let’s recap

Network Topology Choices - Summary

	Player Count	NAT Compat	Security	Host Migration	Join In Progress	Large World	Ease of Integration	Total
Input Based P2P	0	1	2	3	0	3	3	12
State Based P2P	1.5	1	0	3	2	3	2	12.5
Generalized C-S	3	3	1.5	2	2	3	2	14
Strict C-S	3	3	3	1	3	1	0	14

0 = bad, 3 = good

This exercise is completely arbitrary and used to illustrate a point - there’s no one perfect topology choice

Today’s Agenda

- The Internet
- Network Topology Choices
- **Writing a Game Network Protocol**
- Synchronizing Gameplay

Writing a Game Network Protocol

- TCP
- UDP
- MTU
- Replication
- Object Naming
- Packet Structure
- Quantization
- Prediction, Rollback & Correction
- Case Study: Quake

Writing a Game Network Protocol

- TCP
- UDP
- MTU
- Replication
- Object Naming
- Packet Structure
- Quantization
- Prediction, Rollback & Correction
- Case Study: Quake

You will send and receive data using a **Socket**

Two choices:

- **TCP: Transmission Control Protocol**
- **UDP: User Datagram Protocol**

Writing a Game Network Protocol

- **TCP**
- UDP
- MTU
- Replication
- Object Naming
- Packet Structure
- Quantization
- Prediction, Rollback & Correction
- Case Study: Quake

Properties:

- Connection based
- Reliable
- Ordered
- Stream; TCP splits data into packets for you
- Flow control; won’t send data faster than your connection can handle

Basically how the entire internet operates

Writing a Game Network Protocol

- **TCP**
- UDP
- MTU
- Replication
- Object Naming
- Packet Structure
- Quantization
- Prediction, Rollback & Correction
- Case Study: Quake

Caveats:

- The internet is ***unreliable***
- Routers drop packets
- Sometimes messages get delivered twice, or out of order

This is a fact of life, it’s the internet *by definition*

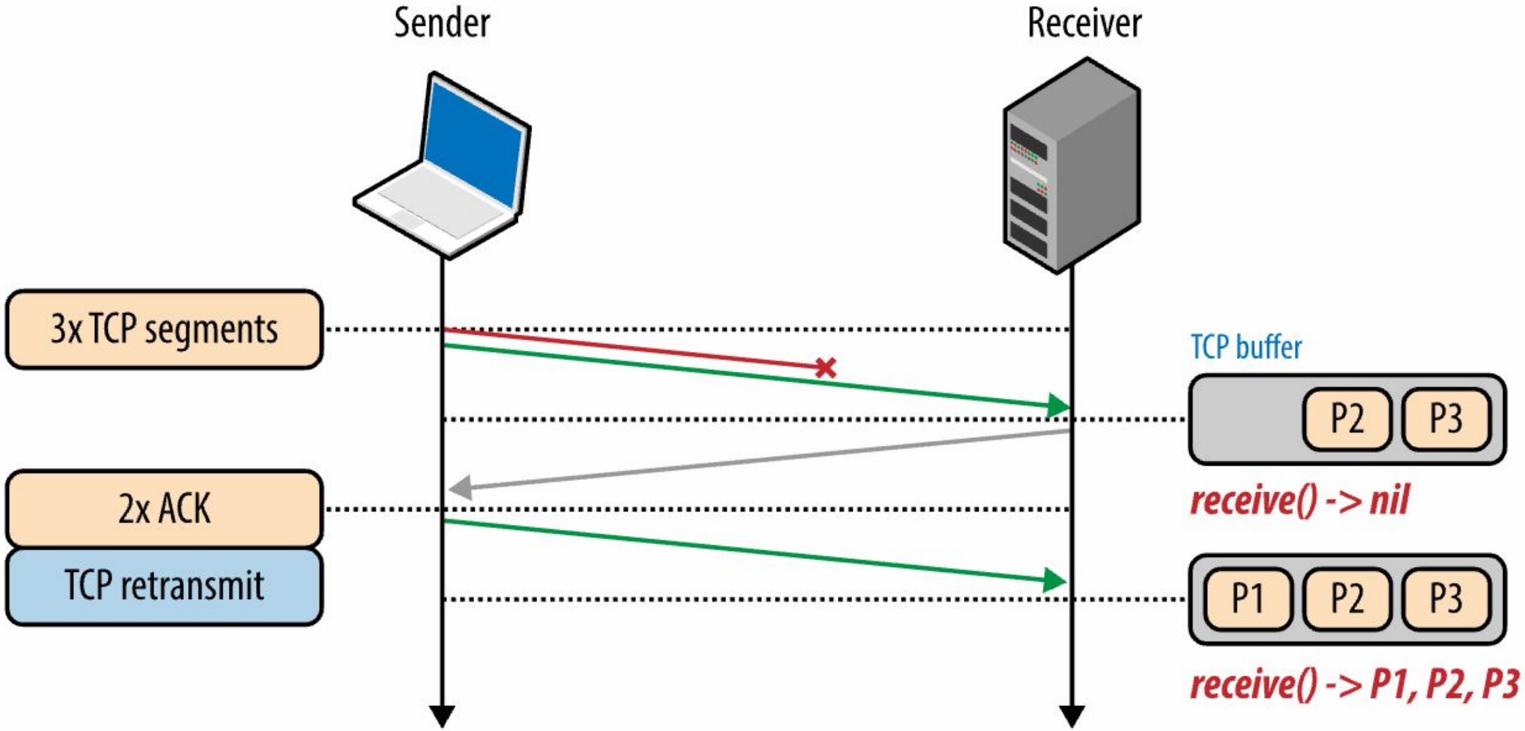
Writing a Game Network Protocol

- **TCP**
- UDP
- MTU
- Replication
- Object Naming
- Packet Structure
- Quantization
- Prediction, Rollback & Correction
- Case Study: Quake

Corollary:

- TCP does a lot of legwork to provide a reliable, ordered stream of data
- This extra work is bad news for real-time applications - *it results in increased latency*
- This problem is called **head of line blocking**

Writing a Game Network Protocol



https://hpbn.co/building-blocks-of-tcp/#head-of-line-blocking

Writing a Game Network Protocol

- **TCP**
- UDP
- MTU
- Replication
- Object Naming
- Packet Structure
- Quantization
- Prediction, Rollback & Correction
- Case Study: Quake

Operation:

1. Send a packet
2. Wait for an acknowledgment
3. If timeout hit, resend

Writing a Game Network Protocol

- **TCP**
- UDP
- MTU
- Replication
- Object Naming
- Packet Structure
- Quantization
- Prediction, Rollback & Correction
- Case Study: Quake

Summary:

- TCP is a hardcore way to send every packet reliably and in order

> “Reality is not a hack you're forced to deal with to solve your abstract, theoretical problem. Reality is the actual problem.” -Mike Acton

Writing a Game Network Protocol

- TCP
- **UDP**
- MTU
- Replication
- Object Naming
- Packet Structure
- Quantization
- Prediction, Rollback & Correction
- Case Study: Quake

Writing a Game Network Protocol

- TCP
- **UDP**
- MTU
- Replication
- Object Naming
- Packet Structure
- Quantization
- Prediction, Rollback & Correction
- Case Study: Quake

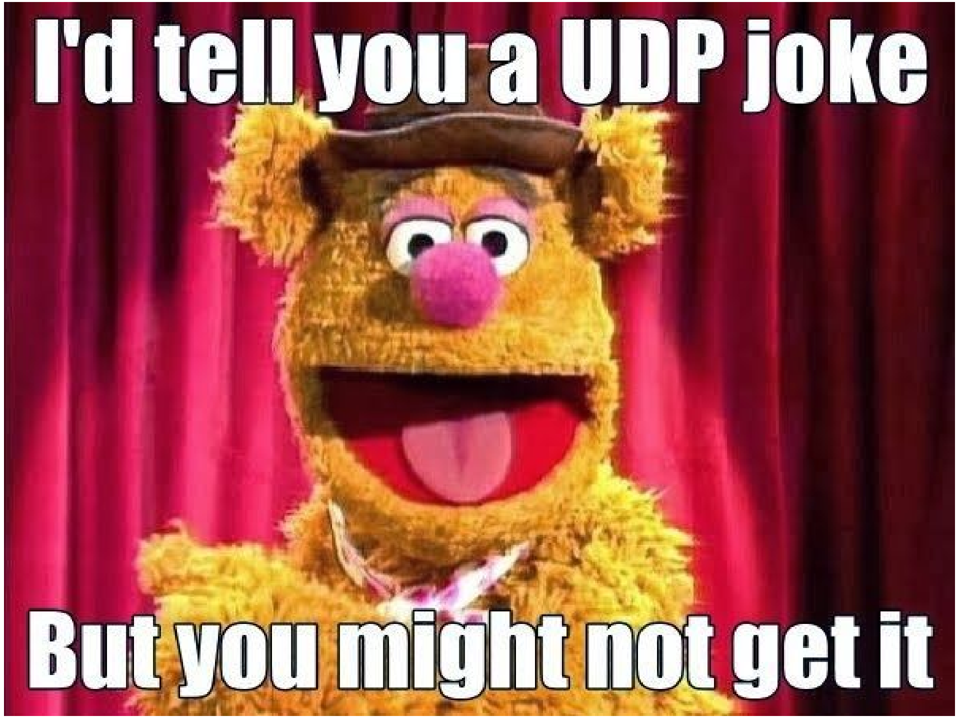
Properties:

- Connectionless
- No guarantees on reliability
- No guarantees on ordering
- No flow control; be careful not to send too much/too often
- No stream; data must be broken up into packets manually*

Usually how real-time applications operate

Writing a Game Network Protocol

- TCP
- **UDP**
- MTU
- Replication
- Object Naming
- Packet Structure
- Quantization
- Prediction, Rollback & Correction
- Case Study: Quake



Writing a Game Network Protocol

- TCP
- **UDP**
- MTU
- Replication
- Object Naming
- Packet Structure
- Quantization
- Prediction, Rollback & Correction
- Case Study: Quake

Why UDP for games?

- If you receive the state of my character at time **T** you want to consume that as soon as possible
- You don't care about my state at time **T-1**, because it's already out of date
- Our game engine will fill in the blanks, via interpolation

Writing a Game Network Protocol

- TCP
- **UDP**
- MTU
- Replication
- Object Naming
- Packet Structure
- Quantization
- Prediction, Rollback & Correction
- Case Study: Quake

What if i really need you to get something?

- Example: Host tells clients to change map
- This is important
- The host needs to resend this until all clients have acknowledged it
- This is a rare case (compared to all of the character state that gets sent every frame) but it’s important, so we must handle it (more on this later...)

Writing a Game Network Protocol

- TCP
- UDP
- **MTU**
- Replication
- Object Naming
- Packet Structure
- Quantization
- Prediction, Rollback & Correction
- Case Study: Quake

Maximum Transmission Unit

- The **maximum sized datagram** that can be transmitted through the next network is called the maximum transmission unit
- MTU is typically 1500 bytes

IP Header + UDP Header + Payload <= MTU

(20 to 60 bytes) + (8 bytes) + Payload <= MTU

Payload <= 1432 bytes

Writing a Game Network Protocol

- TCP
- UDP
- MTU
- **Replication**
- Object Naming
- Packet Structure
- Quantization
- Prediction, Rollback & Correction
- Case Study: Quake

Replication

- Synchronizing data from one game to another (from authority to non-authority)

State Object

- An instance of a class (potentially containing sub-objects) with a lifetime of more than a single frame (e.g. a player)

Event Object

- A transient object that represents the occurrence of an action, similar to a function call (e.g. an explosion)

Writing a Game Network Protocol

- TCP
- UDP
- MTU
- **Replication**
- Object Naming
- Packet Structure
- Quantization
- Prediction, Rollback & Correction
- Case Study: Quake

State Object

1. Authority serializes all replicated fields for state object **S** into a packet **P**
2. Authority adds **Sequence Number++** to packet **P**
3. Packet **P** is sent to Bob
   - ○ Packet may or may not arrive
4. If packet arrives, Bob replies with an acknowledgement - “I received Sequence Number N”
   - ○ Note that the acknowledgement may not arrive either
5. Repeat

Writing a Game Network Protocol

- TCP
- UDP
- MTU
- **Replication**
- Object Naming
- Packet Structure
- Quantization
- Prediction, Rollback & Correction
- Case Study: Quake

Why the Sequence Number?

- Authority can use the sequence number to delta compress replicated fields
- If i send you my location (100, 100, 100) and you acknowledge that, then when my location changes to (150, 150, 100) i can send you a diff (50, 50, 0)
- This is called **Delta Compression**

Writing a Game Network Protocol

- TCP
- UDP
- MTU
- **Replication**
- Object Naming
- Packet Structure
- Quantization
- Prediction, Rollback & Correction
- Case Study: Quake

Event Object

- Similar to a state object, but a bit simpler
- An event is encapsulated into an object
- The event’s parameters are fields
- Authority serializes the fields into a packet and sends it
- When event is deserialized by recipient, an `event->OnReceive()` method is invoked and the effects of the event are executed

Writing a Game Network Protocol

- TCP
- UDP
- MTU
- Replication
- **Object Naming**
- Packet Structure
- Quantization
- Prediction, Rollback & Correction
- Case Study: Quake

Network Object Id

- When the authority sends a State Object, it attaches an Object Id
- On receipt of a packet, the Object Id is read and an Object Map is queried to see if the object exists

```
map<ga_object_id, ga_object*>
```

- If the object does not exist, create one and add it to the map, else, retrieve it
- Object Id is usually a combination of globally unique Machine/Player Id and monotonically increasing integer

Writing a Game Network Protocol

- TCP
- UDP
- MTU
- Replication
- **Object Naming**
- Packet Structure
- Quantization
- Prediction, Rollback & Correction
- Case Study: Quake

Network Object Id

- Authority may also send flags indicating that the object is **new** or that the object has just **died**, so the recipient can perform additional actions
- A new state object will also include a **Type Id** that allows the recipient to construct an object of that type from a factory

Writing a Game Network Protocol

- TCP
- UDP
- MTU
- Replication
- Object Naming
- **Packet Structure**
- Quantization
- Prediction, Rollback & Correction
- Case Study: Quake

Filling a Packet

- Remember we can only send 1432 bytes
- How many entities do we have?

Num Entities x Entity Size <= 1432 bytes

100 x 14 bytes = 1400 bytes

What if our entities are bigger?

Writing a Game Network Protocol

- TCP
- UDP
- MTU
- Replication
- Object Naming
- **Packet Structure**
- Quantization
- Prediction, Rollback & Correction
- Case Study: Quake

Priority

- Give each entity a priority [0.0f, 1.0f]
- Serialize entities in priority order until packet is full
- Bump the priority of entities that were not sent, they will be sent next frame
- Heuristic for priority calculation: Distance to the player’s camera
- Some entities may not have a position in the world (e.g. team score), give these a fixed priority - they will always send if they have changed

Writing a Game Network Protocol

- TCP
- UDP
- MTU
- Replication
- Object Naming
- **Packet Structure**
- Quantization
- Prediction, Rollback & Correction
- Case Study: Quake

Possible Packet Structure

1. Begin filling the packet with *reliable* **Event Objects**
2. Continue filling the packet with *unreliable* **Event Objects**
3. Finally, fill the remainder of the packet with **State Objects**

If you have too many reliable events, then you’re doing something wrong. They are a hog.

Remember, you need to keep sending them until they have been acknowledged.

Writing a Game Network Protocol

- TCP
- UDP
- MTU
- Replication
- Object Naming
- Packet Structure
- **Quantization**
- Prediction, Rollback & Correction
- Case Study: Quake

Definition

“...restrict the number of possible values of (a quantity) or states of (a system) so that certain variables can assume only certain discrete magnitudes.”

Writing a Game Network Protocol

- TCP
- UDP
- MTU
- Replication
- Object Naming
- Packet Structure
- **Quantization**
- Prediction, Rollback & Correction
- Case Study: Quake

Quantizing Fields

- An integer with values from [1,100] can fit in 1 byte, instead of 4 bytes
- A float from [-100,+100] might be quantized to 2 decimal places

Naive way:

50.23543 x 100 = 5023.543

Serialize 5023 as 2 byte integer

Recipient:

5023 / 100 = 50.23

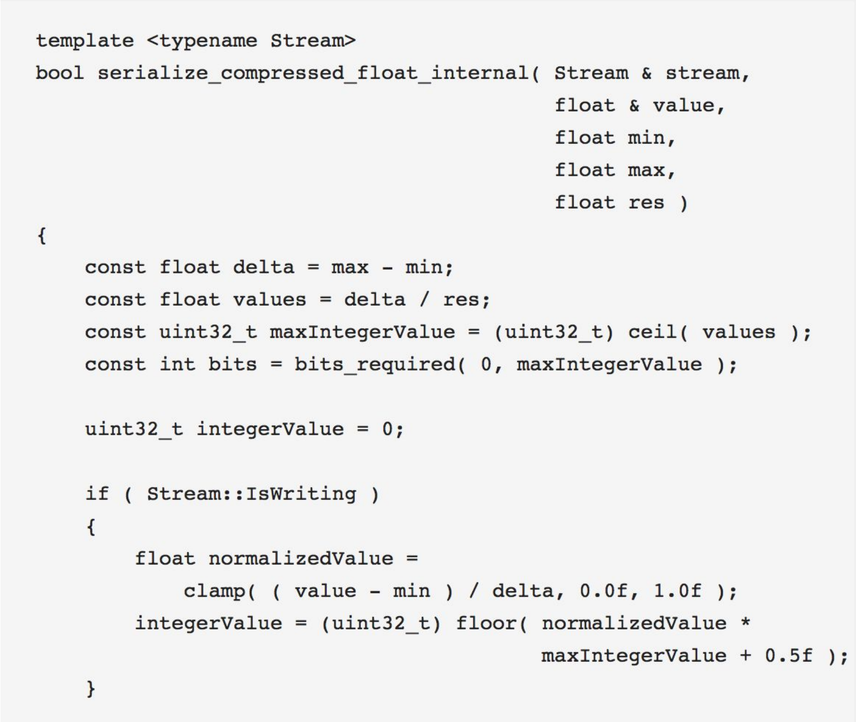
Writing a Game Network Protocol

- TCP
- UDP
- MTU
- Replication
- Object Naming
- Packet Structure
- **Quantization**
- Prediction, Rollback & Correction
- Case Study: Quake

Be Careful

- Quantization needs to happen on both sides
- If recipient is simulating the game with a value of 50.23 then the sender should be doing the *exact same thing* (50.23 != 50.23543)

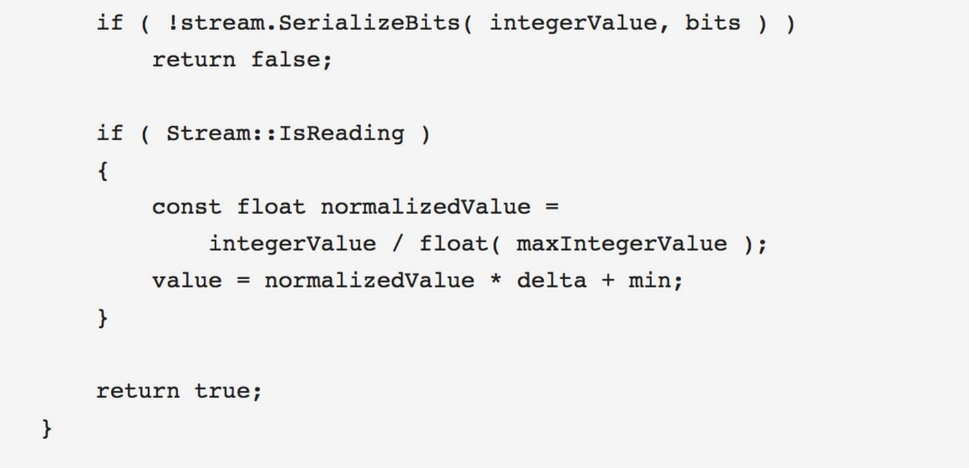
Writing a Game Network Protocol



```
template <typename Stream>
bool serialize_compressed_float_internal( Stream & stream,
                                          float & value,
                                          float min,
                                          float max,
                                          float res )
{
    const float delta = max - min;
    const float values = delta / res;
    const uint32_t maxIntegerValue = (uint32_t) ceil( values );
    const int bits = bits_required( 0, maxIntegerValue );

    uint32_t integerValue = 0;

    if ( Stream::IsWriting )
    {
        float normalizedValue =
            clamp( ( value - min ) / delta, 0.0f, 1.0f );
        integerValue = (uint32_t) floor( normalizedValue *
                                         maxIntegerValue + 0.5f );
    }
```



```
    if ( !stream.SerializeBits( integerValue, bits ) )
        return false;

    if ( Stream::IsReading )
    {
        const float normalizedValue =
            integerValue / float( maxIntegerValue );
        value = normalizedValue * delta + min;
    }

    return true;
}
```

http://gafferongames.com/building-a-game-network-protocol/serialization-strategies/

Writing a Game Network Protocol

- TCP
- UDP
- MTU
- Replication
- Object Naming
- Packet Structure
- **Quantization**
- Prediction, Rollback & Correction
- Case Study: Quake

More Information

- There's a lot of detail here on how to save every last bit

http://gafferongames.com/building-a-game-network-protocol/serialization-strategies/

- On Skylanders SuperChargers we did not bit pack, instead we serialized values to the nearest byte and compressed the packet in one go before sending

Writing a Game Network Protocol

- TCP
- UDP
- MTU
- Replication
- Object Naming
- Packet Structure
- Quantization
- **Prediction, Rollback & Correction**
- Case Study: Quake

Problem Statement

- Given a Strict Client-Server model
- I want to apply inputs locally and see a response on my screen immediately
- If i update the local state of my player, it will probably diverge from the server's simulation of my player over time
- How do we reconcile the two?

Writing a Game Network Protocol

- TCP
- UDP
- MTU
- Replication
- Object Naming
- Packet Structure
- Quantization
- **Prediction, Rollback & Correction**
- Case Study: Quake

Solution

1. I apply inputs and simulate my state
2. I keep snapshots of my last N states over time in a history buffer
3. When the server receives my inputs, my state is calculated and the result is **sent back to me**
4. When i receive the server snapshot of my state, it’s at time T-5 (say) in the **past**
5. I must *rollback* my local history buffer to time T-5, compare against the authoritative state and *correct* if there’s too big of a difference
6. Lastly, I re-apply my states from T-4 to T

Writing a Game Network Protocol

- TCP
- UDP
- MTU
- Replication
- Object Naming
- Packet Structure
- Quantization
- **Prediction, Rollback & Correction**
- Case Study: Quake

Video

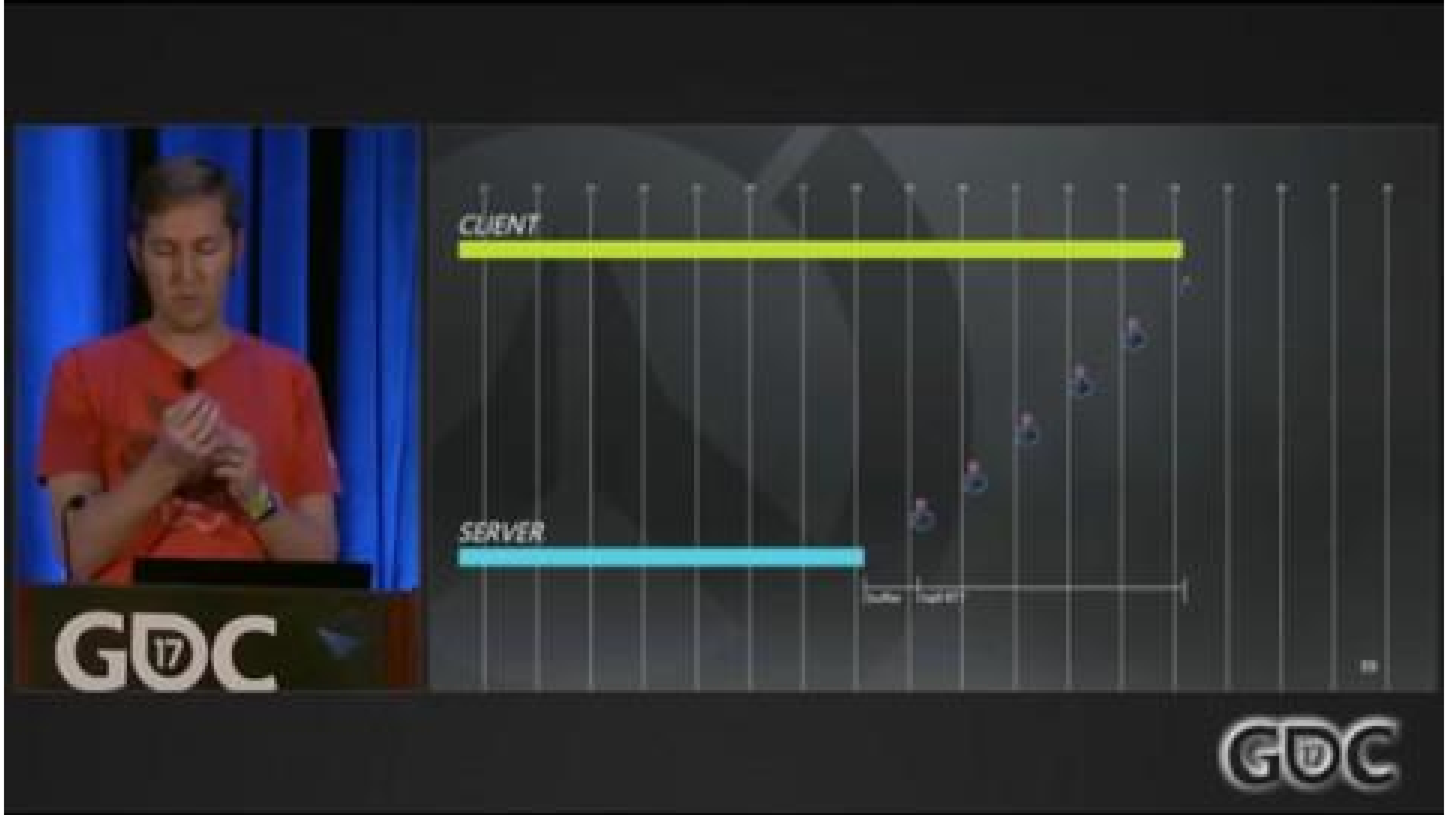
'Overwatch' Gameplay Architecture and Netcode (requires GDC Vault access)

http://www.gdcvault.com/play/1024001/-Overwatch-Gameplay-Architecture-and

Also awesome:

Developer Update | Let's Talk Netcode | Overwatch

https://www.youtube.com/watch?v=vTH2ZPgYujQ



CLIENT
SERVER

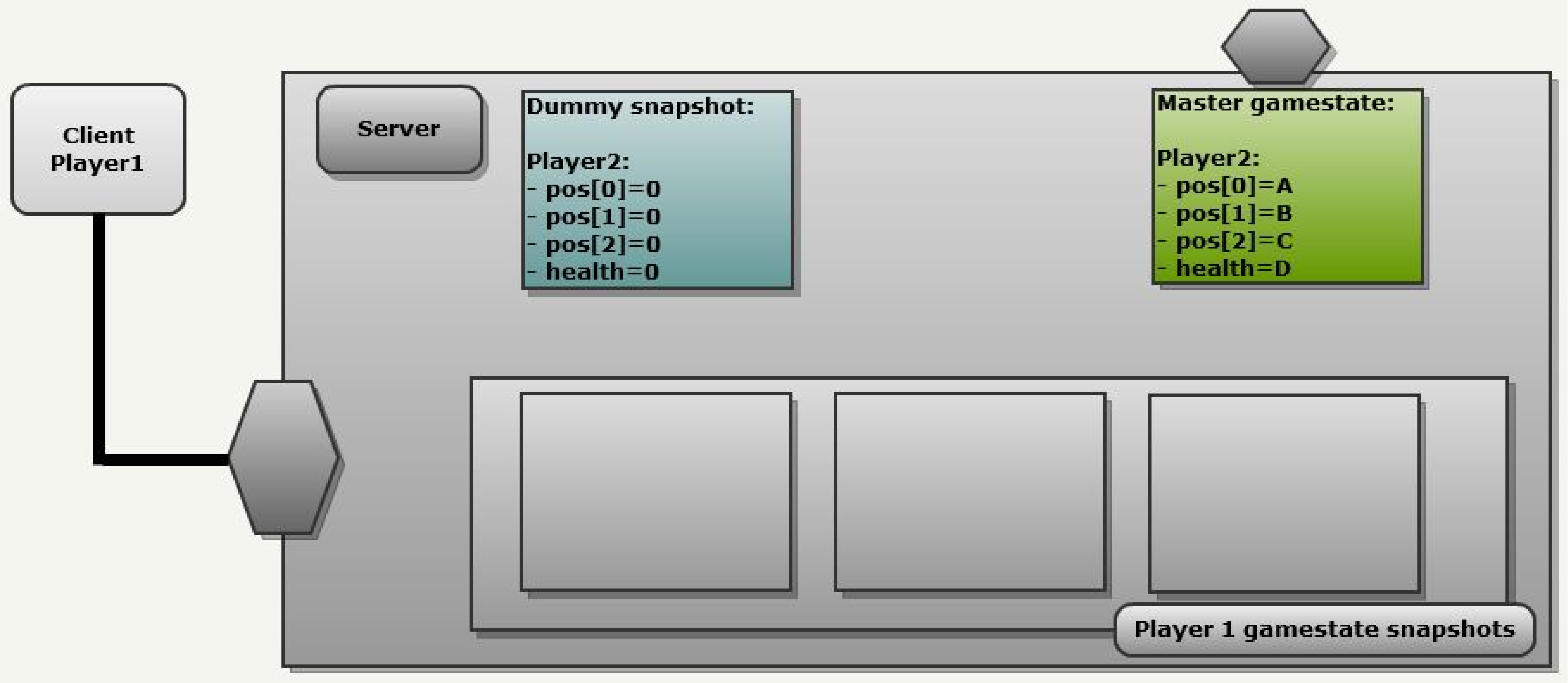
Writing a Game Network Protocol

- TCP
- UDP
- MTU
- Replication
- Object Naming
- Packet Structure
- Quantization
- Prediction, Rollback & Correction
- **Case Study: Quake**

Strict-Client Server Gold Standard

- Let’s take a quick look at snapshot synchronization

http://fabiensanglard.net/quake3/network.php



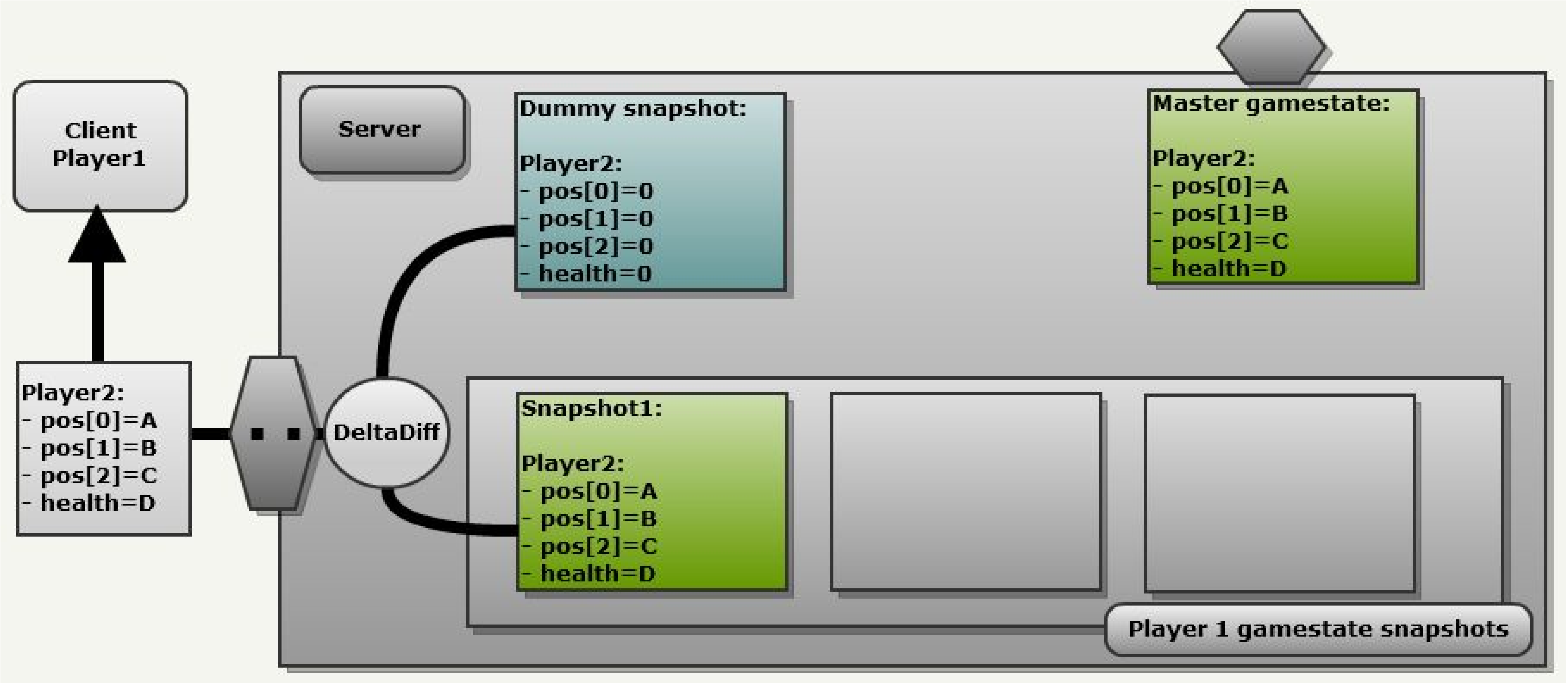
Dummy snapshot:

Player2:
- pos[0]=0
- pos[1]=0
- pos[2]=0
- health=0

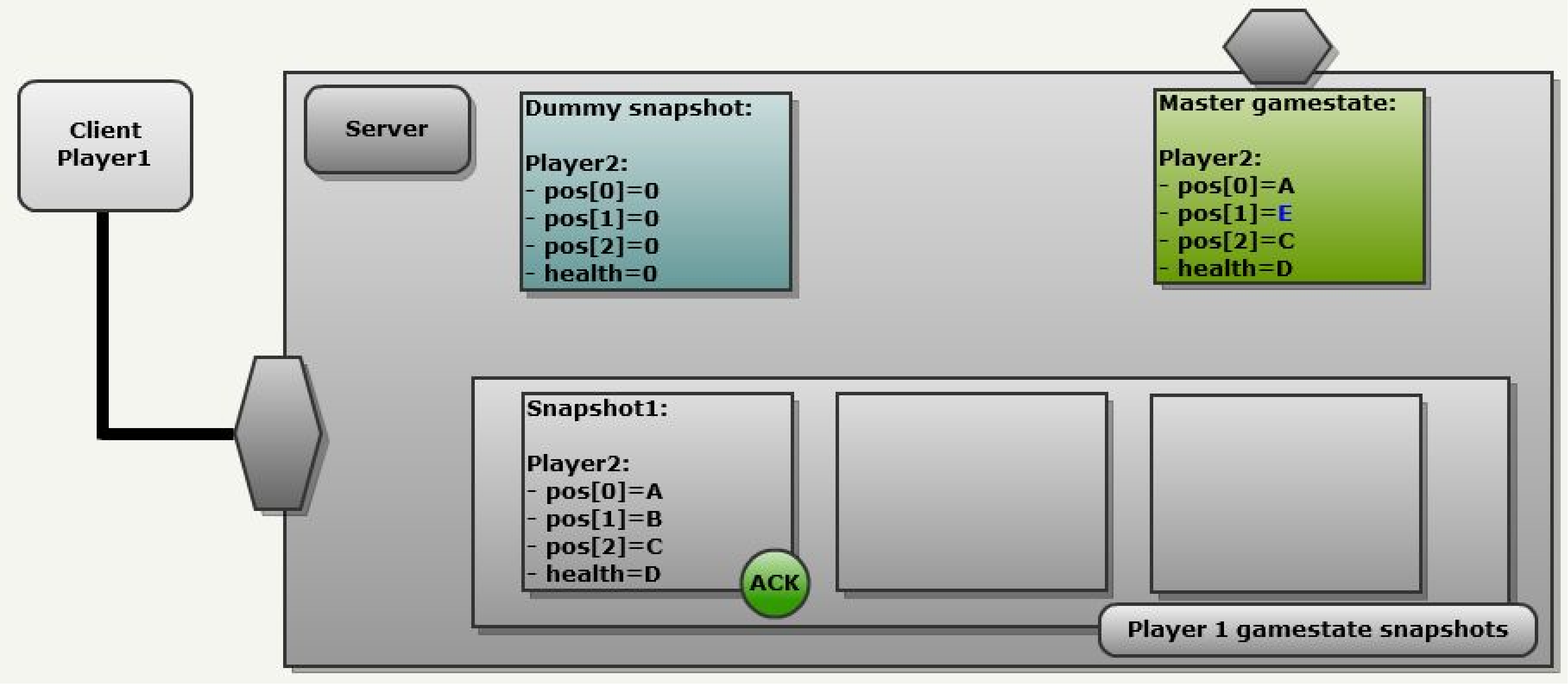
Master gamestate:

Player2:
- pos[0]=A
- pos[1]=B
- pos[2]=C
- health=D

http://fabiensanglard.net/quake3/network.php



http://fabiensanglard.net/quake3/network.php



Client Player1

Server

Dummy snapshot:

Player2:
- pos[0]=0
- pos[1]=0
- pos[2]=0
- health=0

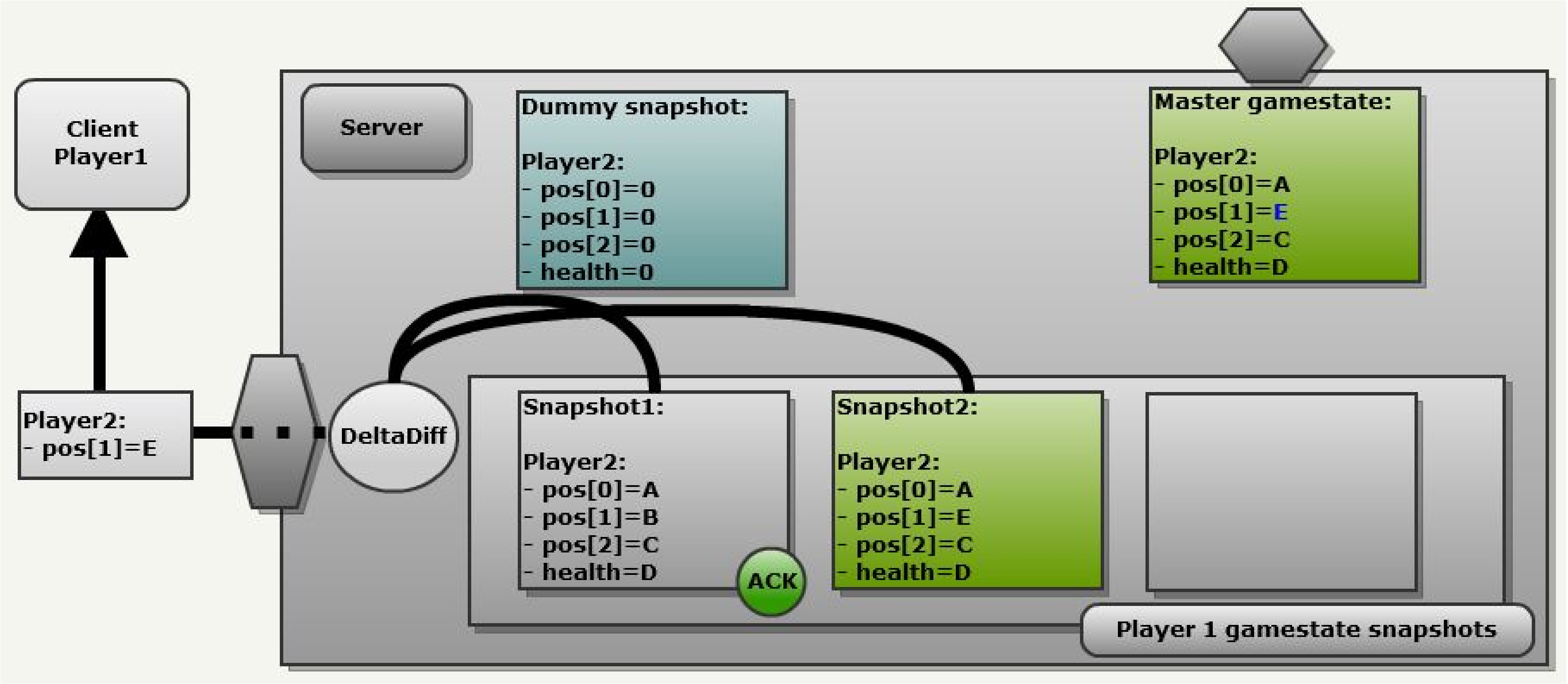
Master gamestate:

Player2:
- pos[0]=A
- pos[1]=E
- pos[2]=C
- health=D

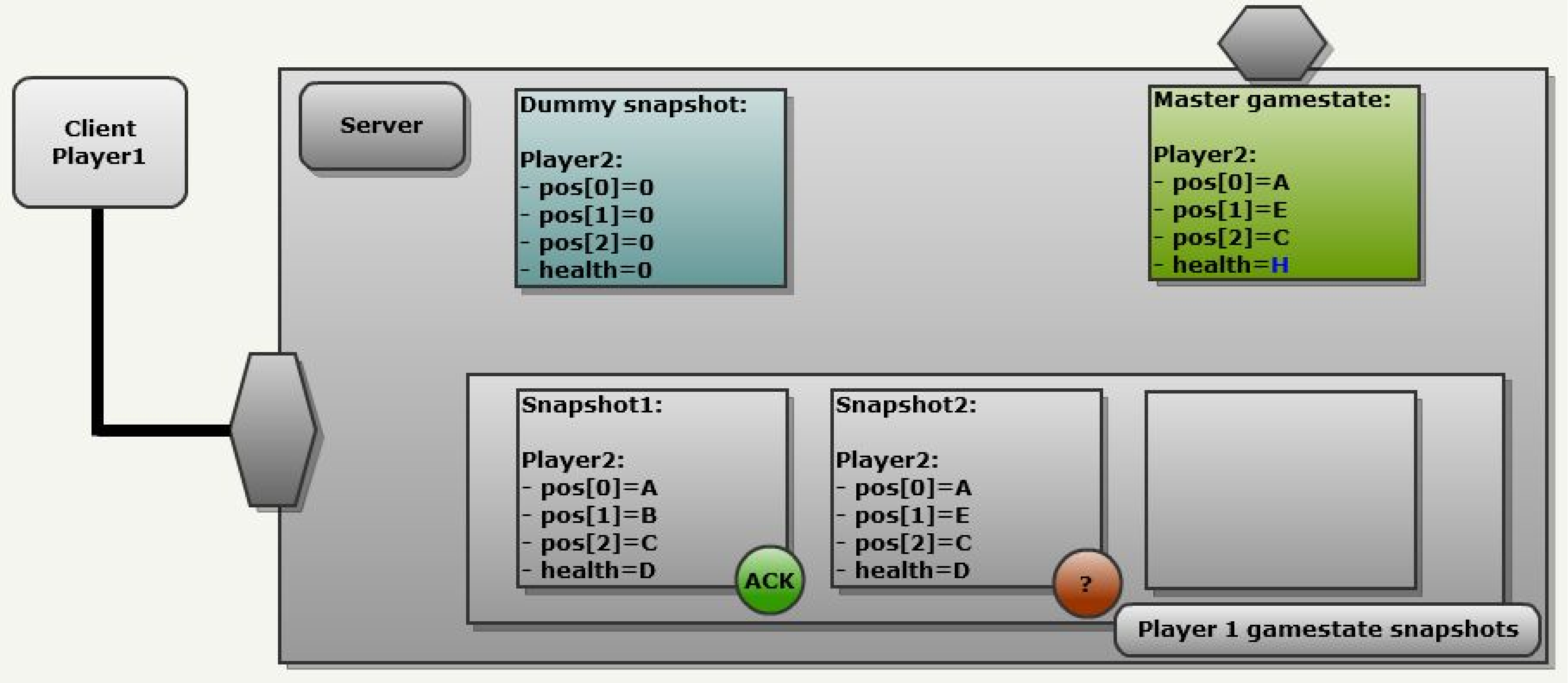
Snapshot1:

Player2:
- pos[0]=A
- pos[1]=B
- pos[2]=C
- health=D

http://fabiensanglard.net/quake3/network.php



http://fabiensanglard.net/quake3/network.php



Client Player1

Server

Dummy snapshot:

Player2:
- pos[0]=0
- pos[1]=0
- pos[2]=0
- health=0

Master gamestate:

Player2:
- pos[0]=A
- pos[1]=E
- pos[2]=C
- health=H

Snapshot1:

Player2:
- pos[0]=A
- pos[1]=B
- pos[2]=C
- health=D

ACK

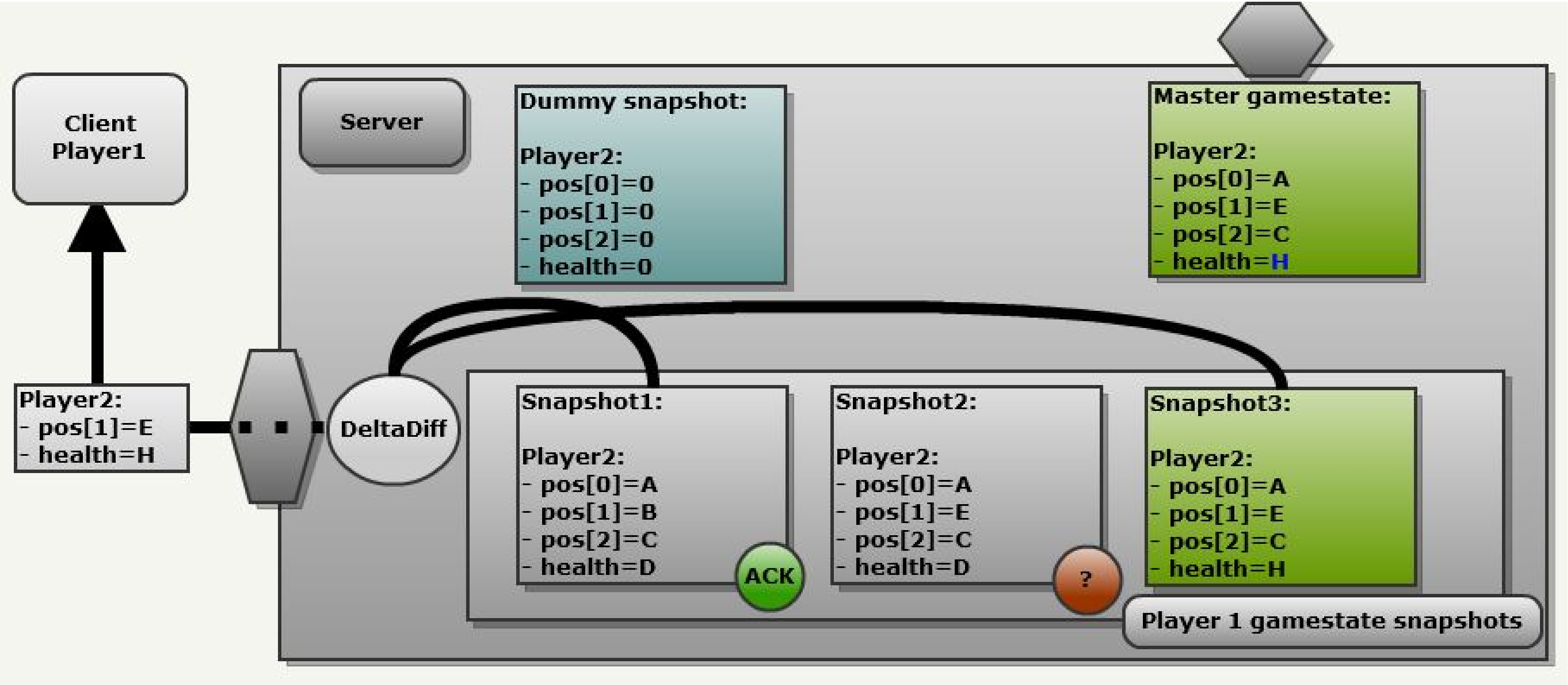
Snapshot2:

Player2:
- pos[0]=A
- pos[1]=E
- pos[2]=C
- health=D

?

Player 1 gamestate snapshots

http://fabiensanglard.net/quake3/network.php



http://fabiensanglard.net/quake3/network.php

Writing a Game Network Protocol

- TCP
- UDP
- MTU
- Replication
- Object Naming
- Packet Structure
- Quantization
- Prediction, Rollback & Correction
- **Case Study: Quake**

Strict-Client Server Gold Standard

- Only send what changed
- Always send the most up-to-date data

If a variable changes from 1 to 10 over 10 frames, say, the client may receive updates like:

{ 1, 2, 3, nothing, nothing, 6, 7, nothing, 9, 10 }

Client code should not rely on getting every intermediate state.

Client code needs to be resilient enough to react to the state of the world changing.

Today’s Agenda

- The Internet
- Network Topology Choices
- Writing a Game Network Protocol
- **Synchronizing Gameplay**

Networking Exercises

Assumptions

Generalized Client/Server

Replication

Events

Turn Invisible

You want your character to turn invisible on a button press.

```
bool _invisible;

// ...

if (button_pressed(A_BUTTON))
{
     set_invisible(true);

     on_invisible_ability();
}
```

Turn Invisible

Proposed solution:

```
Replicate
bool _invisible;

// ...

if (button_pressed(A_BUTTON) &&
     is_authority())
{
     set_invisible(true);
}

// ...

void on_invisible_changed()
{
     on_invisible_ability();
}
```

Simple Projectile

A shot that moves forward in a single direction.

```
void fire_projectile(
    vector_3 start,
    vector_3 direction)
{
    // spawn a projectile at start,
    // move it toward direction
}

// ...

fire_projectile(_position, _forward);
```

Simple Projectile

Proposed solution:

```
class fire_projectile_event
{
     vector_3 _start;
     vector_3 _direction;
     void receive()
     {
          fire_projectile(
               _start,
               _direction);
     }
}

// ...

send_network_event(
     new fire_projectile_event(
          _position,
          _forward));
```

Exploding Mine

You have a mine that explodes in proximity.

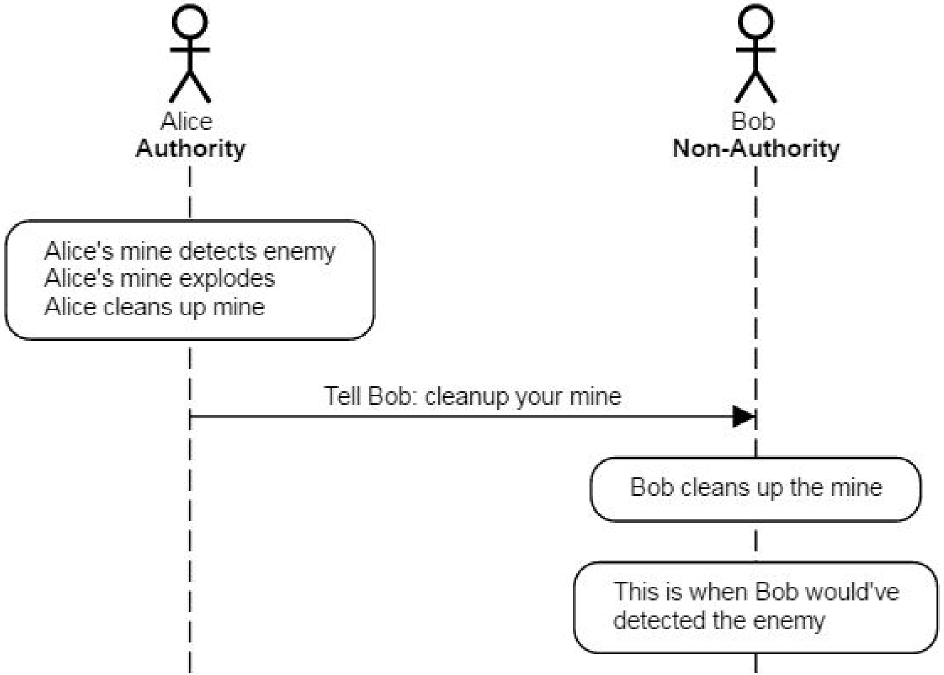
```
void update()
{
    if (enemy_in_range())
    {
        explode();
    }
}

// ...

void explode()
{
    do_explosive_damage();

    cleanup_mine();
}
```

Exploding Mine



Alice
Authority

Bob
Non-Authority

Alice's mine detects enemy
Alice's mine explodes
Alice cleans up mine

Tell Bob: cleanup your mine

Bob cleans up the mine

This is when Bob would've detected the enemy

```
void update()
{
        if (enemy_in_range())
        {
                explode();
        }
}

// ...

void explode()
{
        do_explosive_damage();

        cleanup_mine();
}
```

Exploding Mine

Proposed solution:

```
Replicate
bool _should_explode;

bool _has_exploded;

void update()
{
    if (enemy_in_range() && is_authority())
    {
        set_should_explode(true);
    }

    if (_should_explode && !_has_exploded)
    {
        explode();
        set_has_exploded(true);
    }
}

// continued...
```

Exploding Mine

Proposed solution:

```
// continued...

void explode()
{
     do_explosive_damage();

     set_invisible(true);

     // Set up a timer to call cleanup_mine
     // in 1 second
     setup_timer(cleanup_mine, 1.0f);
}
```

Fast, Constant Projectiles

A steady stream of projectiles that move forward in a single direction.

```
if (enemy_in_range() && cooldown_passed())
{
     fire_projectile();
}
```

Fast, Constant Projectiles

Proposed solution:

```
Replicate
bool _should_fire;

// ...

if (is_authority())
{
    if (enemy_in_range() &&
        cooldown_passed())
    {
        set_should_fire(true);
    }
    else
    {
        set_should_fire(false);
    }
}

if (_should_fire)
{
    fire_projectile();
}
```

Slow, Homing Projectile

A slow projectile that homes in on the closest target.

```
entity _homing_target;
projectile _homing_projectile;

if (_homing_target && _homing_projectile)
{
     _homing_projectile.move_toward(
           _homing_target);
}
```

Slow, Homing Projectile

Proposed solution:

```
Replicate
entity _homing_target;
Replicate
projectile _homing_projectile;

if (_homing_target && _homing_projectile)
{
     _homing_projectile.move_toward(
           _homing_target);
}
```