Voices from Community - T-shirt Style

- T-shirt is in manufacturing, expected to be sent out in 3 weeksz

- Please add Wechat “小秘书-阿曼达” in wechat groupchat, for shipping information, if you are not in our wechat group, please direct message to “GAMES-WEBINAR” Bilibili account

- Other ways to get our t-shirt: we will sent T-shirts to community members who finished our homework over 2 times

- Deadlines of all Homework 1, 2, 3, 4, 00:00 , 2022/08/31

Rewarding list for last event:

@大绒毛怪, @元宇宙羊, @SHtech、BoBo,

@尤水就下, @秋后算命, @今天好蓝瘦,

@TheBigFrog, @waterroomwater, @rmlx, @huangbaichao

Voices from Community - Great Comments

小引擎小巧且灵动自由的在天空舞蹈

TheBigFrog

老师体能太好了，一下讲这么久，辛苦！

前两天重新看了下宫崎骏的动画《红猪》，突然发现给红猪造飞机的工作室就叫 Piccolo，而主创设计师是 Piccolo家族的 Fio Piccolo（一位17岁的女孩），突然感觉我们的小引擎确确实实也就像主角的座驾一样，小巧且灵动，自由的在天空中舞蹈；而课程组也就像 Fio Piccolo 一样，年轻、充满活力与创造力，让我们得以有这么好的实践机会；

同时想听听游戏引擎在混合现实方面有没有什么针对性的或者新奇的发展，毕竟GAMES全称就是 计算机图形学与混合现实在线平台，比如说我认为一个好的混合现实项目应当不仅仅是我们通过游戏引擎往MR设备里面去加东西，同时也应该允许游戏引擎去反向控制我们身边的各种智能化设备，以对游戏里的事件在现实世界产生对应的反馈，这里可能就涉及了太多方面，比如计算机视觉、三维重建、物联网、各种智能设备的通讯和控制，这会是一个很有趣的话题。

2022-07-27 00:22 8 回复

UP主觉得很赞

搭建游戏引擎是一件浪漫的事

秋后算命

游戏引擎真是越学越觉得自己知识不够用，一个大型3d游戏引擎真的可以说是包容万象了。

每次想到很多跨领域的研究成果都能在游戏引擎上得到一席之地，就觉得搭建游戏引擎是一件很浪漫的事。

2022-07-27 10:29 1 回复

我认识到的引擎不仅是世界的科学技术，更是带着镣铐起舞的艺术

waterroomwater

感谢王希老师和课程组同学的辛勤付出，提供了这么高质量的课程！

作为一名在校学生，games104让我对周一充满了期待，有时候希望周末赶快过去hhh。目前主要在学习骨骼动画这个小分支，通过104的课程学习，不仅给我动画模块更深入的认知，也培养了我从整体上思考一个功能模块在引擎系统中如何更好发挥作用的习惯。

经历了这几个月的"追课"，我的思考模式渐渐从"如何完成一项功能"到"如何在资源有限的情况下更好地完成功能"的转变，我认识到引擎不仅是创造世界的科学技术，更是戴着镣铐起舞的艺术，在实际开发过程中要考虑如何更合理分配计算资源。

（在20万播放的时候王希老师撸了一个Q&A番外，也希望以后有机会能看到整个课程组准备课程的幕后花絮~ 最后，我也想要T恤，以后开组会穿上，成为全场最靓的仔

2022-07-29 19:59 4 回复

Q&A

- Q1: What's your opinion on the future trending on deep learning and reinforcement learning?
- Q2: Does the AI module need modification with game version updates?
- Q3: Could you explain DLSS to us?

Lecture 18

Online Gaming Architecture

Fundamentals

WANG XI

GAMES 104

2022

PLAY ANYWHERE WITH ANYONE

Game developers have never stopped exploring multiplayer online gaming

LET'S ENJOY IT !

Challenges in Multiplayer Online Gaming (1/5)

Consistency

- Network Synchronization

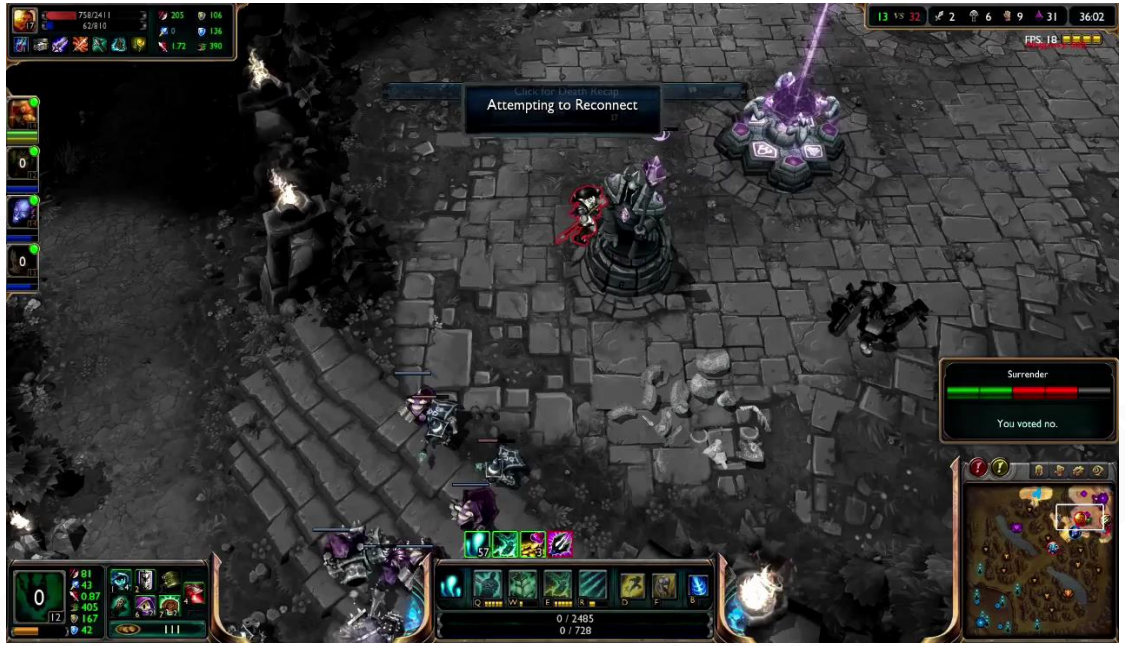
Different Behavior in Online Game

Challenges in Multiplayer Online Gaming (2/5)

Reliability

- Network Latency
- Drop and Reconnect

Latency Impact



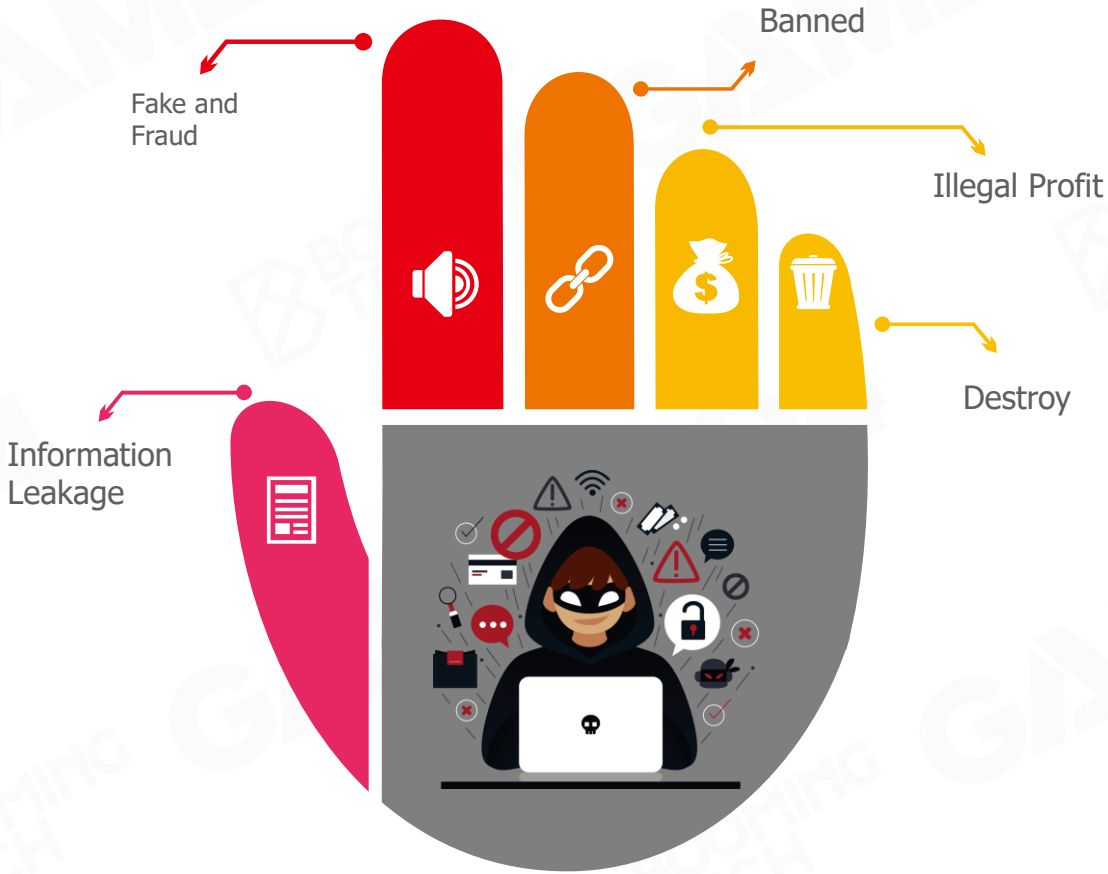
Attempting to Reconnect...

Challenges in Multiplayer Online Gaming (3/5)

Security

- Cheats
- Accounts Hacked

Cheating Program

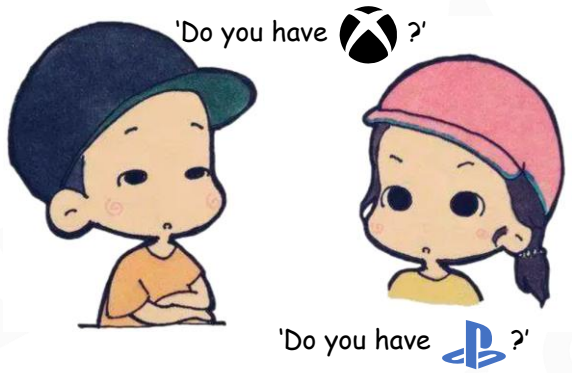


Accounts Hacked

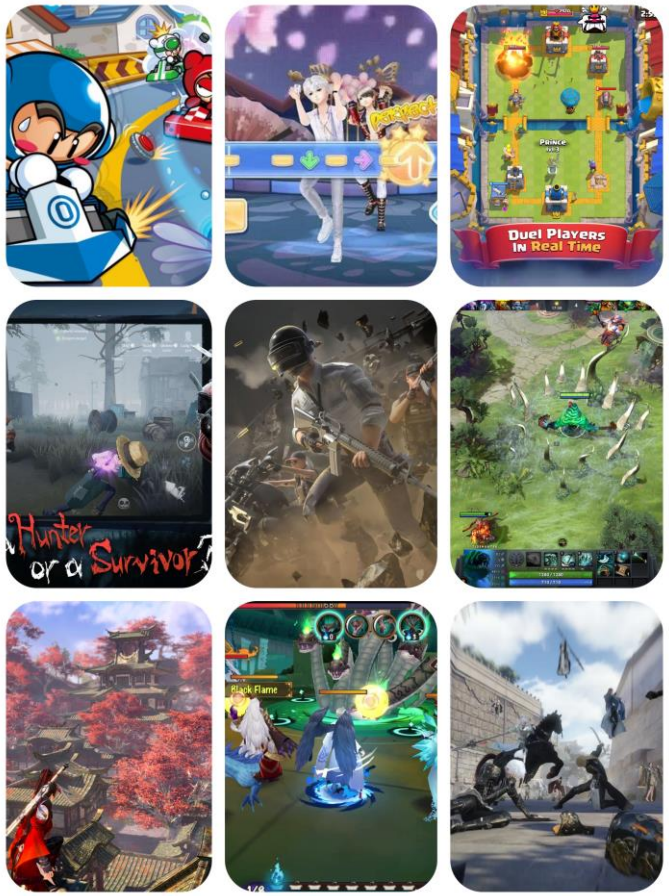
Challenges in Multiplayer Online Gaming (4/5)

Diversities

- Cross-Play
- Rapid iteration
- Multiple Game Systems



How to Play Together
in Different Platforms?



Variety of Game Types

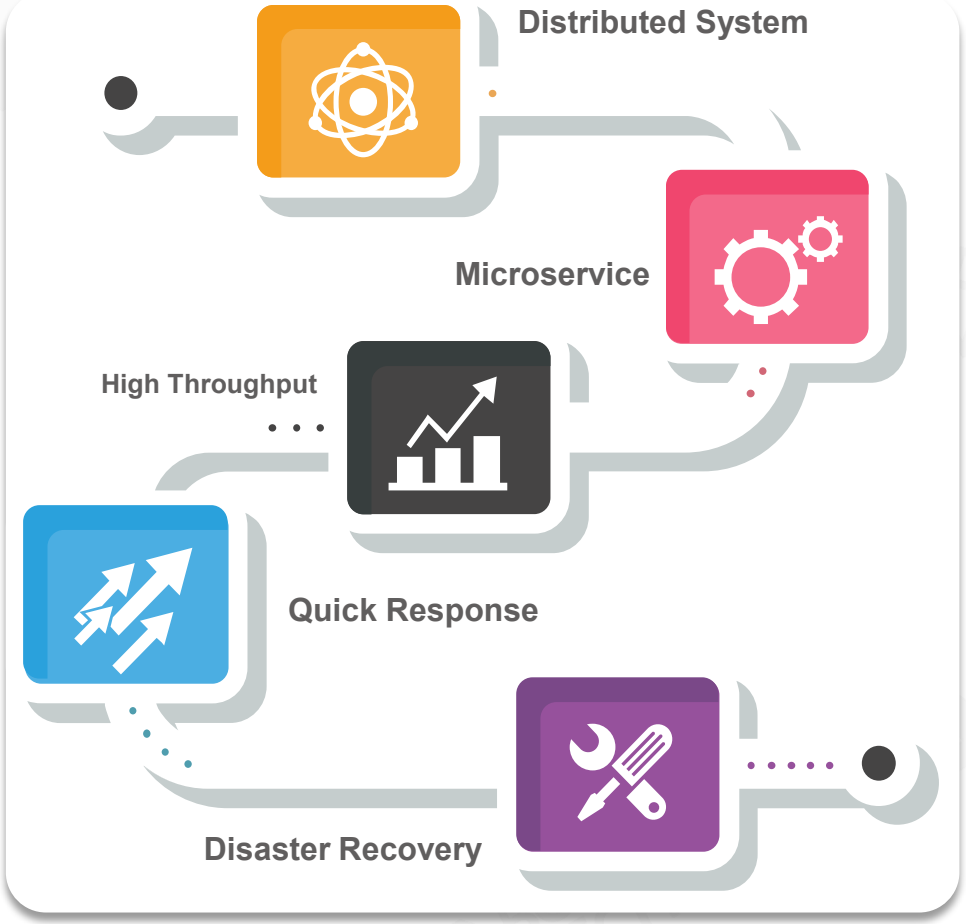
Game Sub-Systems

Challenges in Multiplayer Online Gaming (5/5)

Complexities

- High Concurrency
- High Availability
- High Performance

Ready Player One, Massive Players



Outline

01.

Basics

- Network Protocols
  - TCP, UDP and Reliable UDP
- Clock Synchronization
- Remote Procedure Call (RPC)
- Network Topology
- Game Synchronization
  - Snapshot Sync.
  - Lockstep Sync.
  - State Sync.

02.

Advanced

- Character Movement Replication
- Hit Registration
- Lag Compensation
- MMO Game Network Architecture
- AOI
- Anticheat
- The Future

Network Protocols

The Founding Fathers of the Internet

Designed the ***TCP/IP*** protocols and the internet architecture.

In 1977 Cerf and Kahn will link three networks(*packet radio, satellite, and the ARPANET*) and prove the efficacy of their ***TCP/IP*** protocol in a dramatic round-the-world transmission from a moving vehicle, the SRI Packet Radio Research van.

An SRI International Packet Radio Van, used for the first three-way internetworked transmission

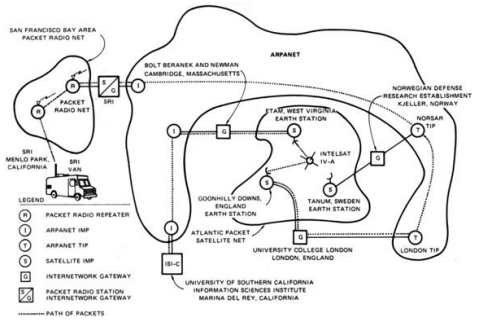
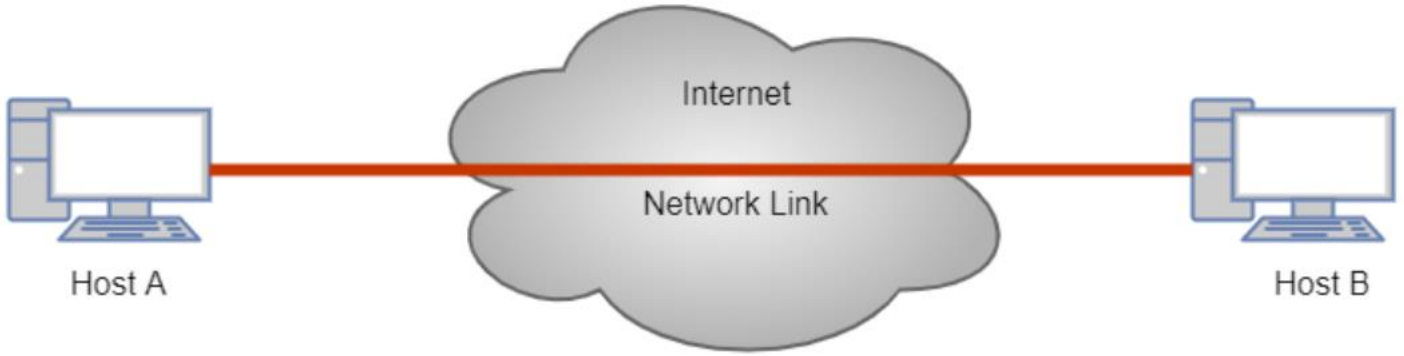


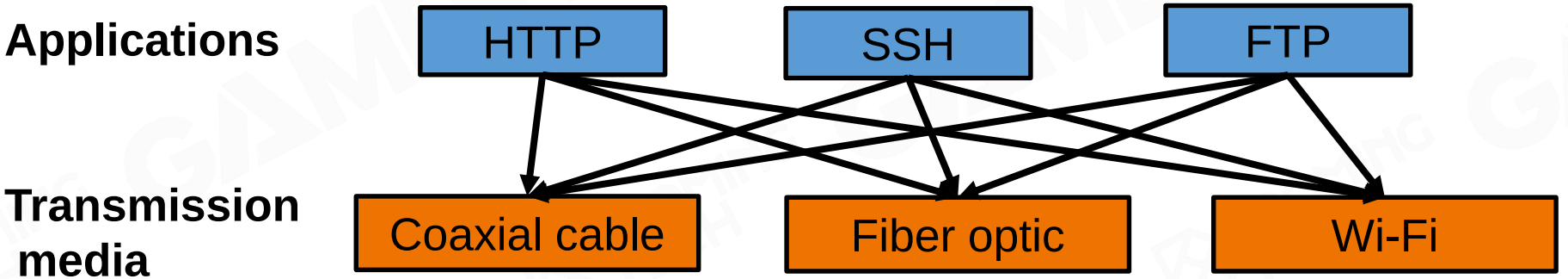
Diagram of the first internetworked connection

How to communication between two PCs

- A and B must agree on the meaning of the bits being sent and received at many different levels, including
  - How many volts represents a 0 bit, and for a 1 bit?
  - How does receiver know which is the last bit?
  - How many bits long is a number?



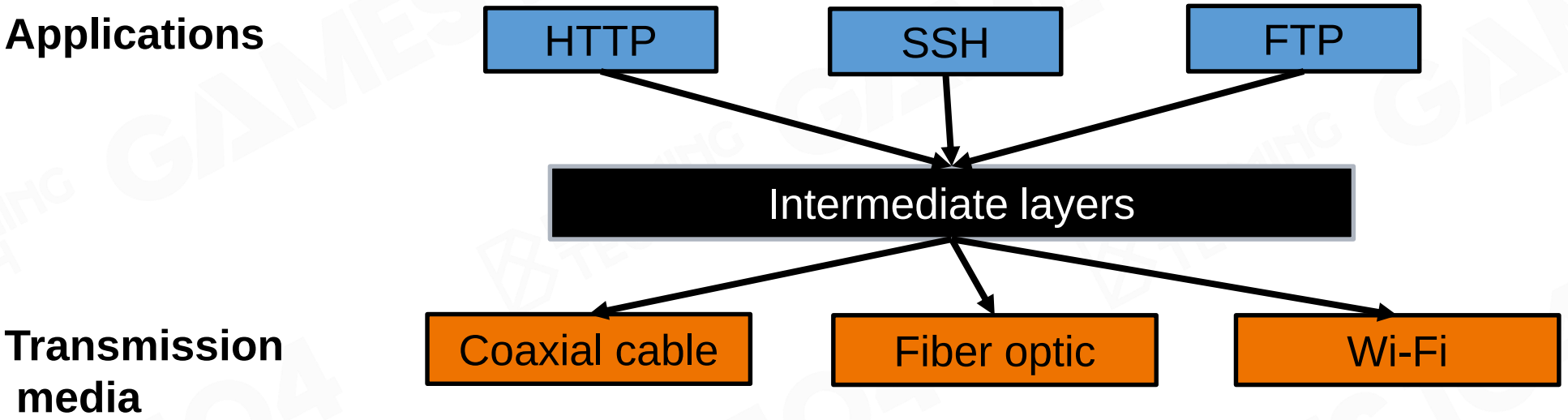
The Problem of Communication



- Re-implement every application for every new underlying transmission medium?
- Change every application on any change to an underlying transmission medium?

- No! But how does the Internet design avoid this?

Solution: Layering



- Intermediate layers provide a set of abstractions for applications and media
- New applications or media only need implementation for intermediate layer's interface

Layering in the Internet - OSI Model

Application
- Provides functions to users

Presentation
- Converts different representations

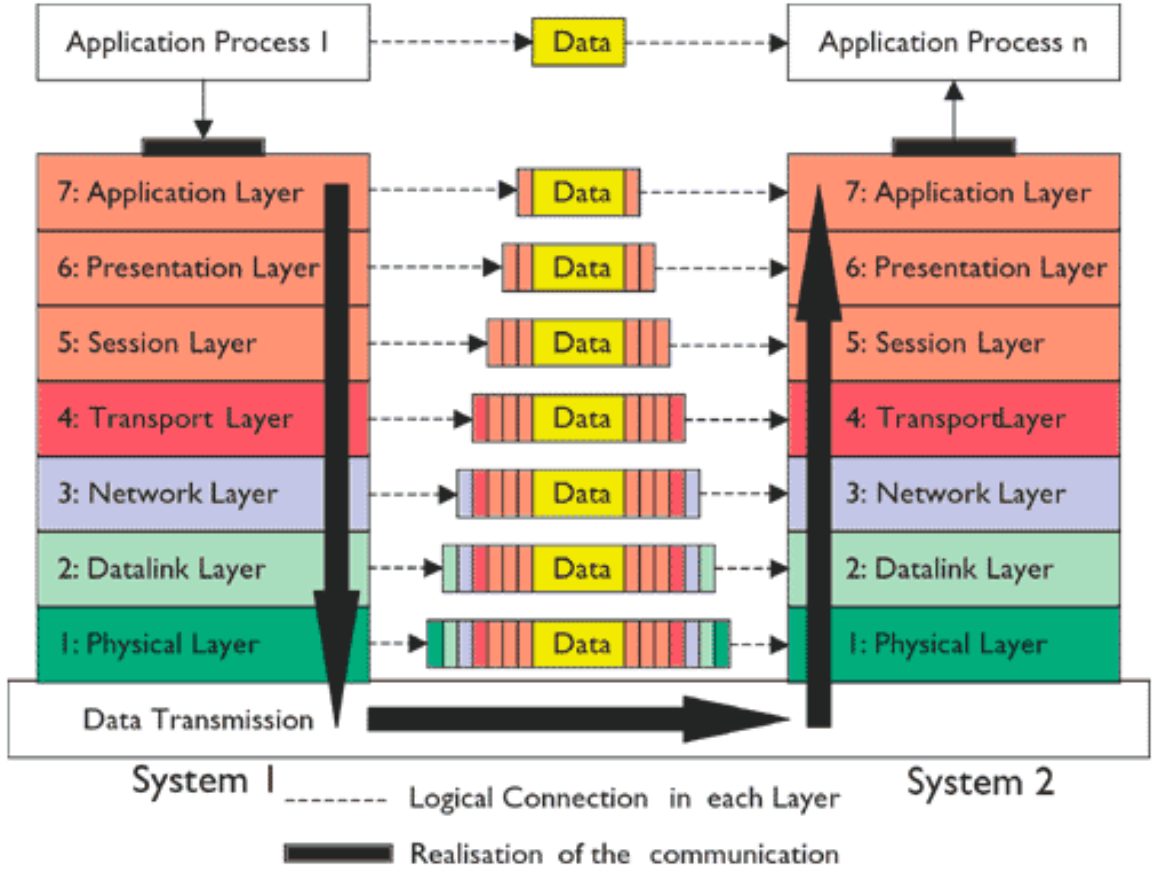
Session
- Manages task dialogs

Transport
- Provides end-to-end delivery

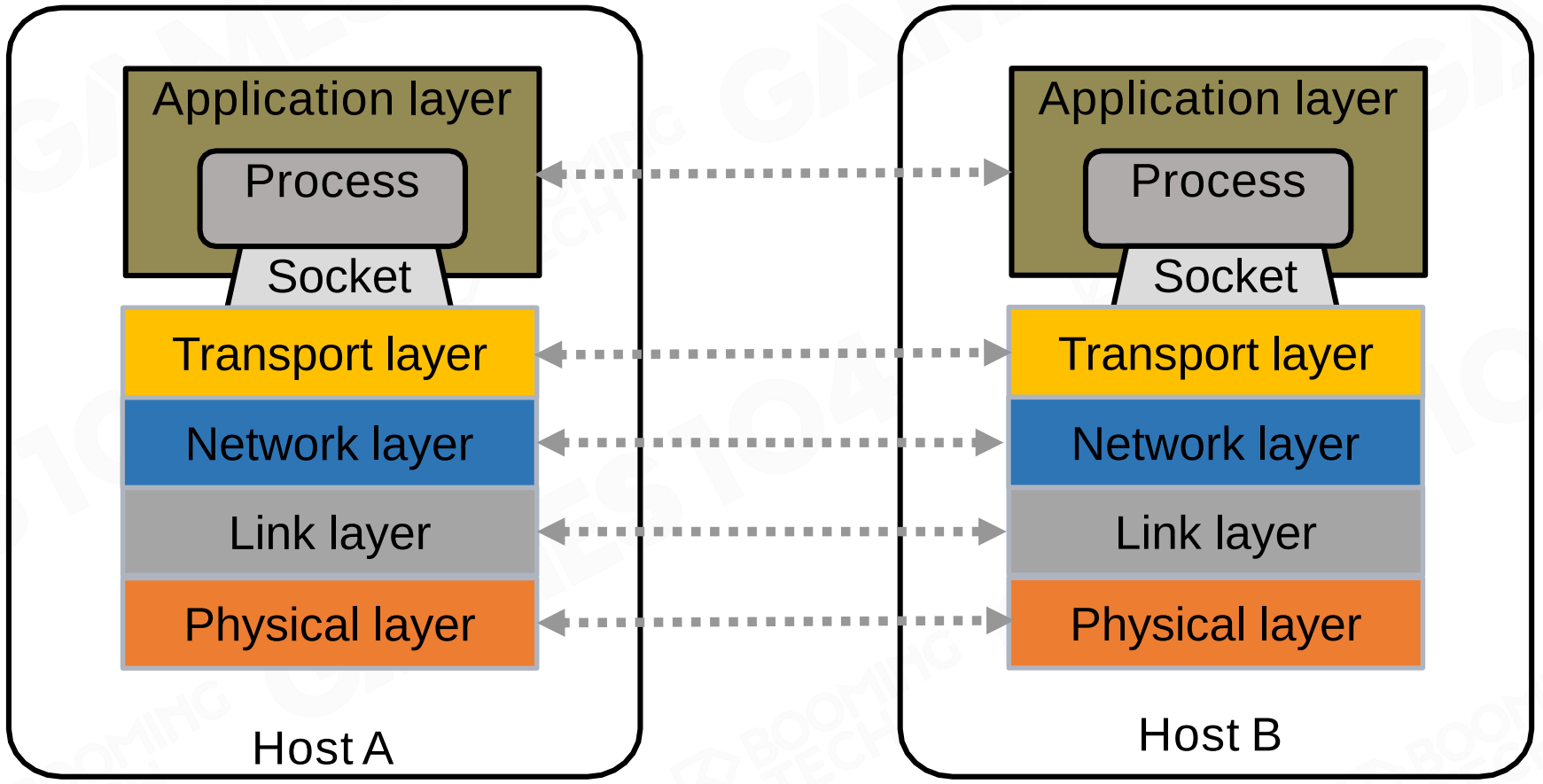
Network
- Sends packets over multiple links

Data Link
- Sends frames of information

Physical
- Sends bits as signals



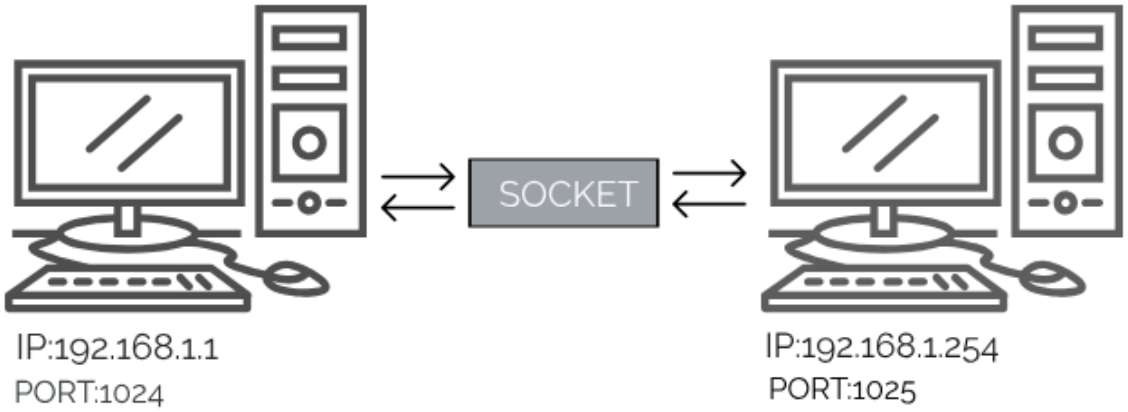
Network Socket-based Communication



Socket

A software structure within a network node of a computer network that serves as an endpoint for sending and receiving data across the network.

A Socket is combination of an IP Address and a Port Number.




```
// server side
struct sockaddr_in saddr, caddr;
int sockfd, clen, isock;
unsigned short port = 80;

if((sockfd=socket(AF_INET, SOCK_STREAM, 0) < 0) {
    printf("Error creating socket\n");
    ...
}

memset(&saddr, '\0', sizeof(saddr));        // zero structure out
saddr.sin_family = AF_INET;                 // match the socket() call
saddr.sin_addr.s_addr = htonl(INADDR_ANY);  // bind to any local address
saddr.sin_port = htons(port);               // specify port to listen on

if((bind(sockfd, (struct sockaddr *) &saddr, sizeof(saddr)) < 0) { // bind
    printf("Error binding\n");
    ...
}

if(listen(sockfd, 5) < 0) {                 // listen for incoming connections
    printf("Error listening\n");
    ...
}

clen=sizeof(caddr)
if((isock=accept(sockfd, (struct sockaddr *) &caddr, &clen)) < 0) { // accept one
    printf("Error accepting\n");
    ...
}
```

Setup Socket

Both client and server need to setup the socket

- Function

  int socket (int *domain*, int *type*, int *protocol*)

  - *domain*

    - AF_INET -- IPv4
    - AF_INET6 -- IPv6

      ...

  - *type*

    - SOCK_STREAM -- TCP
    - SOCK_DGRAM -- UDP

      ...

  - *protocol*

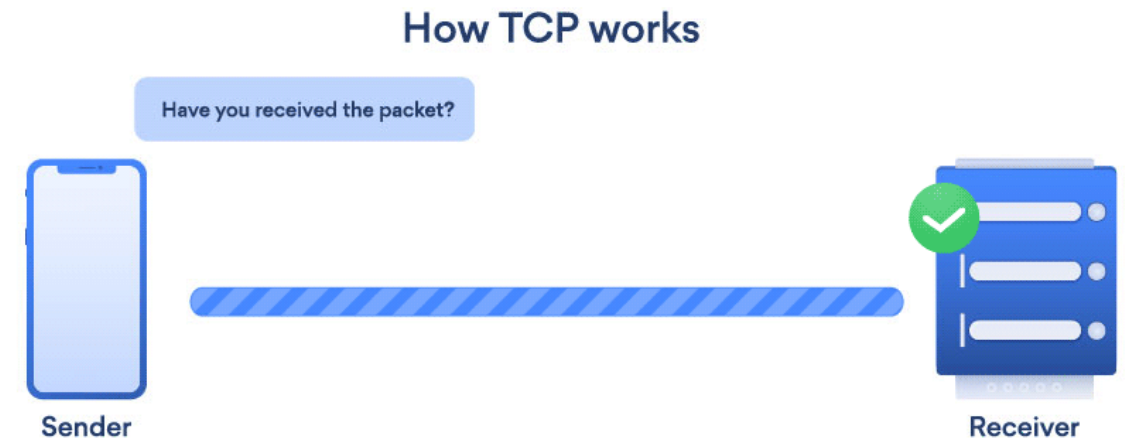
    - 0

- Eg

  int sockfd = socket(AF_INET, SOCK_STREAM, 0)

Transmission Control Protocol (TCP)

- Connection-Oriented
- Reliable and Ordered
- Flow Control
- Congestion Control



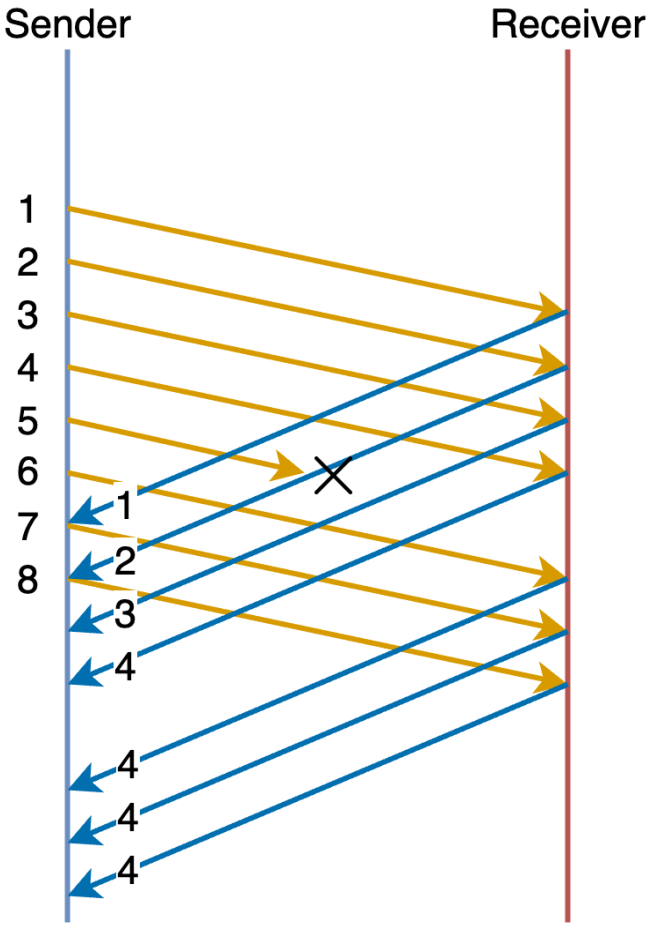
TCP segment header

Offsets	Octet	0								1								2								3							
Octet	Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset				Reserved 0 0 0				NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size														
16	128	Checksum																	Urgent pointer (if URG set)														
20	160	Options (if *data offset* > 5. Padded at the end with "0" bits if necessary.)																															
⋮	⋮																																
60	480																																

TCP Retransmission Mechanisms

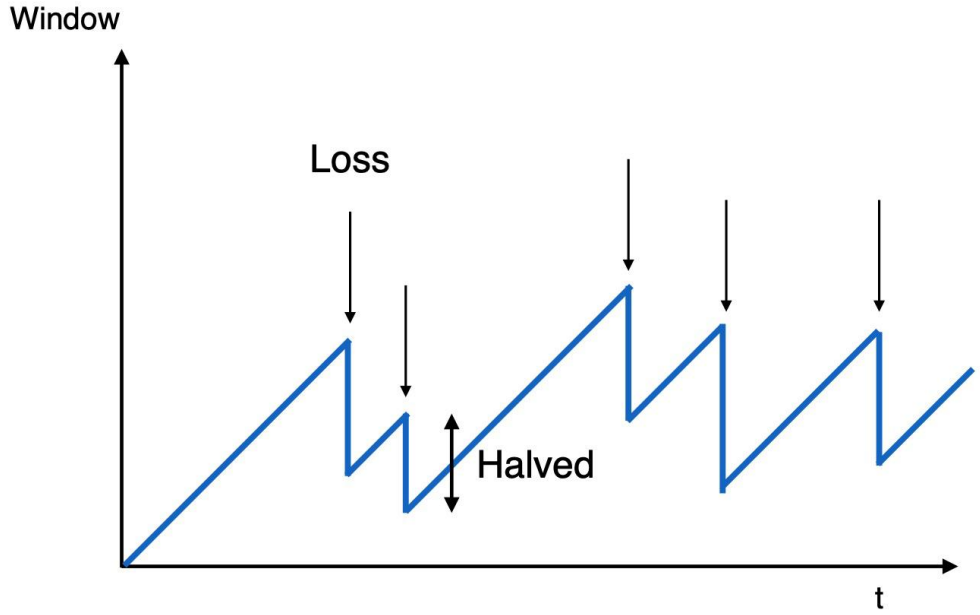
Duplicate ACKs

- Senders sends packets and seqnos
  - 1, 2, 3, 4, 5, 6, 7, 8
- Assume 5th packet (seqno 5) is lost, Stream of ACKs will be
  - 1, 2, 3, 4, 4, 4, 4



TCP Congestion Control

- The congestion window (**CWND**) of TCP starts to grow from a small value
- When congestion occurs, packet loss or timeout, CWND will be reduced according to a certain algorithm
- This leads to high delay and cause delay jitter



As the main transmission protocol on the Internet, TCP congestion control is necessary, otherwise it will cause congestion collapse. TCP congestion control is the main congestion control measure on the Internet, and it is also the main cause of TCP performance problems

User Datagram Protocol (UDP)

David P. Reed

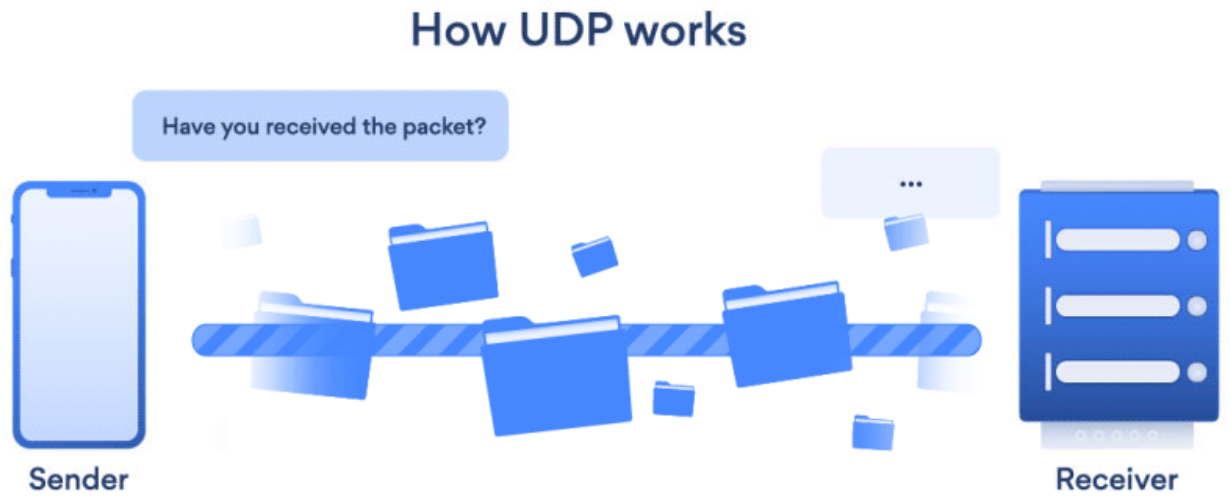
He was involved in the early development of TCP/IP, and was the designer of the ***User Datagram Protocol (UDP)***, though he finds this title "a little embarrassing".

He was also one of the authors of the original paper about the end-to-end principle, ***End-to-end arguments in system design***, published in 1984.

UDP Features

UDP (User Datagram Protocol)

- Connectionless
- UnReliable and Unordered
- NO Flow Control
- NO Congerstion Control



UDP datagram header

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port															
4	32	Length																Checksum															

Network Protocols Usage in Game

Game suitable for use

- TCP

- UDP

	TCP	UDP
Structure	Segments	Datagrams
Connection Model	Connection-Oriented	Connectionless
Speed	Slow	Fast
Reliability	Reliable	Unreliable
Header	20 Bytes	8 Bytes
Data Transfer	Ordered	Unordered
Control	Flow Control	NO Flow Control

Reliable UDP

TCP is Not Time Critical

- TCP is the complex and heavyweight protocol. It provides reliable delivery and advanced features, but it has more overhead
- TCP is a fair, traffic oriented protocol designed to improve bandwidth utilization. But it's not designed for speed

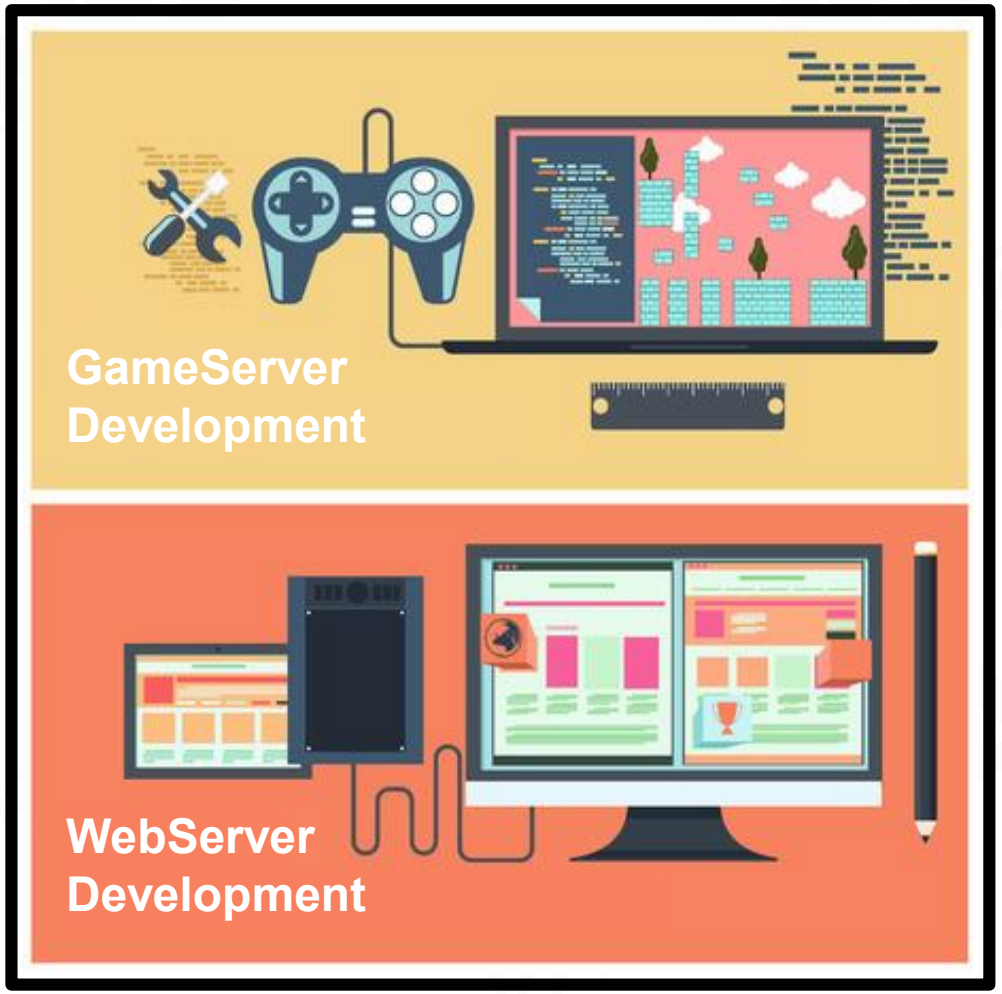
- So Why TCP is slow?

UDP is Fast but Unreliable

- UDP is lightweight and fast but unreliable, packet loss and disorder will occur
- How to achieve reliable and real-time communication?

Why We Need to Customize Protocol

- **Game Server**
  - Keep-alived connection (TCP)
  - Need keep logic consistency in “order” (TCP)
  - High responsive & low latency (UDP)
  - Broadcast commonly used (UDP)
- **Web Server**
  - Handles the HTTP protocol
  - Delivers static web content—e.g., HTML pages, files, images, video.



Acknowledgement & Sequence Number

- **Positive acknowledgment (ACK)** is a signal that is passed between communicating processes, computers, or devices to signify acknowledgment, or receipt of message
- **Negative ACK (NACK or NAK)** is a signal that is sent to reject a previously received message or to indicate some kind of error
- **Sequence number (SEQ)** is a counter used to keep track of every byte sent outward by a host
- **Timeouts** specified periods of time allowed to elapse before an acknowledgment is to be received

Automatic Repeat Request (ARQ)

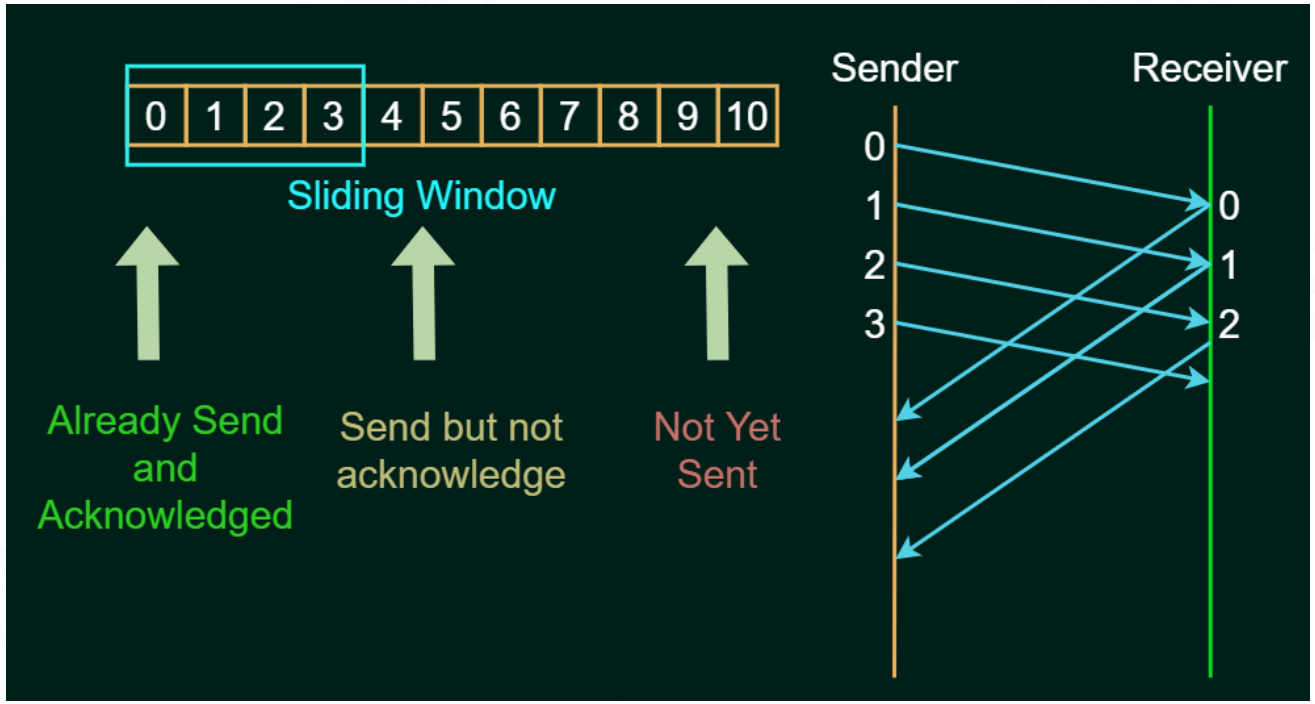
An error-control method for data transmission that uses ACK and timeouts to achieve reliable data transmission over an unreliable communication channel.

If the sender does not receive an acknowledgment before the timeout, it re-transmits the packet until it receives an acknowledgment or exceeds a predefined number of retransmissions

- Sliding window protocol
  - Stop-and-Wait ARQ
  - Go-Back-N ARQ
  - Selective Repeat ARQ

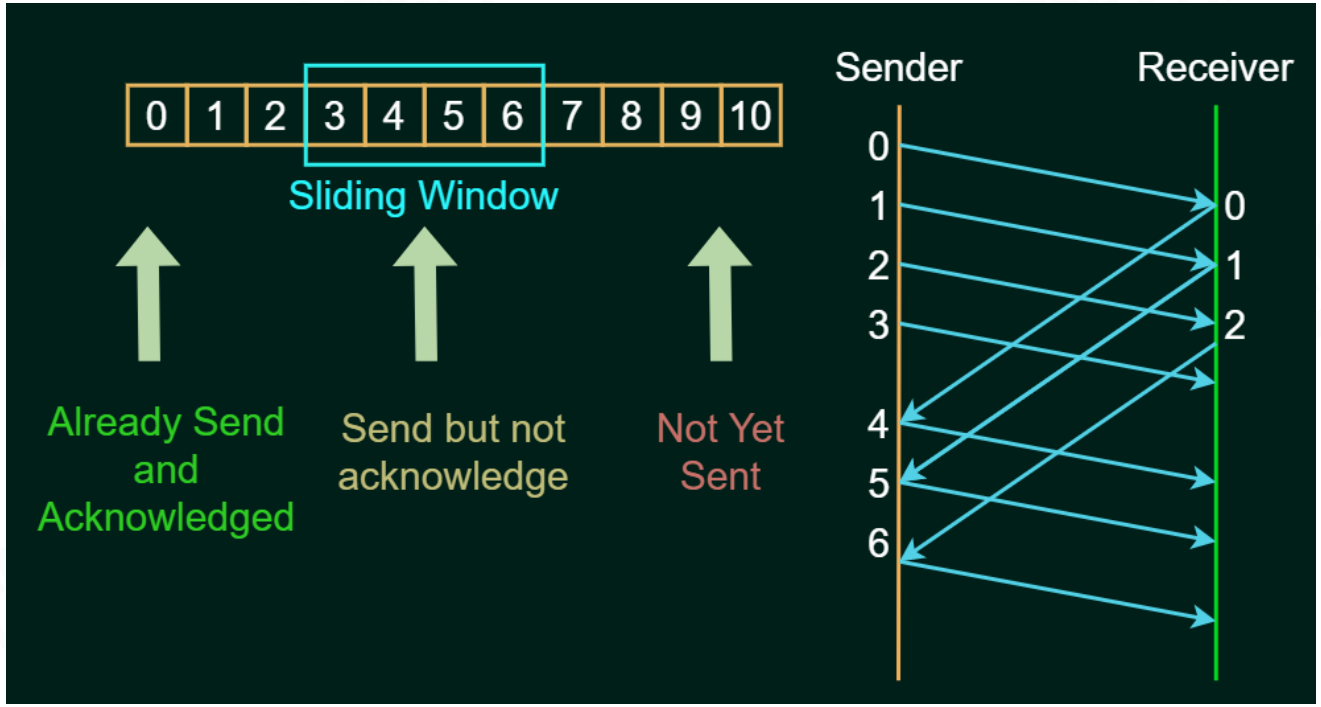
Sliding Window Protocol

- Send mutilple frames at a time, number of frames to be sent is based on **Window size**
- Each frame is numbered by Sequence number
- When the frame at the front of the window is received, the window slides



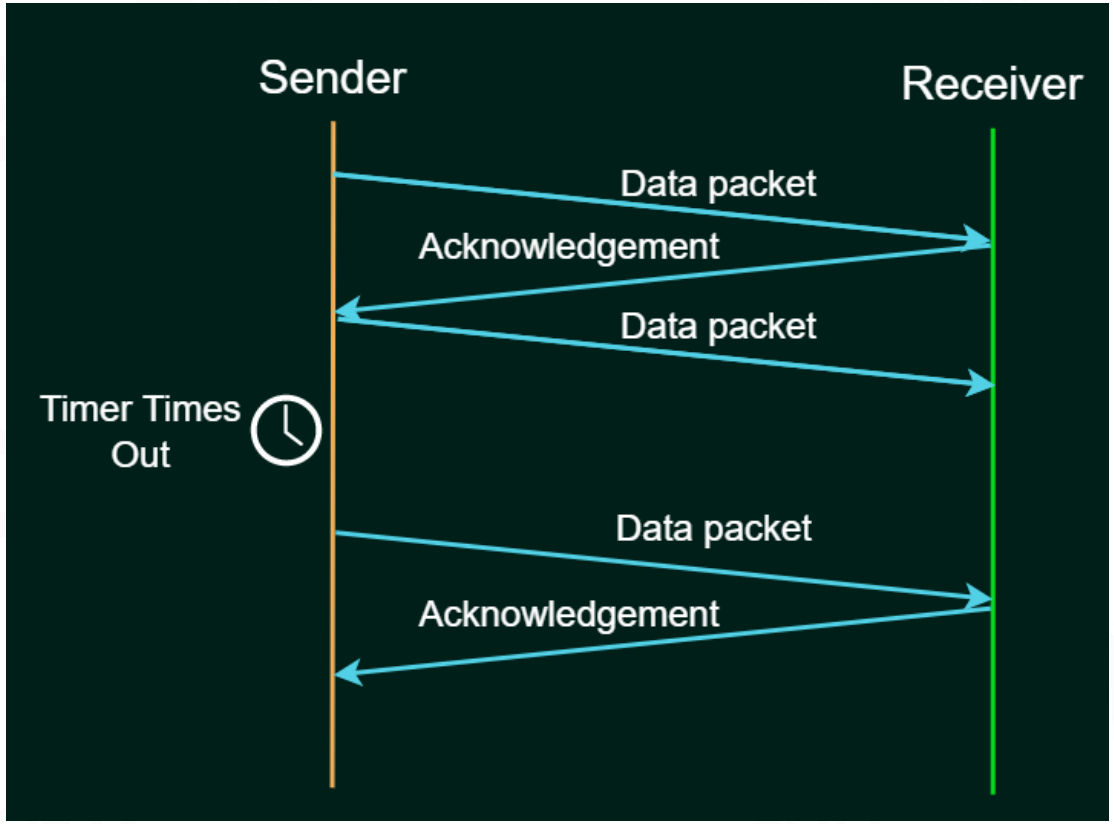
Sliding Window Protocol

- Send mutilple frames at a time, number of frames to be sent is based on **Window size**
- Each frame is numbered by Sequence number
- When the frame at the front of the window is received, the window slides



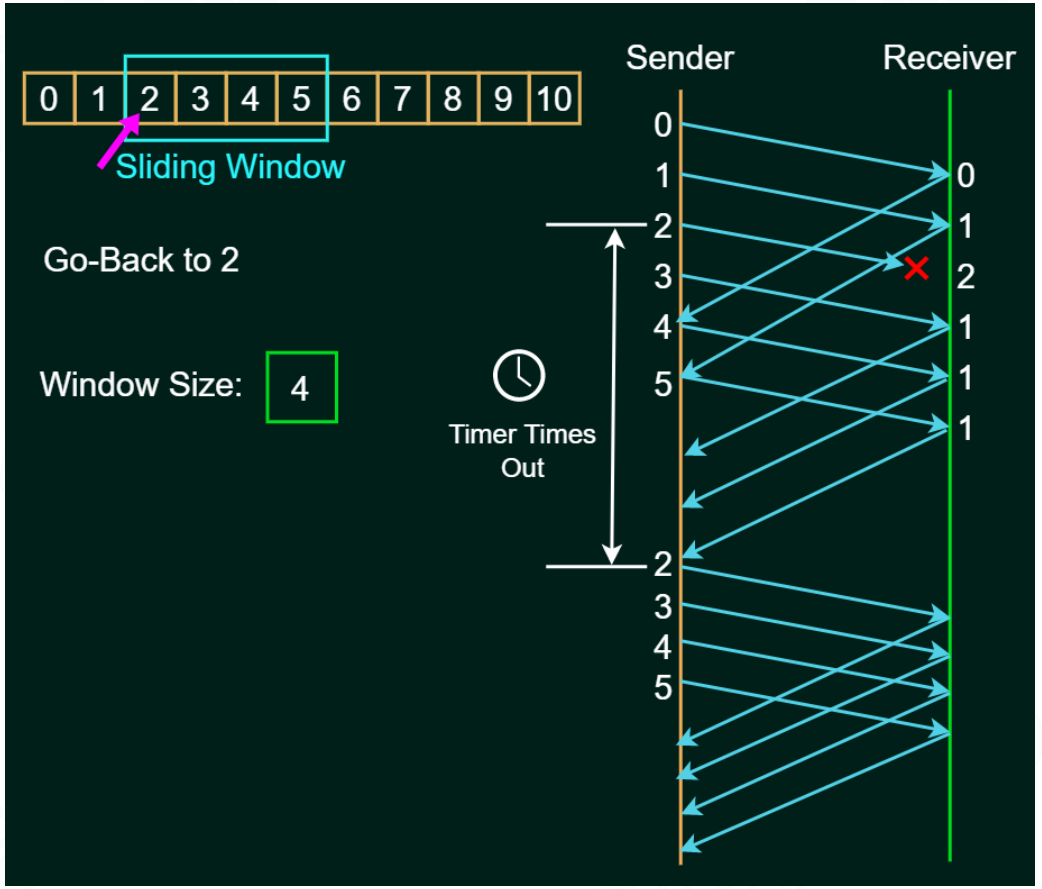
Stop-and-Wait ARQ

- Windows size = 1
- After transmitting one frame, the sender waits for an ACK before transmitting the next frame
- If the ACK does not arrive after a certain time, the sender times out and retransmits the original frame
- **Poor utilization of bandwidth, poor performance**



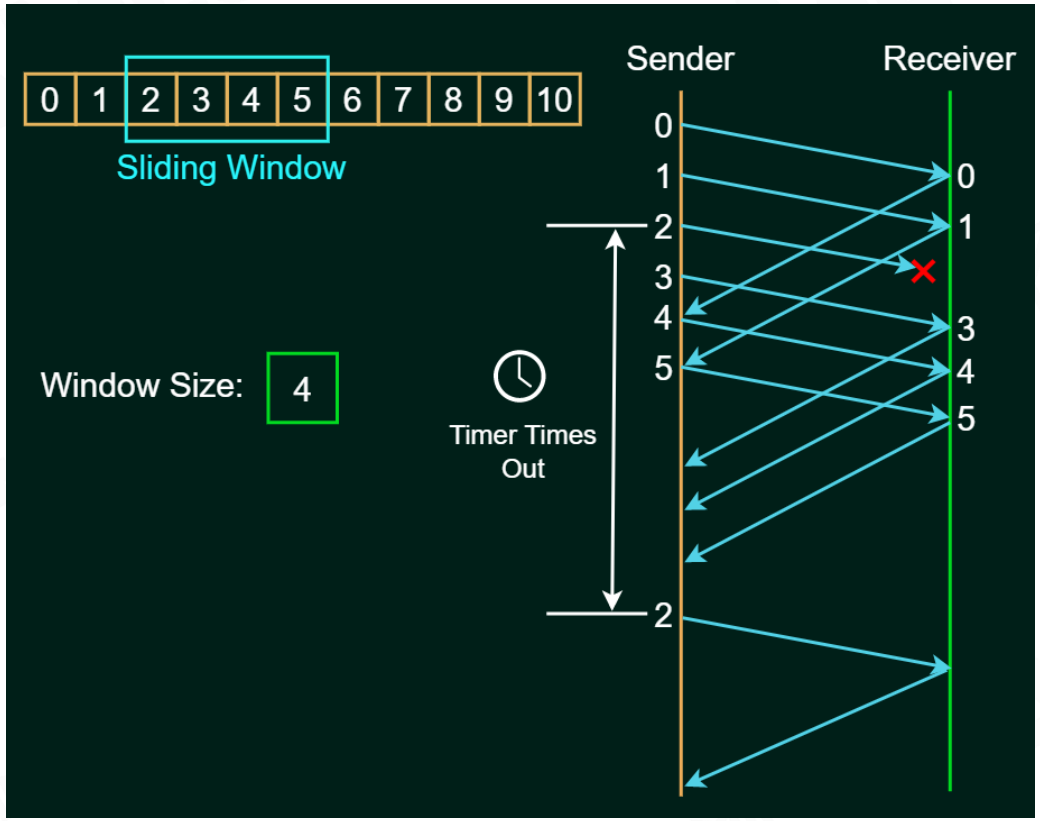
Go-Back-N ARQ

- N is Sender's Windows Size
- The Receiver only sends cumulative ACK
- If an ACK is not received within an agreed-upon time period, all frames in the current window are transmitted



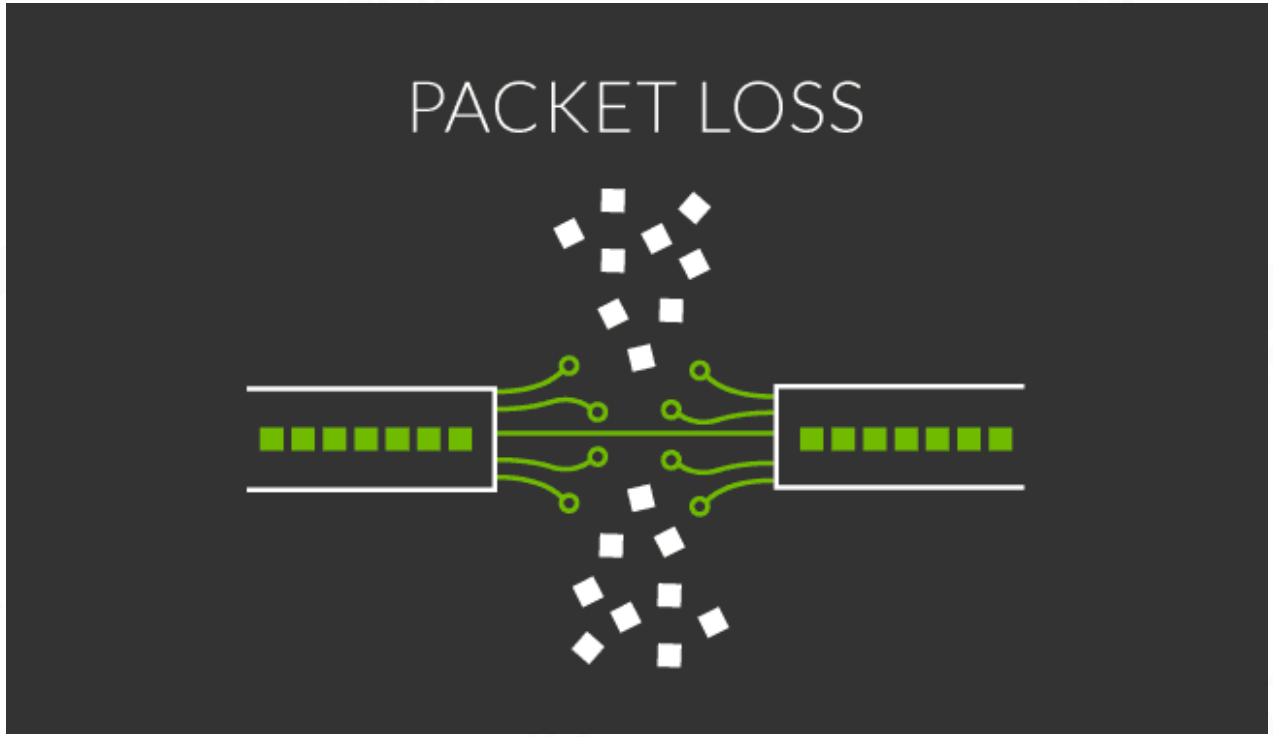
Selective Repeat ARQ

- In Selective Repeat ARQ, only the damaged or lost frames are retransmitted
- The receiver sends the ack of each frame, and the sender maintains the timeout time of each frame
- When receiver receive damaged packet, it will send a NACK, The sender will send/retransmit frame for which NACK is received



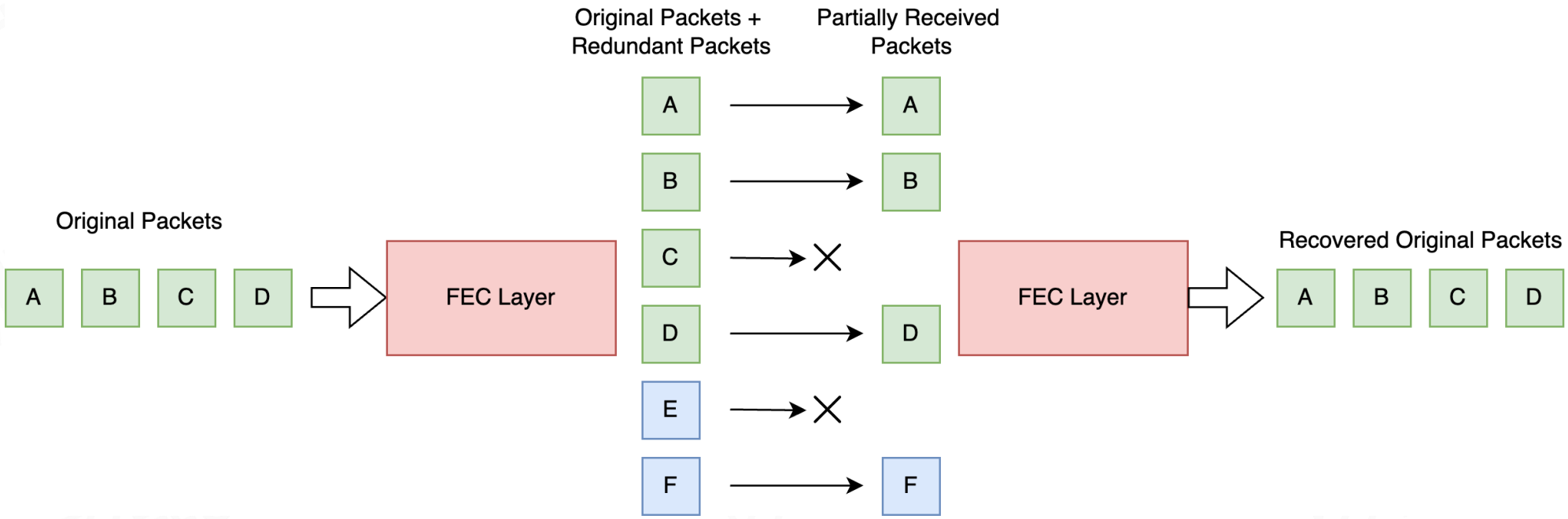
Make UDP Reliable in Packet Loss Scenario

With the increase of packet loss rate and delay, the reliable UDP can not meet the transmission requirements gradually.eg.If packet loss rate increase to 20%, use reliable UDP is still with high delay.



Forward Error Correction (FEC)

- The transmission of enough additional redundant information with the primary data stream to reconstruct lost IP packets up to a certain extent



FEC Algorithms

- Reduce packet loss rate, but cost addtional bandwidth
- The packet loss rate is high, the effect of packet loss compensation is more obvious
- Two FEC algorithms
  - XOR FEC
  - Reed-Solomon Codes

XOR-FEC (1/2)

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Law	Desc
Law of identity	A xor 0 = A
Zeroing law	A xor A = 0
Commutative law	A xor B = B xor A
Associative law	A xor(B xor C) = (A xor B) xor C

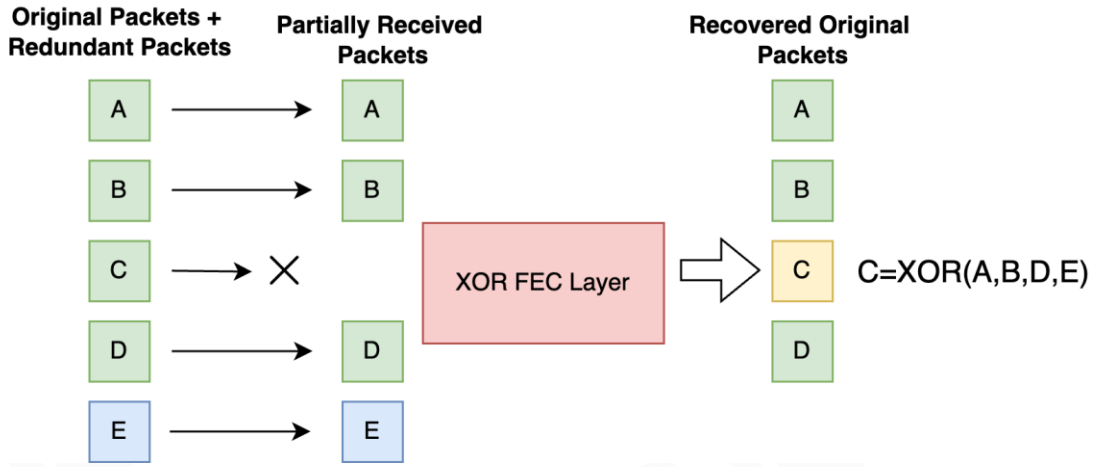
C = A xor B

A = A xor (B xor B) = (A xor B) xor B = C xor B

B = (A xor A) xor B = A xor C

XOR-FEC (2/2)

- There are four packets A, B, C, D
  - Let E = XOR (A, B, C, D)
  - A = XOR (B, C, D, E)
  - B = XOR (A, C, D, E)
  - C = XOR (A, B, D, E)
  - D = XOR (A, B, C, E)

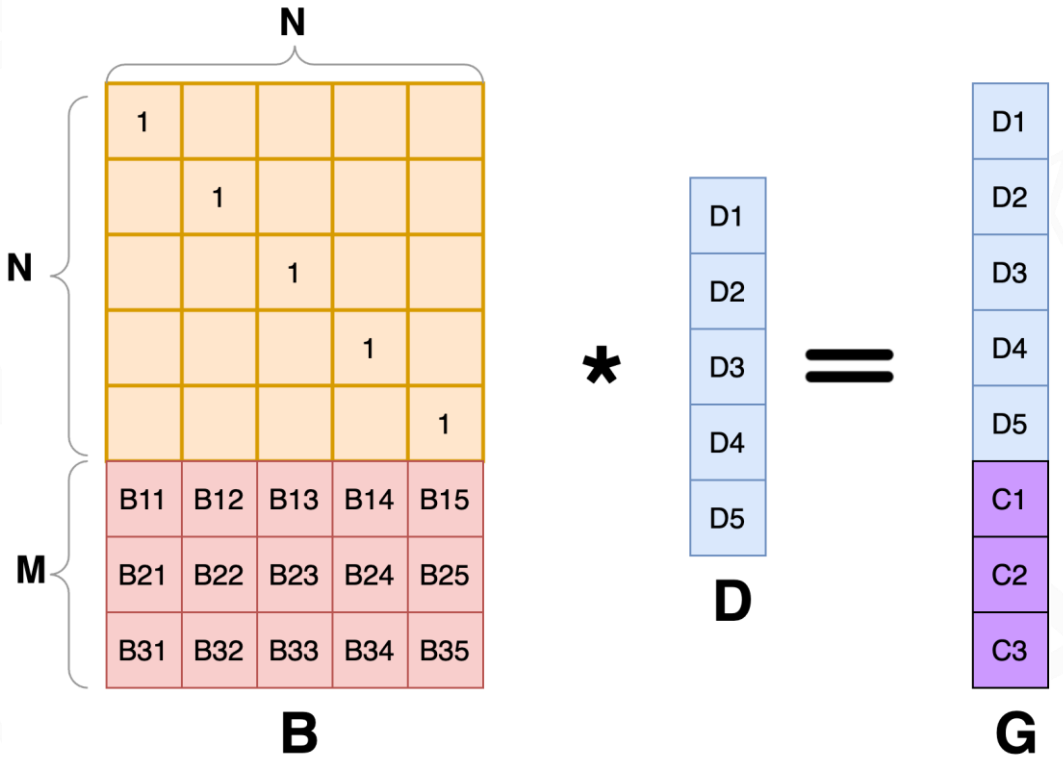


- If any packet is lost, it can be recovered with the other four packets
- Only one packet can be lost in continuous data. If A and B are lost at the same time, the algorithm cannot recover

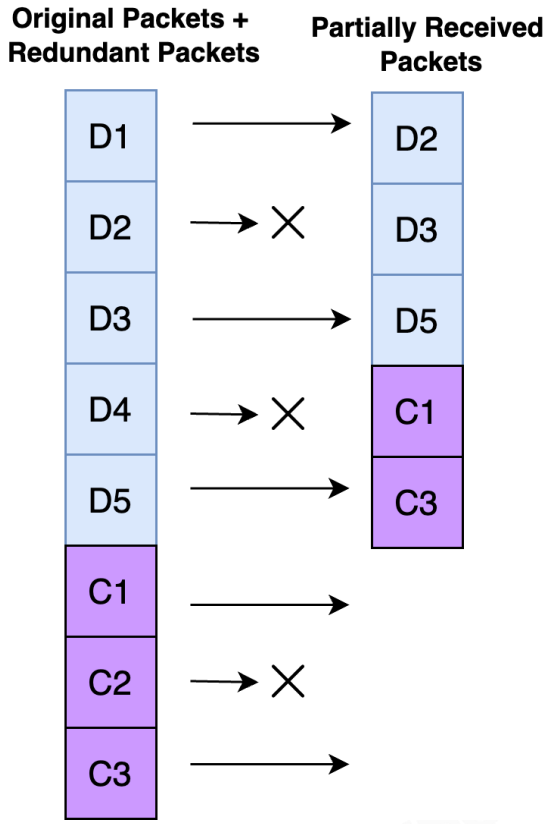
Reed-Solomon Codes (1/3)

There are N valid data, and M FEC data are expected to be generated

- Form N valid data into a unit vector D
- Generate a transformation matrix B: it is composed of a N-order identity matrix and a N * M Vandemode matrix (The matrix composed of any n rows of matrix B is reversible)
- The matrix G obtained by multiplying the matrix B and Vector D contains M redundant FEC data

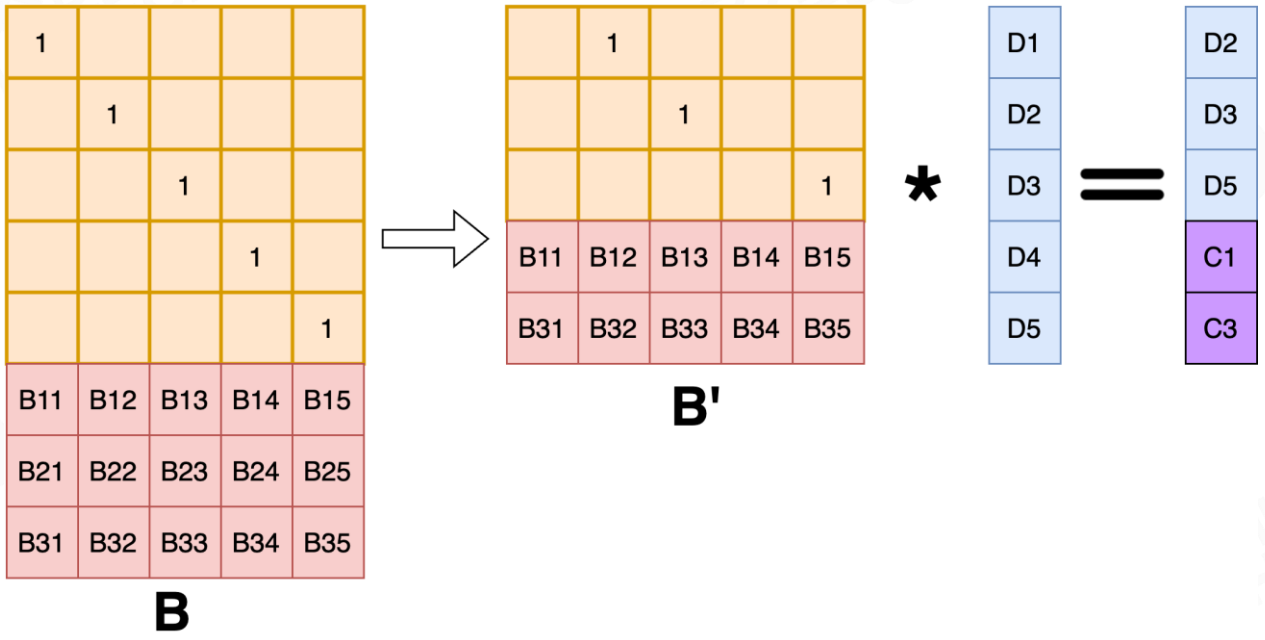


Reed-Solomon Codes (2/3)



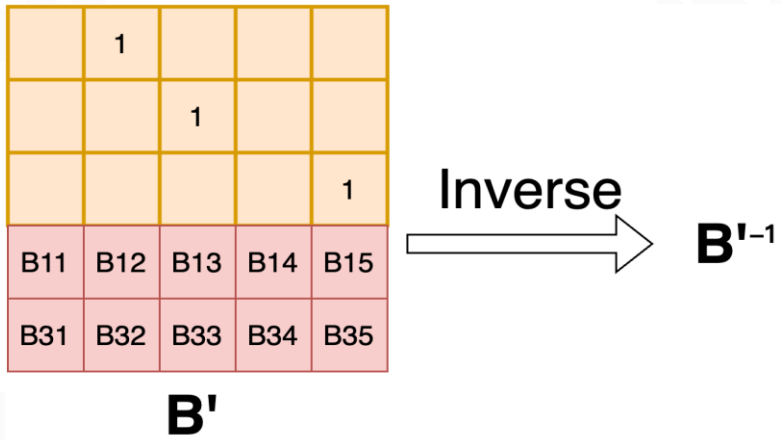
Assume D1, D4, C2 are lost

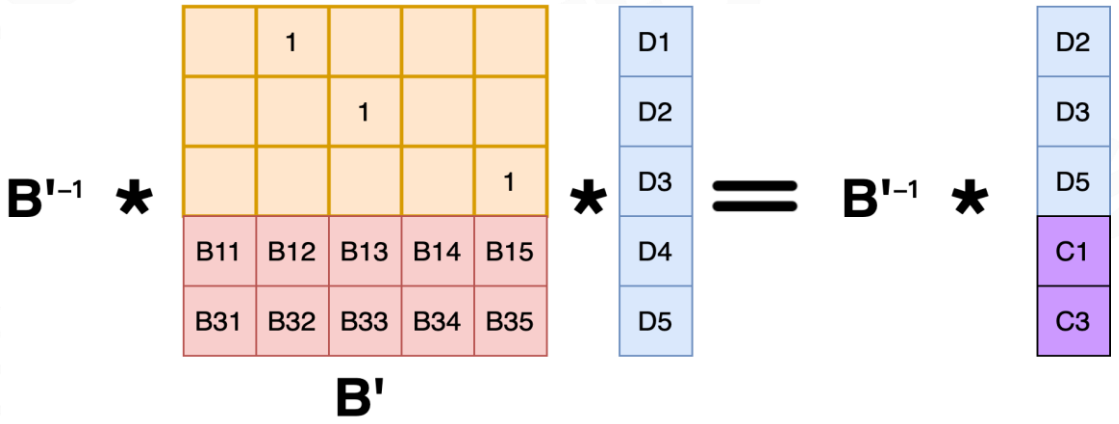
- The B matrix also needs to delete the corresponding M rows to obtain a deformation matrix of B'



Reed-Solomon Codes (3/3)

- Inverse matrix B' get B'^{-1}
- Multiply B'^{-1} on both sides to recover the original data





Customize Your UDP based on ARQ and FEC

Reliability

- Use Selective Repeat ARQ

Real-time

- Smaller RTO growth
- No congestion control
- Fast retransmission mechanism
- No delay ACK

Hybrid ARQ and FEC

- Before ARQ, FEC is used for error correction

Flexibility

- Design protocol for speed
- Support both reliable and unreliable transmission

Clock Synchronization

RTT

Round-Trip Time

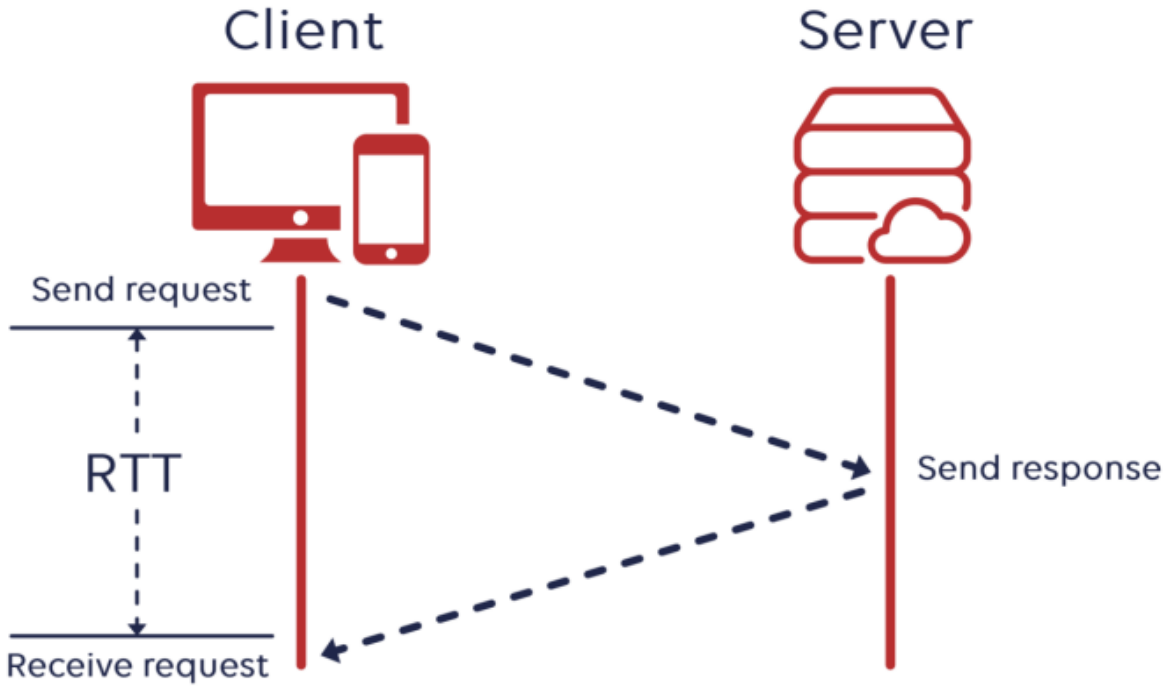
- Send/Recv delay
- Propagation delay
- Response time of the origin server

RTT vs. Ping

- Ping tests are usually performed within a transport protocol that uses ICMP packets
- RTT is measured at the application layer

RTT vs. Latency

- Latency is the time required for a data packet to travel from the sending endpoint to the receiving endpoint (only one trip)



Network Time Protocol (NTP)

Network Time Protocol is an internet protocol used to synchronize with computer clock time sources in a network

- **Reference clock**
  - GPS clock or radio ransmiiting station
  - Amazinglly precise timekeeping devices such as atomic clocks
  - **Not connected to the internet**
  - Send their time through radio or optical fiber

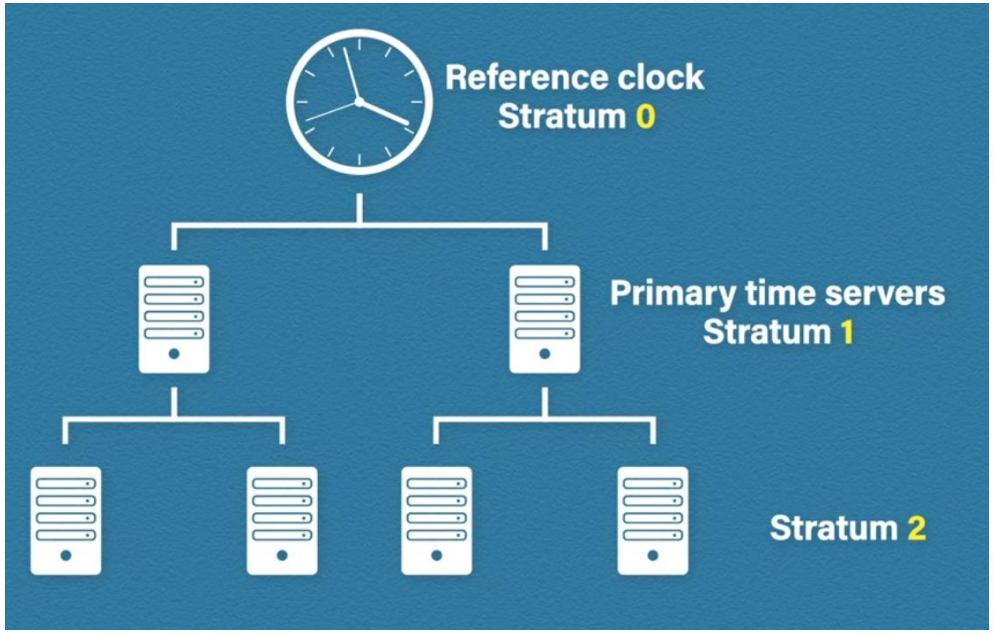
Standard radio transmitting station of Mount Otakadoya (JJY),Fukushima prefecture,Japan.

The atomic clock on board HMS Prince of Wales

Time Server Stratums

Stratum Values

- Degrees of separation from the reference clock
- Reference clock has stratum value of 0
- Servers with stratum value 1 is called primary time servers
- If a device's stratum value is over 15, its time is not trustworthy
- Device will choose server with less stratum value automatically when correcting time

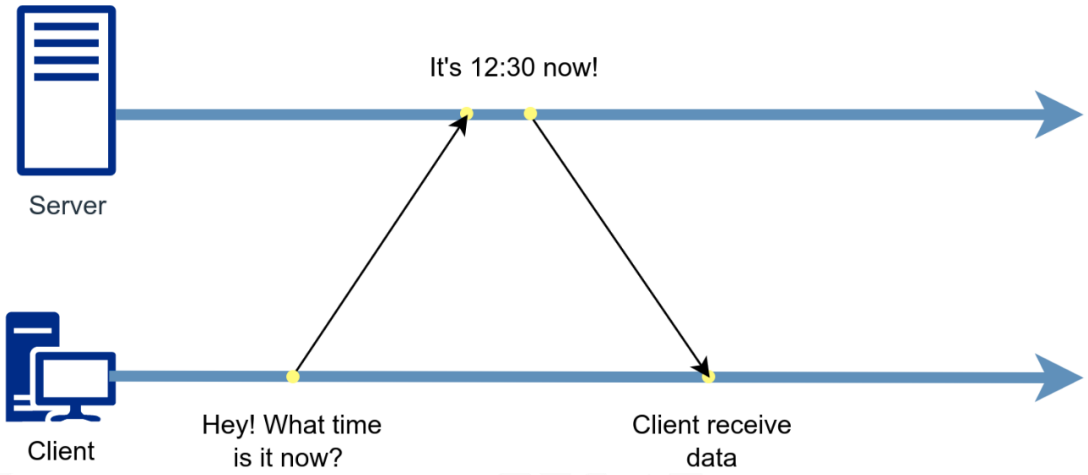


NTP Algorithm (1/3)

Use NTP is quite simple, just like this

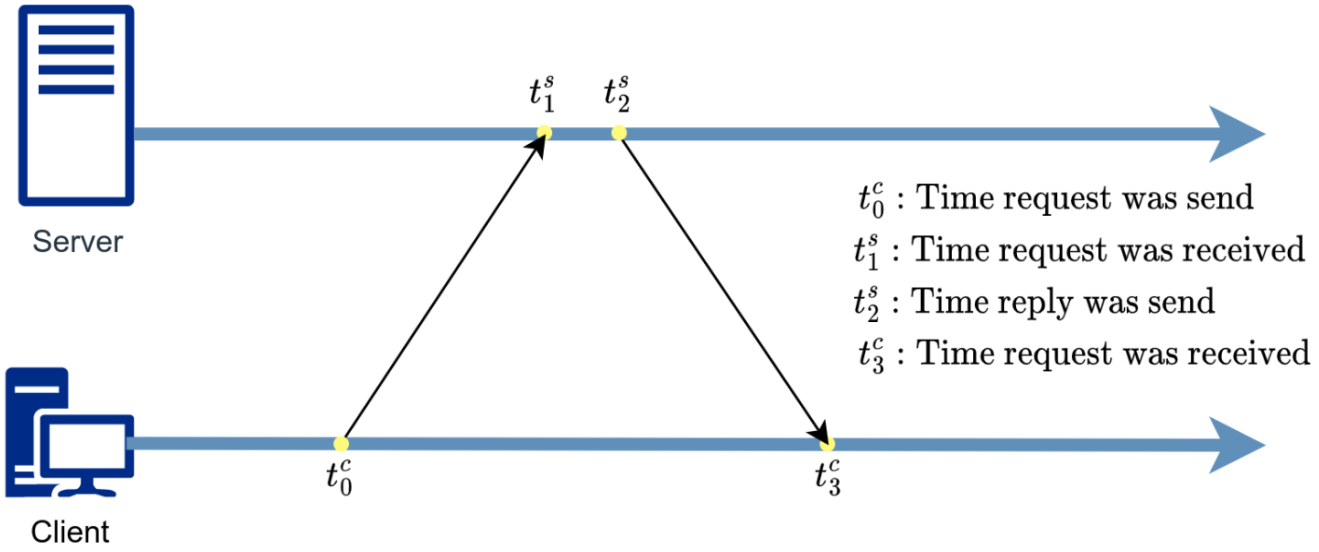
- Client ask time server for time
- Server receives the request and reply
- Client receives the reply

But we have to do something with the Delay!



NTP Algorithm (2/3)

We recordes 4 timestamps as $t_0^c, t_1^s, t_2^s, t_3^c$



$$\text{Round Trip Delay} = (t_3^c - t_0^c) - (t_2^s - t_1^s)$$

$$\text{Offset} = \frac{(t_1^s - t_0^c + t_2^s - t_3^c)}{2}$$

The implicit assumption that the one-way delay is statistically half the round trip delay

Local-clock correction is computed from the offset data by:

- t_3^c + Offset

*The delay and clock-offset samples obtained can be filtered using maximum-likelihood techniques

NTP Algorithm (3/3)

Let's take an example:

It's 17:01:00 on the client

- t_0^c is 17:01:00
- t_1^s is 17:01:32
- t_2^s is 17:01:33
- t_3^c is 17:01:05

It's 17:01:30 on the server

(17:01:30)
(17:01:32)
(17:01:33)
(17:01:35)

Round Trip Delay = (05-00)-(33-32)=4s
Offset = (32-00+33-05)/2 = 30s

$$\text{Round Trip Delay} = (t_3^c - t_0^c) - (t_2^s - t_1^s)$$

$$\text{Offset} = \frac{(t_1^s - t_0^c + t_2^s - t_3^c)}{2}$$

So at t_3^c client's time is corrected to:

- t_3^c + Offset = 17:01:35

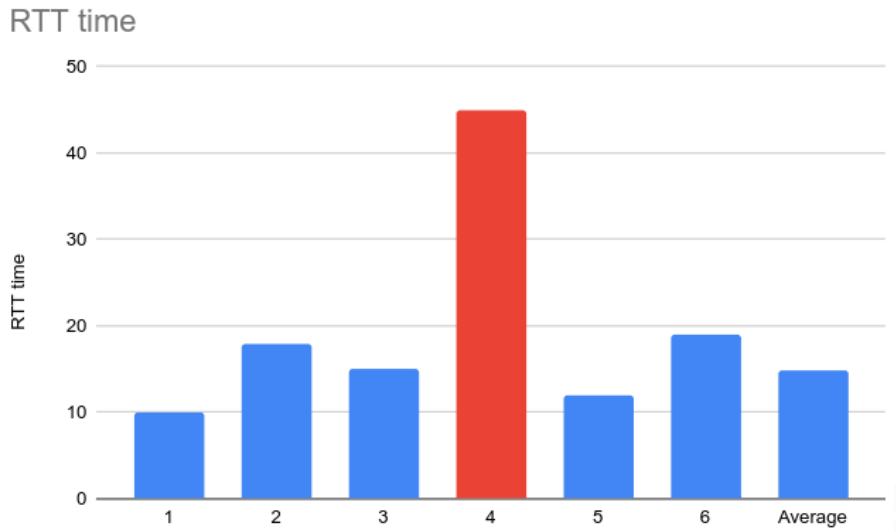
Stream-Based Time Synchronization with Elimination of Higher Order Modes (1/2)

1. Client stamps current local time on a "time request" packet and sends to server
2. Upon receipt by server, server stamps server-time and returns
3. Upon receipt by client, a time delta is calculated by delta = (current Time-sent Time) / 2

So far this algorithm is very like to NTP

Stream-Based Time Synchronization with Elimination of Higher Order Modes (2/2)

4. The first result should immediately be used to update the clock
5. The client repeats Steps 1-3 (NTP-like process), five or more times
6. The results of the packet receipts are accumulated and sorted in ascending order by latency
7. All samples above that are approximately 1.5 times the median are discarded, and the remaining samples are averaged using an arithmetic mean

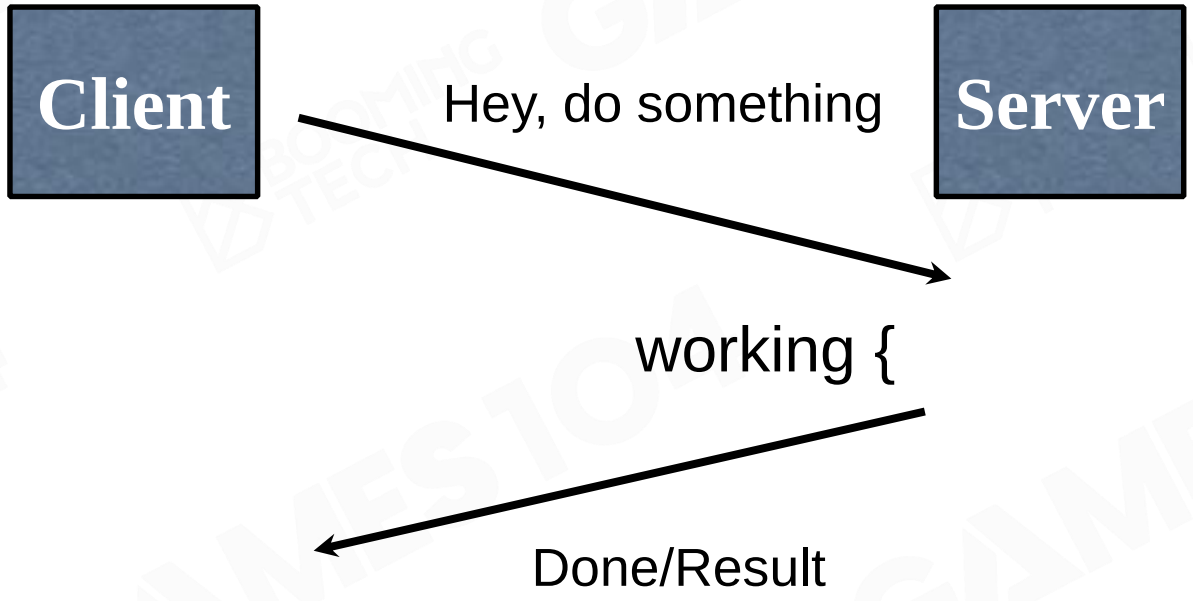


Remote Procedure Call (RPC)

Socket Programming: Still not Great

- Lots for the programmer to deal with every time
  - How to separate different requests on the same connection?
  - How to write bytes to the network/read bytes from the network?
  - What if Host A's process is written in Go and Host B's process is in C++?
  - What to do with those bytes?
- Still pretty painful… have to worry a lot about the network
  - Have you received the message?

Communication with Messages



Communication Way (2/3)

- Initially, people "hand-coded" messages to send requests and responses
  - Message is a stream of bytes – "op codes" and operands
- Lots of drawbacks
  - Need to worry about message format
  - Have to pack and unpack data from messages
  - Servers have to decode messages and dispatch them to handlers
  - Messages are often asynchronous
    - After sending one, what do you do until the response comes back?
  - Messages aren't a natural programming model

```
struct foomsg {
  uint32_t len;
};

void send_foo(char *contents) {
    int msglen = sizeof(struct foomsg) + strlen(contents);
    char buf = malloc(msglen);
    struct foomsg *fm = (struct foomsg *)buf;
    fm->len = htonl(strlen(contents));
    memcpy(buf + sizeof(struct foomsg),
                contents,
                strlen(contents));
    write(outsock, buf, msglen);
}
```

Writing it by hand...

More Challenges on Logic Communication

- For a remote procedure call, a remote machine may:
  - Run process written in a different language
  - Represent data types using different sizes
  - Use a different byte ordering (endianness)
  - Represent floating point numbers differently
  - Have different data alignment requirements

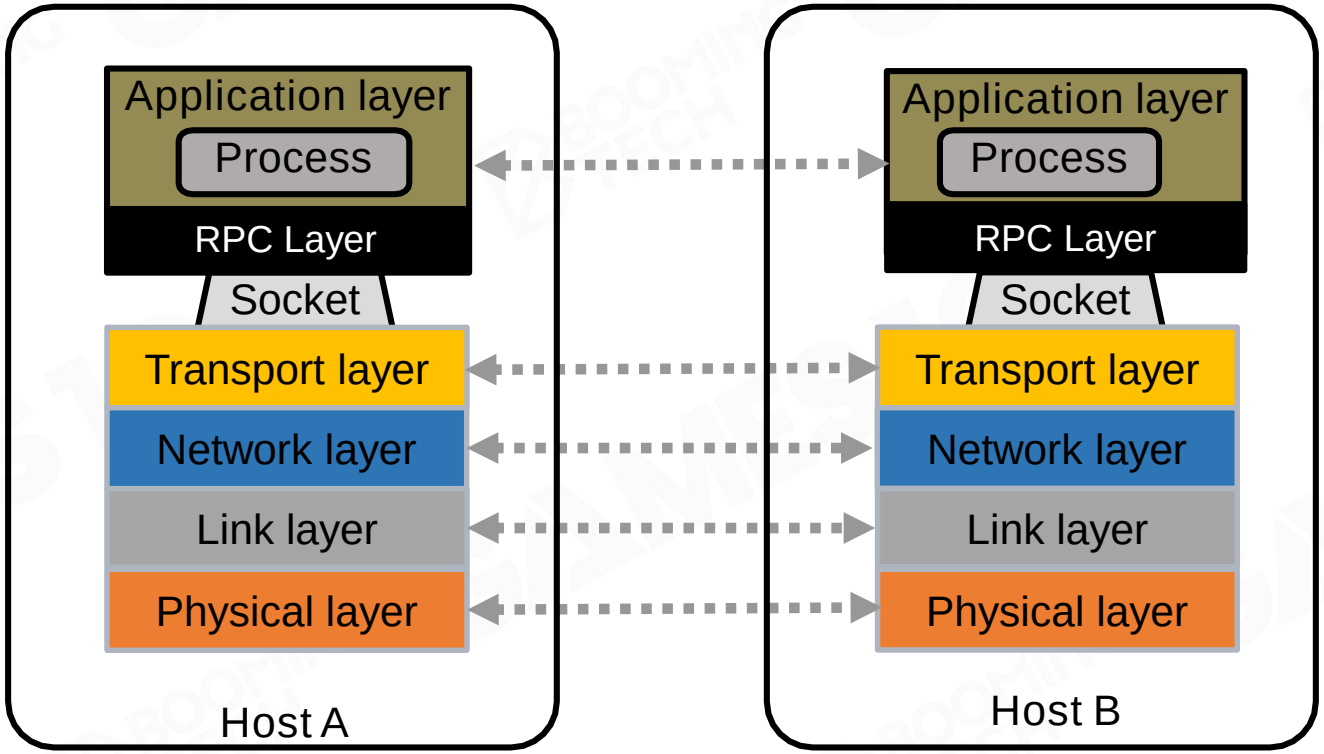
    e.g., 4-byte type begins only on 4-byte memory boundary

Remote Procedure Call (RPC)

- RPC is a request–response protocol. An RPC is initiated by the *client*, which sends a request message to a known remote *server* to execute a specified procedure with supplied parameters

- Goals
  - Ease of programming
  - Hide complexity
  - Familiar model for programmers

  (just make a function call)



RPC example

- Go language

```go
//Client
package main

import (
    "fmt"
    "log"
    "net/rpc"
)

func main() {
    client, err := rpc.Dial("tcp", "127.0.0.1:8000")
    if err != nil {
        log.Fatal("dial", err)
    }
    var response string
    err = client.Call("HelloWorldService.SayHello", "World", &response)
    if err != nil {
        log.Fatal("caller", err)
    }
    fmt.Println(response)
}
```

```go
//Server
package main

import (
    "net"
    "net/rpc"
)

type HelloWorldService struct {
}

func (s *HelloWorldService) SayHello(request string, response *string) error {
    *response = "Hello " + request
    return nil
}


func main() {
    _ = rpc.RegisterName("HelloWorldService", &HelloWorldService{})
    listener, err := net.Listen("tcp", ":8000")
    if err != nil {
        panic("Monitor port failed!")
    }
    conn, err := listener.Accept()
    if err != nil {
        panic("Establish a connection failure!")
    }
    rpc.ServeConn(conn)
}
```

- Output

Hello World

Why RPC?

- Goal: Easy-to-program network communication that makes client-server communication transparent
  - Retains the "feel" of writing centralized code
    - Programmers needn't think about the network
    - Make communication appear like a local procedure call
  - Don't need to worry about serialization/deserialization for network
  - Don't need to worry about complexities of network

Interface Definition Language

- A server defines the service interface using an interface definition language (IDL)

  The IDL specifies the names, parameters, and types for all client-callable server procedures

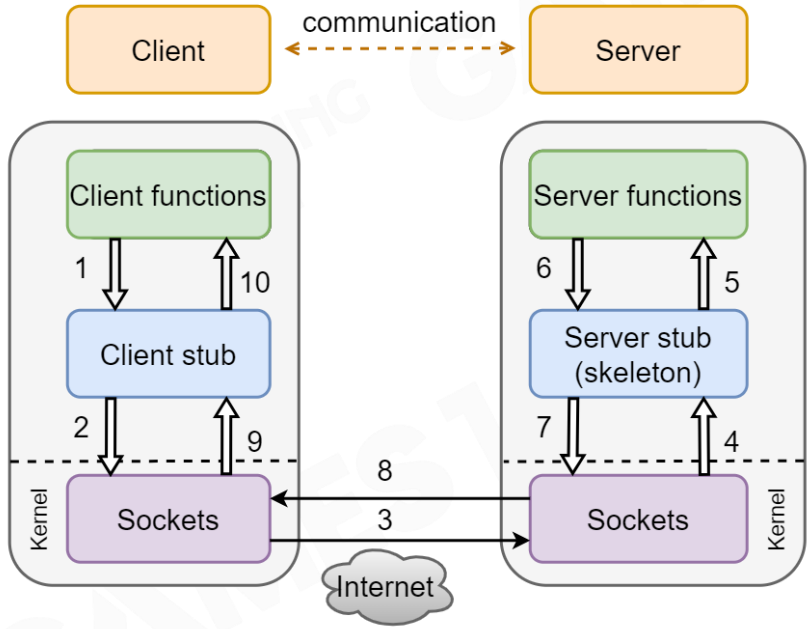
  - example: ASN.1 in the OSI reference model
  - example: Protobuf (Google's data interchange format)
  - ...

```
//polyline.proto
syntax = "proto2";

message Point {
  required int32 x = 1;
  required int32 y = 2;
  optional string label = 3;
}

message Line {
  required Point start = 1;
  required Point end = 2;
  optional string label = 3;
}

message Polyline {
  repeated Point point = 1;
  optional string label = 2;
}
```

Google ProtoBuf

RPC Stubs

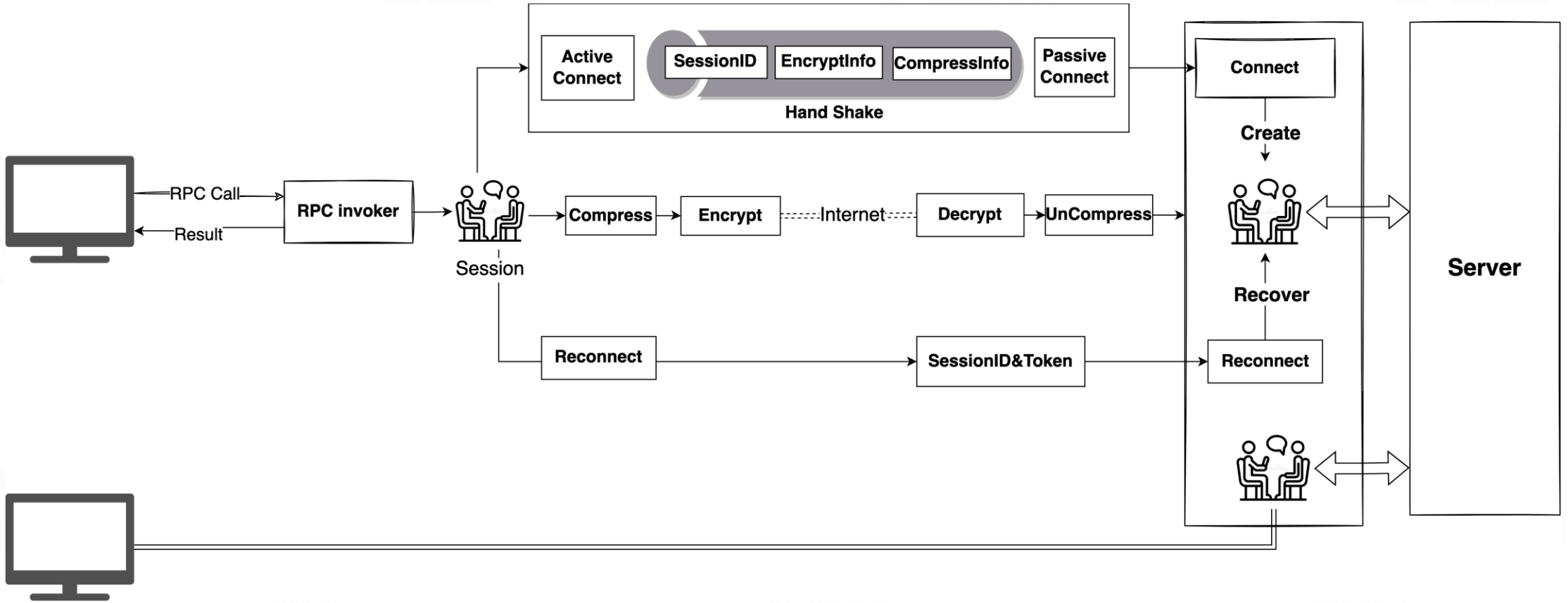
- A client-side stub is a procedure that looks to the client as if it were a callable server procedure
  - The client program thinks it's invoking the server but it's calling into the client-side stub
- A server-side stub looks like a caller to the server
  - The server program thinks it's called by the client but it's really called by the server-side stub
- The stubs send messages to each other to make the RPC happen transparently



Stub Compiler

- A "stub compiler" reads the IDL declarations and produces two stub procedures for each server procedure
  - The server programmer implements the service's procedures and links them with the server-side stubs
  - The client programmer implements the client program and links it with the client-side stubs
  - The stubs manage all of the details of remote communication between client and server

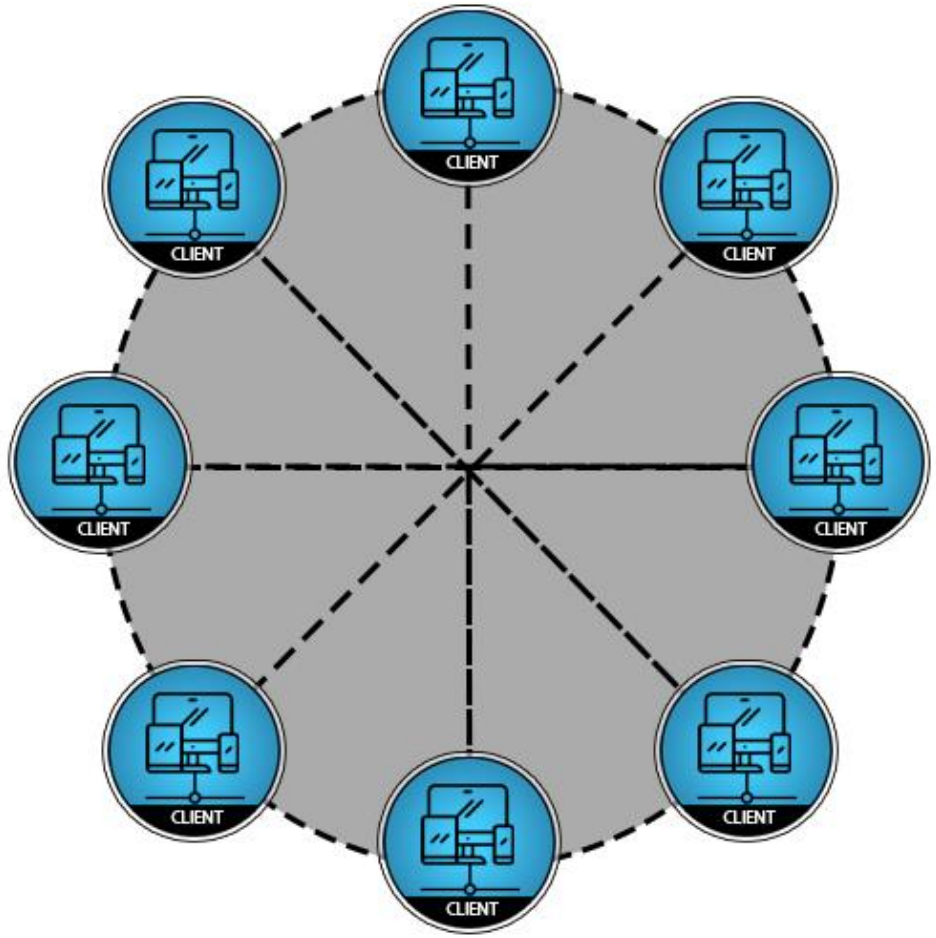
Real RPC Package Journey



Network Topology

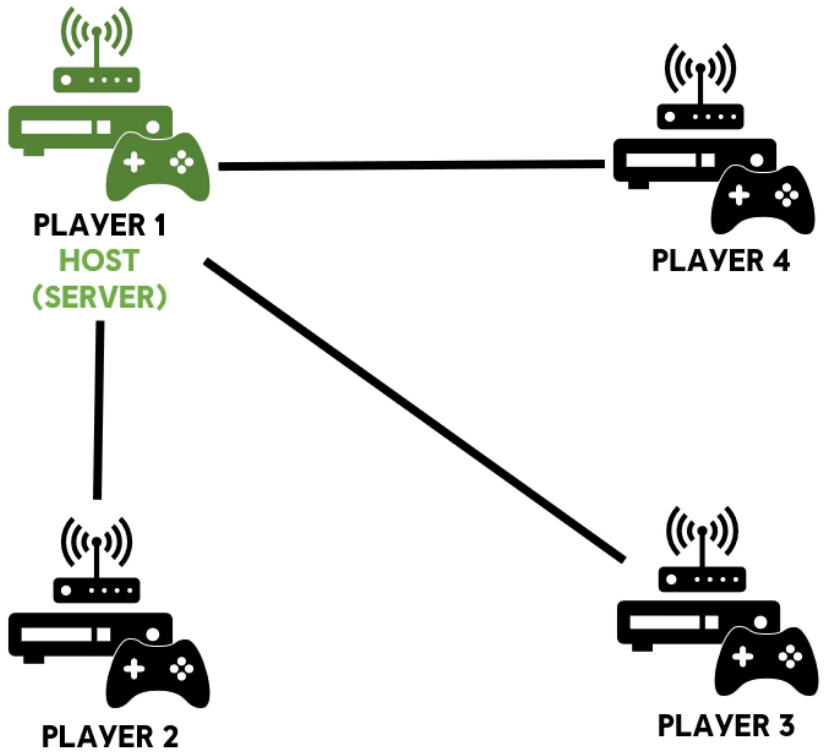
Original Peer-to-Peer (P2P)

- Each client broadcasts game event to the all others
- Robustness
- Cheating is much easier
- Synchronization is required among all nodes to maintain the consistency of the distributed game state



P2P with Host Server

- A player can act as "server", known as host
- If host disconnected, the game may end
- The host need to handle game actor that can not be controled by players, such as bot

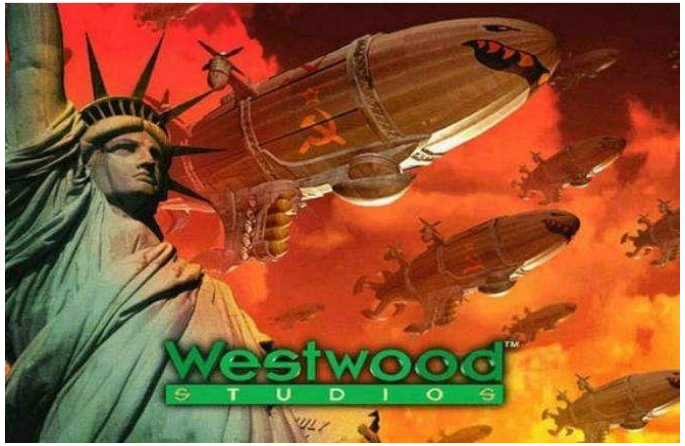


P2P Games

- No rely on server
- Used in Lan commonly
- The “Host” is basically in control of the sessions
- A limited number of players at once

Counter-Strike

Warcraft III

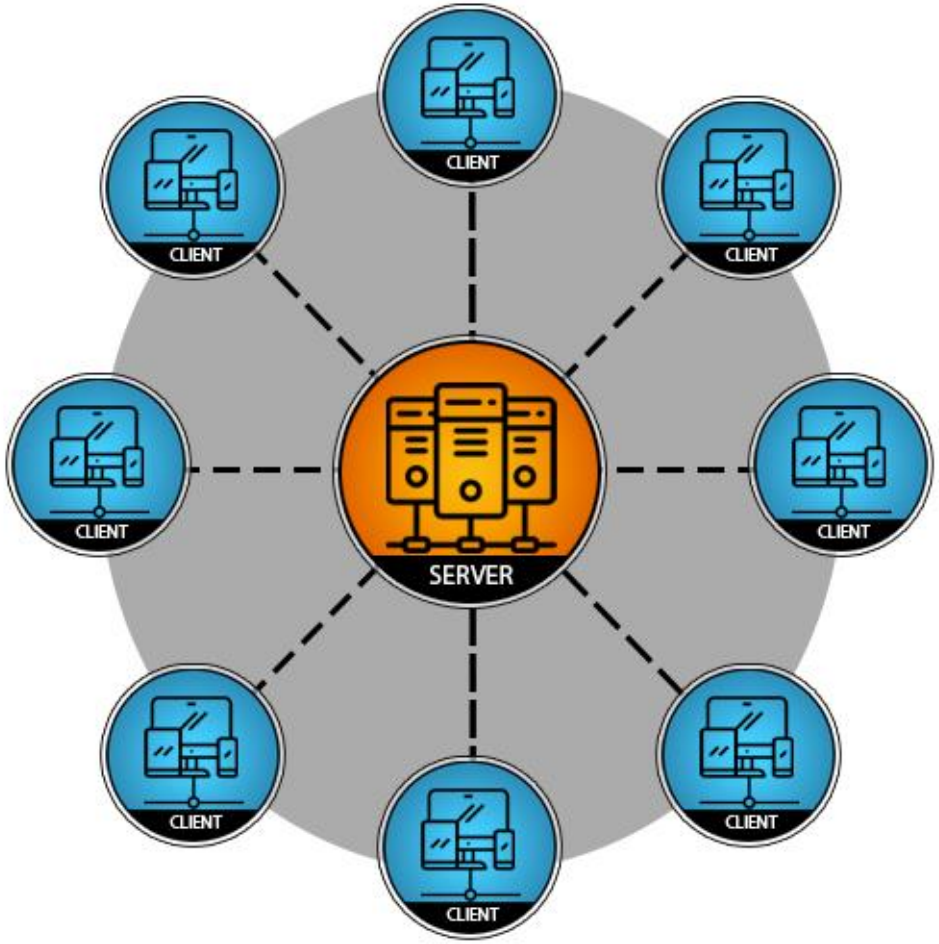


Red Alert

Left 4 Dead

Dedicated Server

- Authority
- Simulate game world
- Dispatch data to players
- High performance requirements

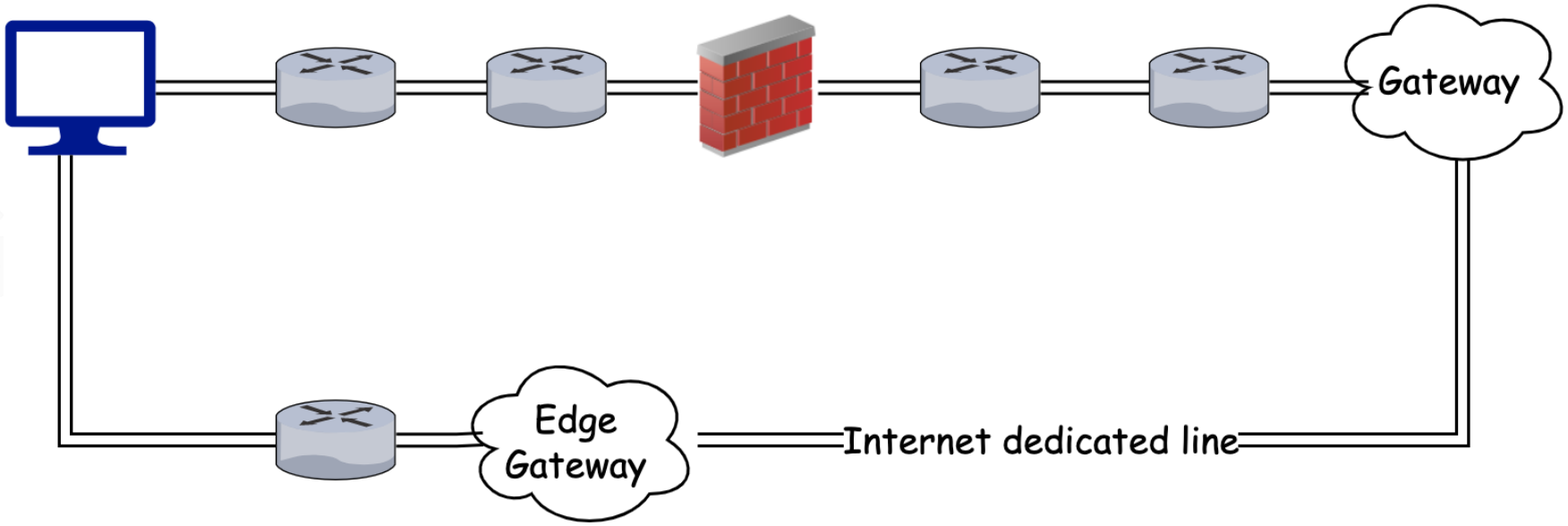


P2P vs Dedicated Server

	P2P	Dedicated Server
Pros	1. Robustness 2. Removes the "server issues" problem in multiplayer sessions. 3. No extra cost on server	1. Easy to maintain as well as cheating avoidance 2. Can handle massive game world 3. Responsiveness of the game is not relay on the network conditions of **each individual** client
Cons	1. Cheating is much easier 2. Every player needs a decent network connection for game to function properly 3. Can only handle a **limited number** of players	1. High cost on server 2. Much more work on server side program 3. Single point of failure

When RTT is too high

- When players are in different countries, far away, or when the network environment is complex
- Use dedicated line and edge gateway to reduce latency

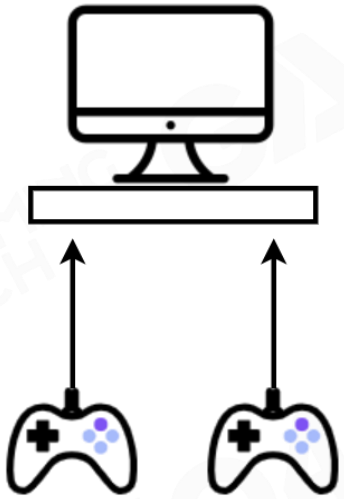


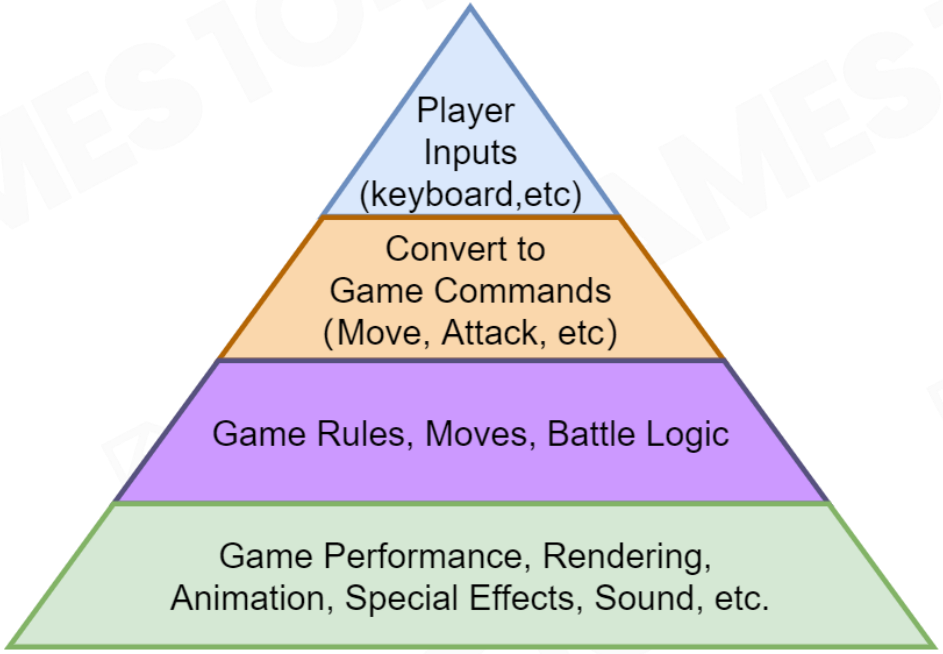
Game Synchronization

Single-Player Gaming

Game Tick

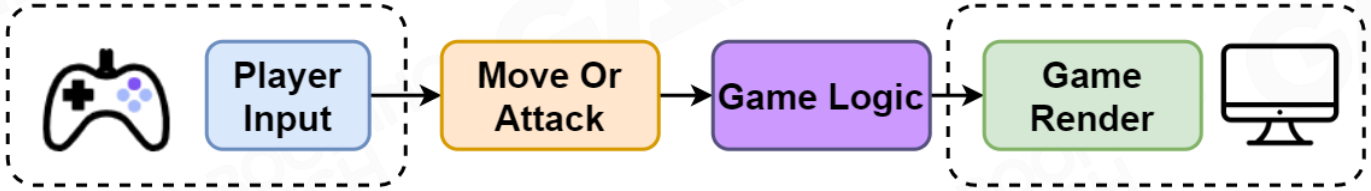
- Player inputs
- Convert to game commands
- Game logic
- Game render





For Player

- Player inputs
- Consistency in each other



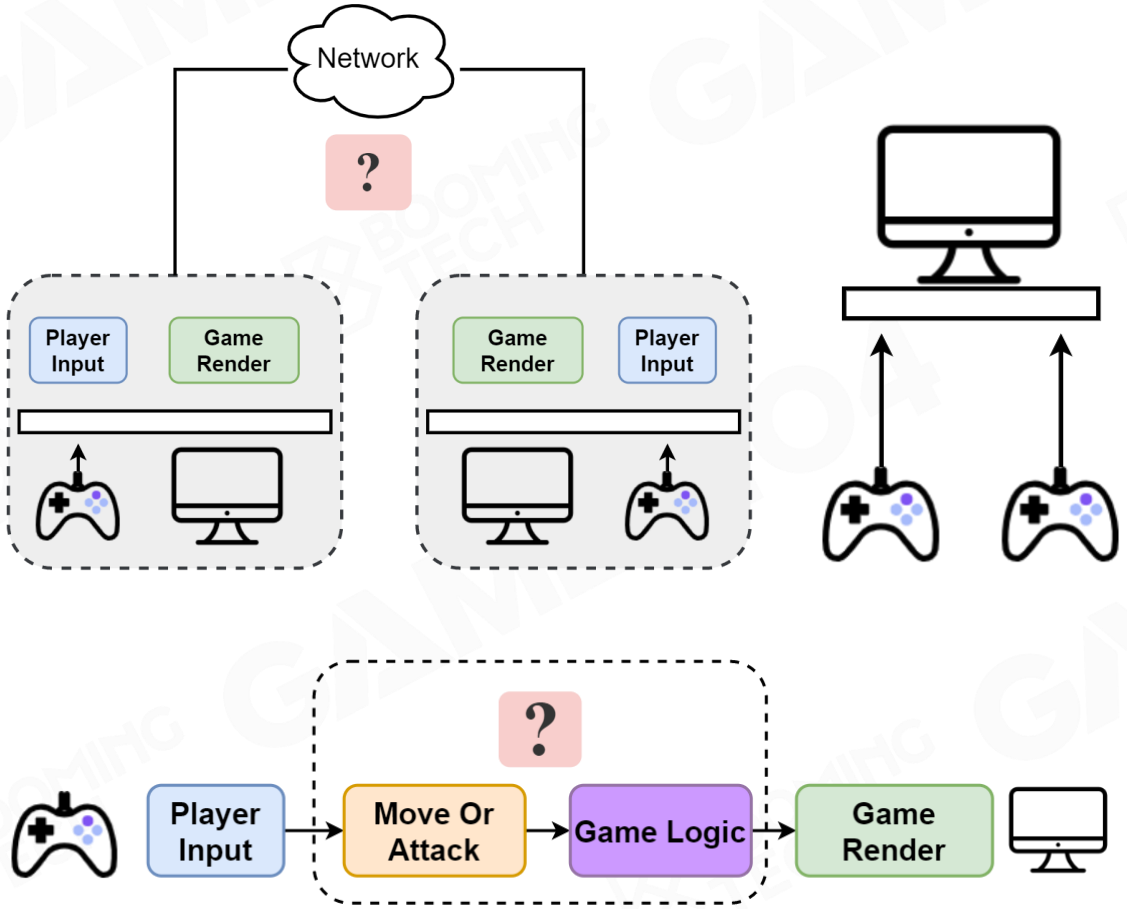
Online Gaming

For Player

- Player inputs
- Consistency in each other

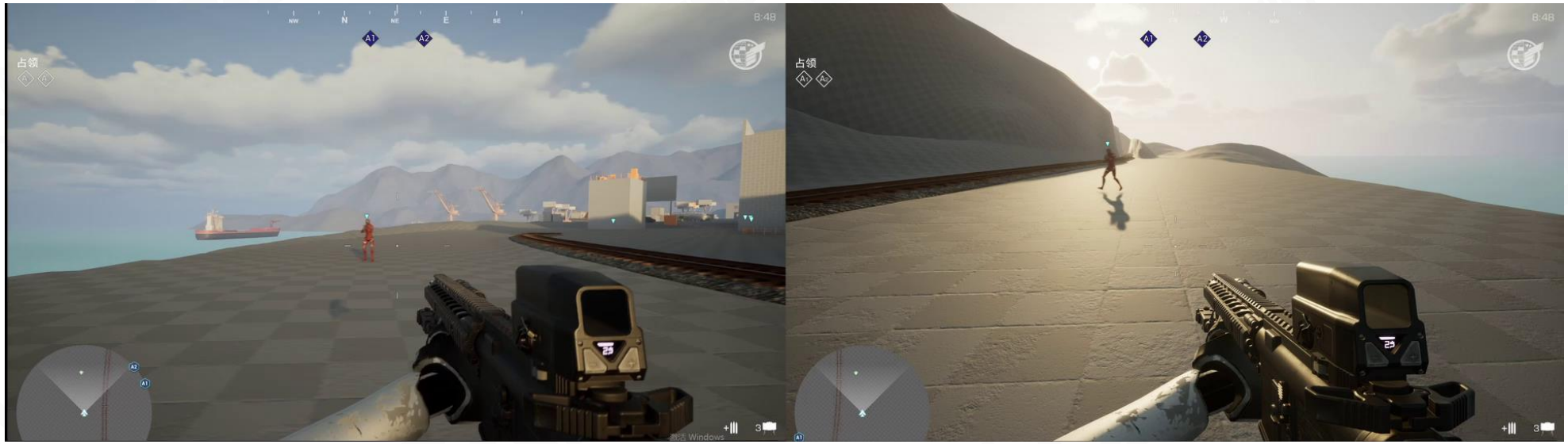
How to play together at different terminals?

- Game commands
- Game Logic



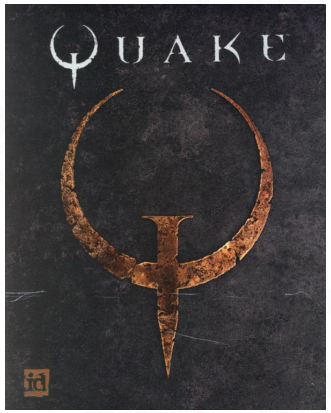
Game Synchronization

To answer the demand for responsive strategies, the synchronization rule is designed to solve the delay and consistency of all destination.



Synchronization Methods

Snapshot



Quake

Lockstep

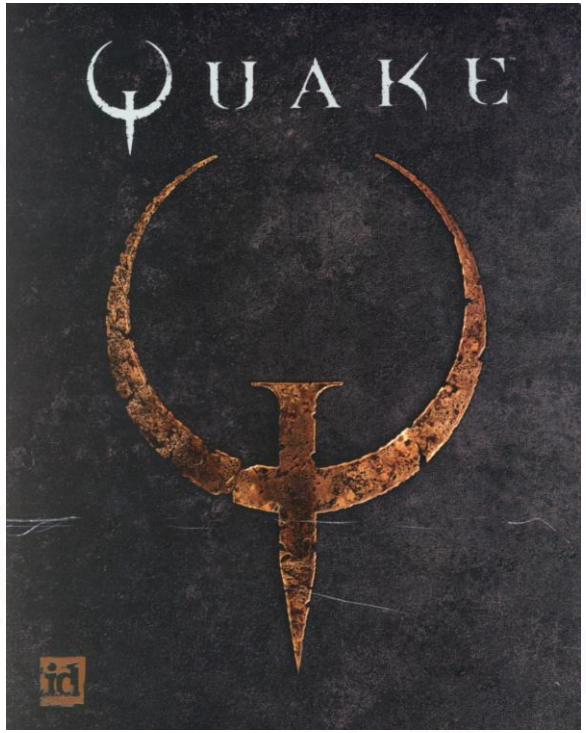
Honor of Kings

State Synchornization

Counter Strike

Snapshot Synchronization

Snapshot Synchronization



Snapshot Synchronization

- Client sends inputs to server

- Server simulates the game world
- Generates whole game state snapshots
- Sends them down to clients

- Client updates the display according to the snapshot

Snapshot Synchronization - Jitter and Hitches

- Server tick rate is limited
  - Performance
  - Bandwidth

60 pps

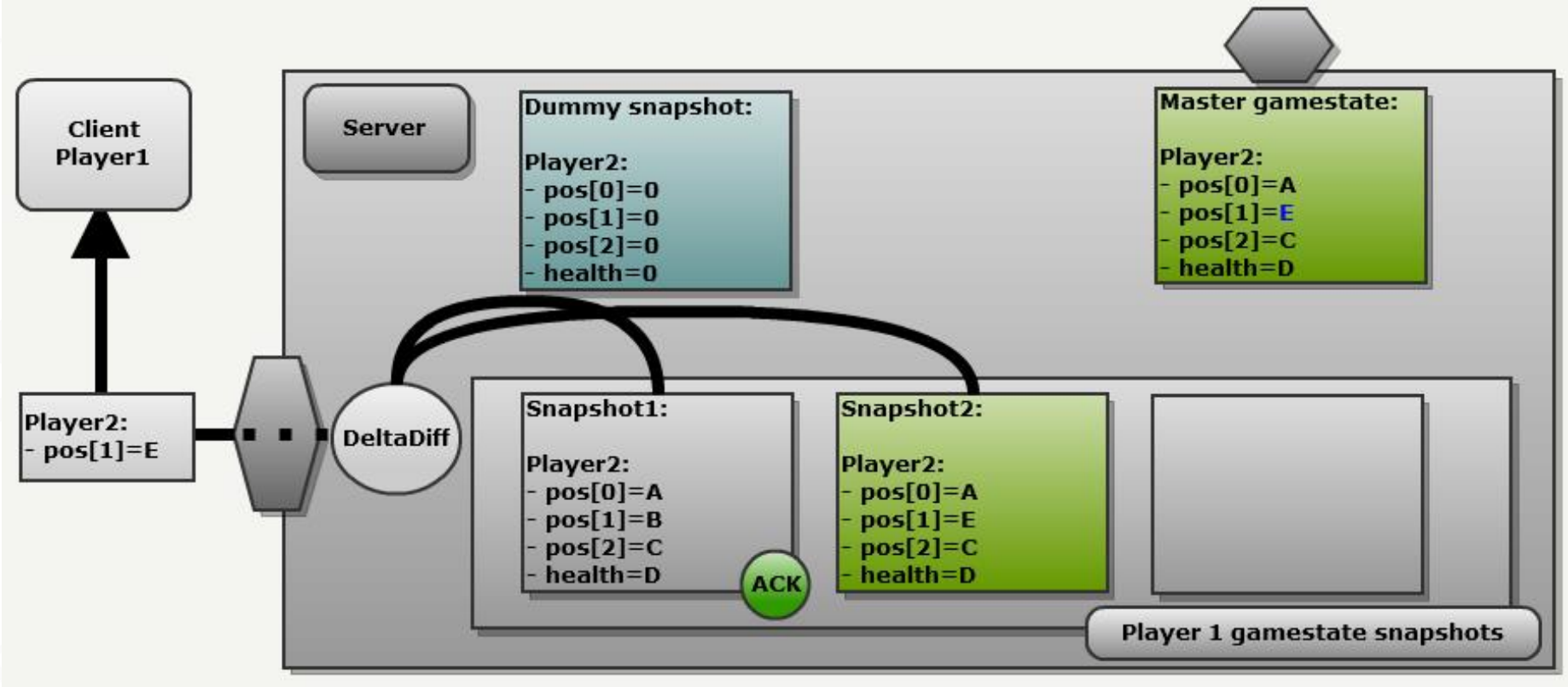
10 pps

Snapshot Interpolation

- Not rendering immediately after snapshot recevied
- Keep an interpolation buffer
- Interpolation between the two delayed snapshots

Delta Compression

- Only sync snapshot delta to client
- Example Quake3



Delta Compression

Snapshot Synchorization

- 60HZ
- Max Bandwidth 4.5 mbps

Bandwidth: 450 kbps

Delta Compression

- 60HZ
- Max Bandwidth 676 kbps

Bandwidth: 6 kbps

Synchronizing Snapshot

- Client performance is wasted
- High server pressure
- High data volume and high bandwidth requirements
- As games get more complex, snapshots get bigger

Lockstep Synchronization

Lockstep Origin (1/2)

Lockstep synchronization, used in military simulation, is by far the simplest techique to ensure consistency

- Same Result
  - Same time
  - Same action

Lockstep in Military

Lockstep Origin (2/2)

No member is allowed to advance its simulation clock until all others members have acknowledged that they are done.

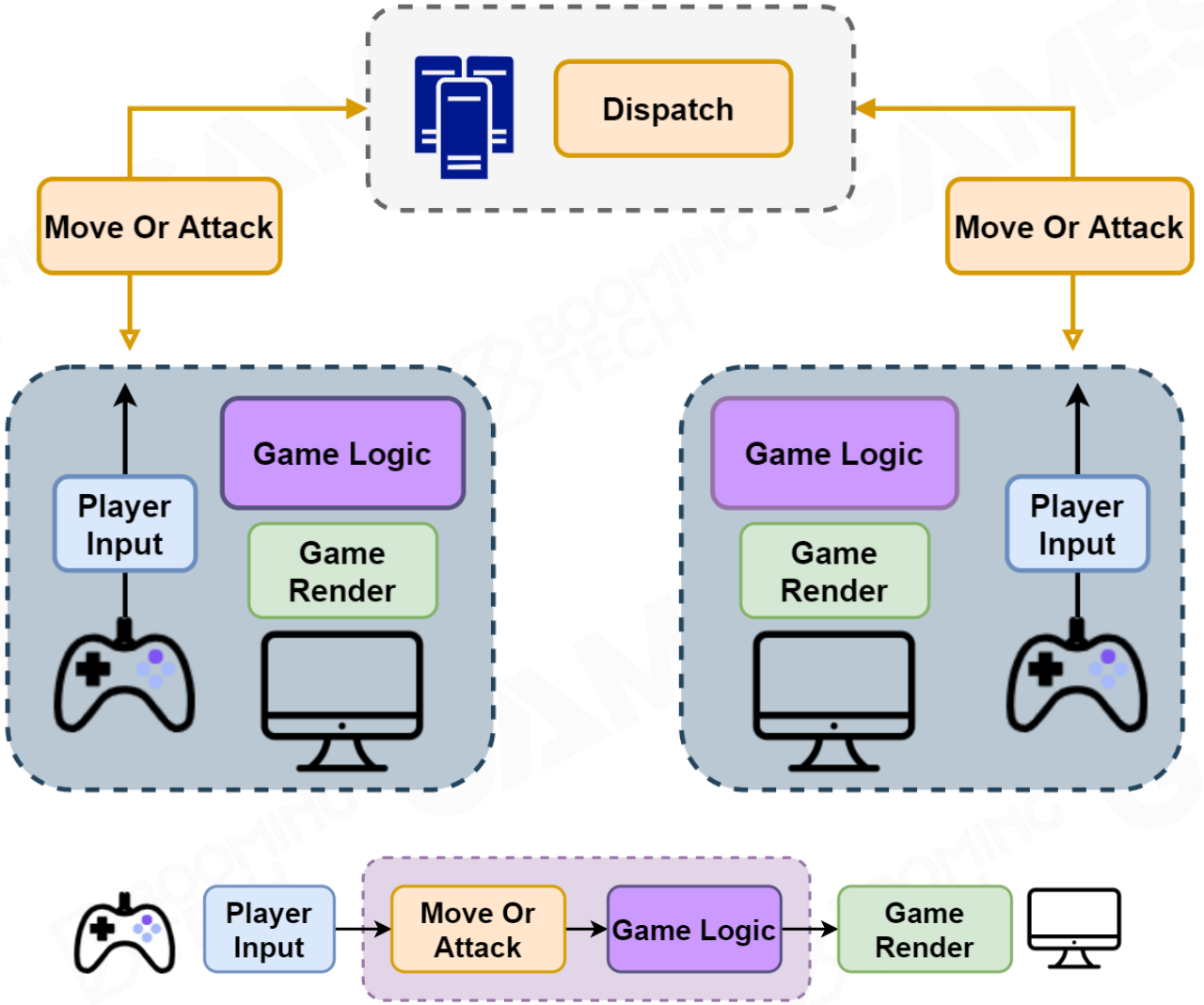
Chess

Five Card Stud

In particular, it is clear that a totally ordered delivery is a sufficient condition to ensure game state consistency across different nodes, as it guarantees that all generated events are reliably delivered according to the same unique order.

Lockstep in Online Game

- Lockstep Principle



First Game Used Lockstep

- The network synchronization method of DOOM (1994) was pointed out in a 2006 paper
- Lockstep is not mentioned in the paper, but it is now generally accepted that Doom (1994) was the first multiplayer FPS online game to use this type of synchronization
- It uses P2P architecture

Doom 1994

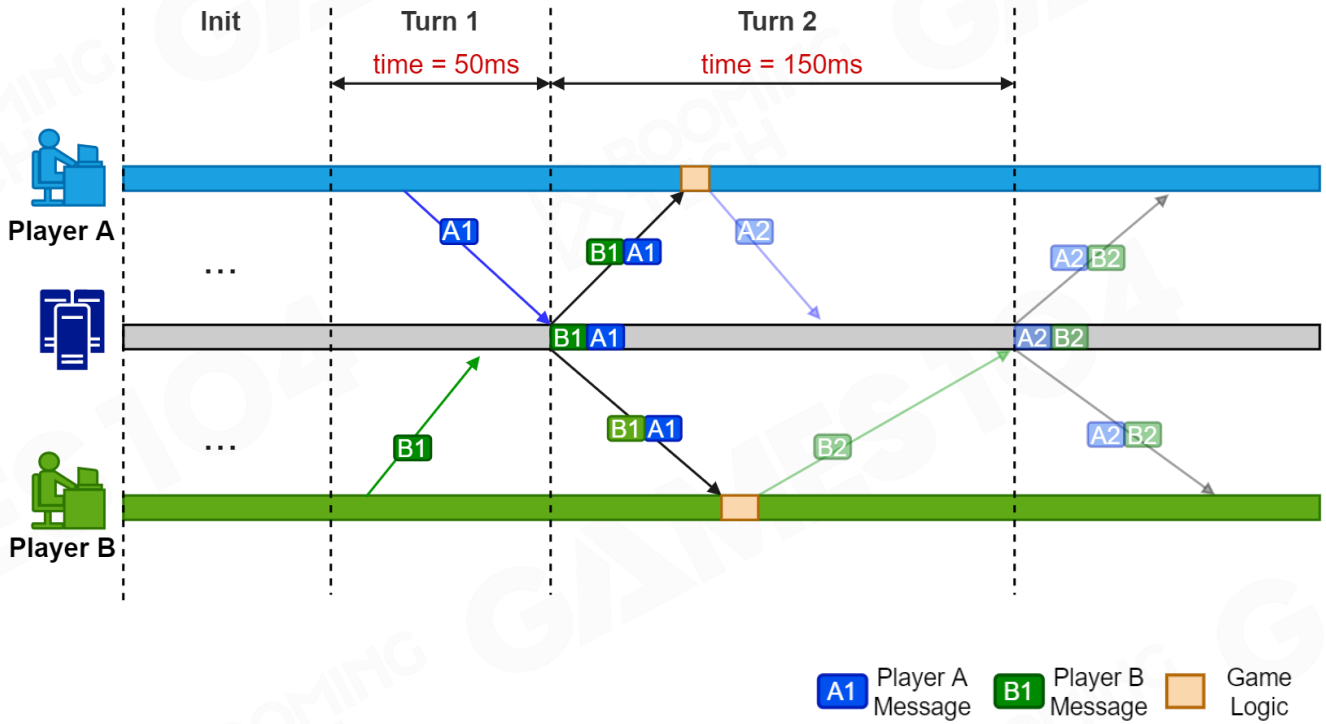
Lockstep Initialization

Loading...

- Ensure that the initial data of each client is deterministic
  - Game model
  - Static data
  - ...
- Synchronize clock

Deterministic Lockstep (1/2)

- Client sends inputs to Server
- Server receives and sorts
- Wait for input from all clients before forwarding
- After receiving data from the server, the client executes the game logic

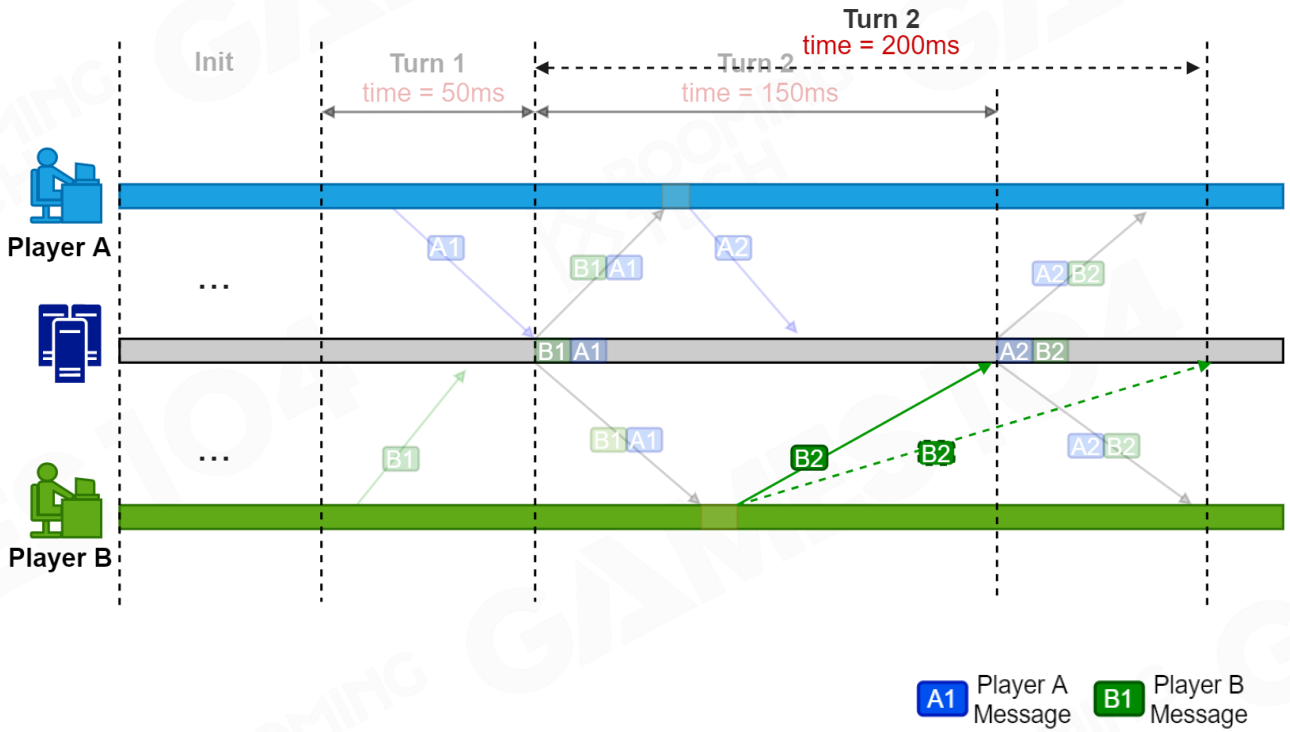


Deterministic Lockstep (2/2)

If Player B's message B2 arrives later?

(The dotted line B2 in the figure)

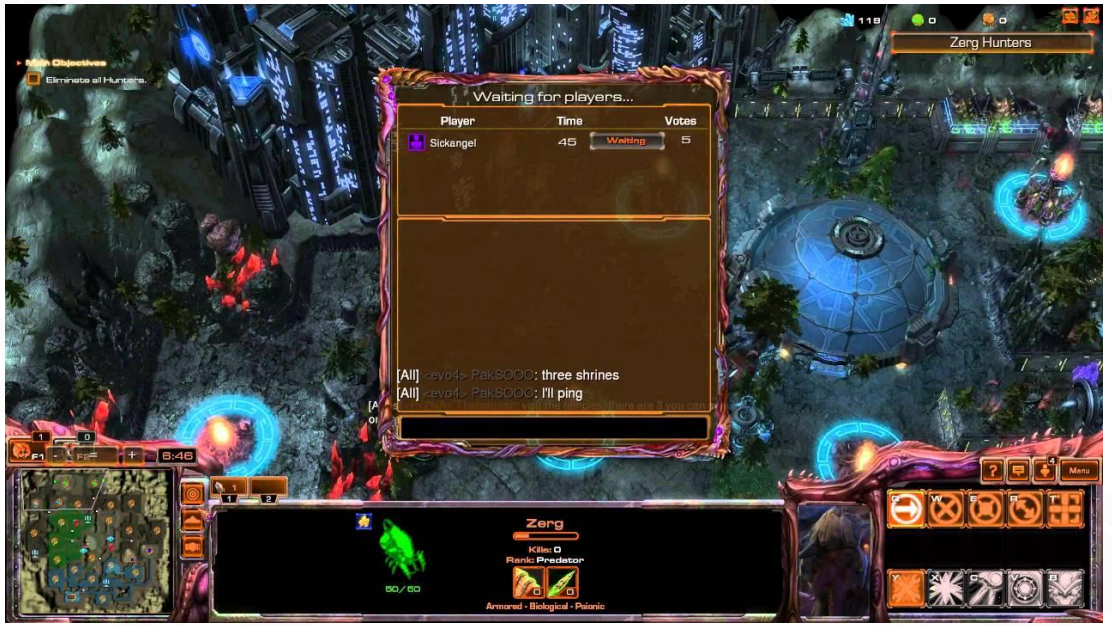
- Disadvantages
  - Game progress depends on slowest player
  - The delay of the game is not fixed, the experience is not good
  - All the players will wait if a player offline



Player Offline in Deterministic Lockstep

- Waiting for players...

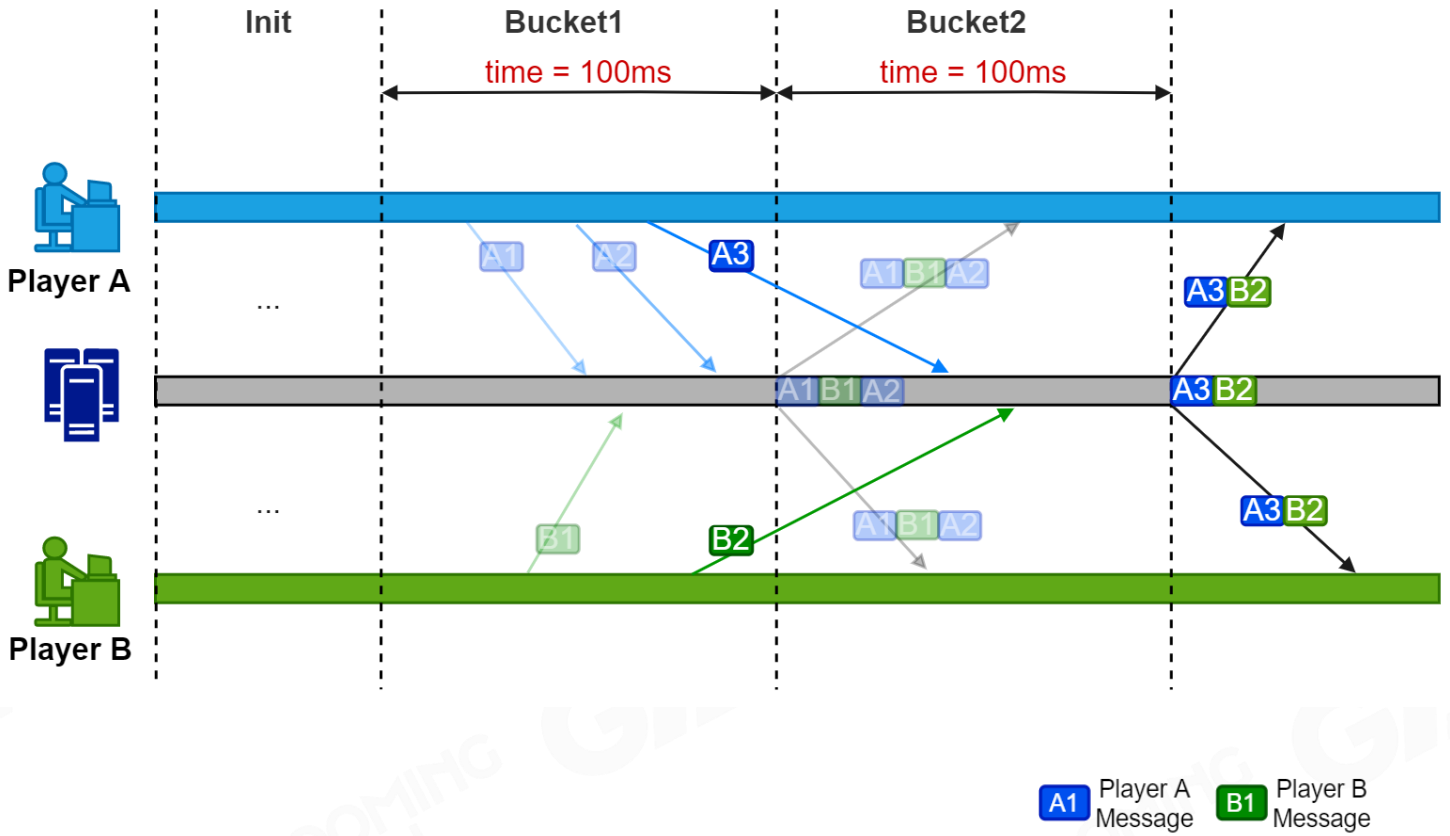
Dota



StarCraft

Bucket Synchronization

- Bucket: a fixed time period
- Each bucket
  - Collect all instructions
  - Broadcast to all players
- There is no need to wait for all players' commands to be received before forwarding

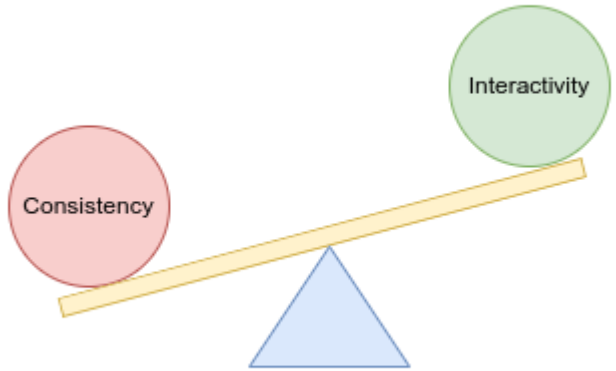


A Good Trade-off between Consistency and Interactivity Maintenance

So we need to find a basis balance between them.

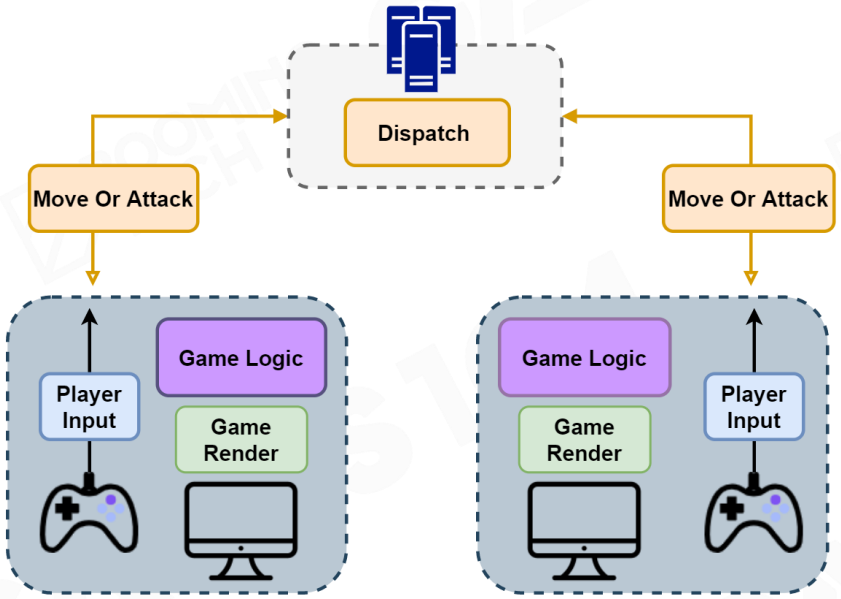
The threshold:

As soon as the measured interactivity degree decreases below a given threshold, take some procedure skips processing obsolete game events with the aim of bringing back a satisfactory interactivity level



Deterministic Difficulties

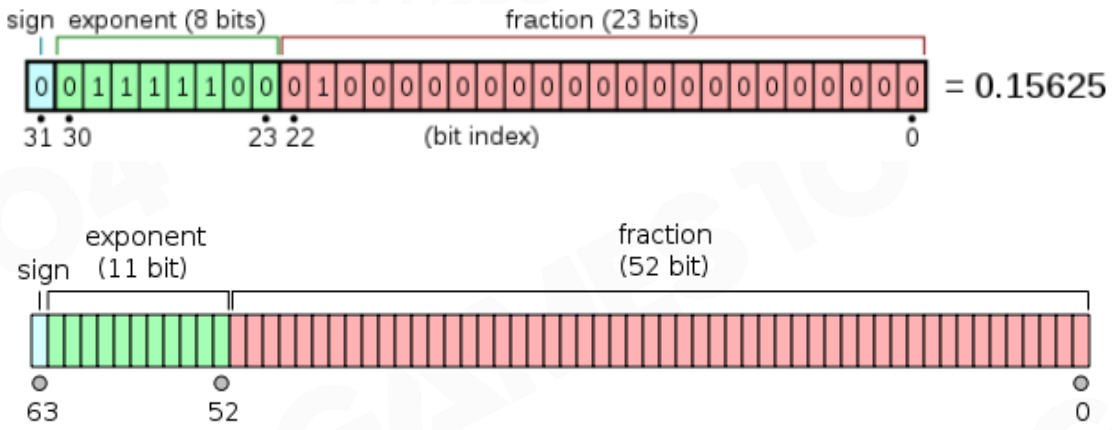
- **Deterministic**
  - The same input sequence need to produce the same game state on all machines
- Deterministic is Hard
  - Floating point
  - Random number
  - Containers and algorithms (sort, add, remove, etc.)
  - Math tools (vectors, quaternions, etc)
  - Physics simulation (very difficult)
  - Code logic execution order



Floating Point Numbers (1/4)

- Because of the computer binary, These numbers can be accurately represented
  - 0.5 = 1/2
  - 0.25 = 1/4
  - 0.75 = 1/2+1/4
  - 0.875 = 1/2+1/4+1/8
- Such numbers can only be approximated
  - 2/3 ≈ 0.66..7

- Floating point numbers must comply with the IEEE 754 standard



Floating Point Numbers (2/4)

- Floating point numbers conform to the IEEE 754 standard
- But different platforms may have different behavior

Floating Point Hardware & OS Behaviour

- Intel / Amd
- PS / Xbox
- Windows / linux
- Android / Ios
- ...

Floating Point Compilers Behaviour

- Math Library(*sin*、*cosin*、*tan*、*exp*、*pow ...*)
- Third party components
- Different platforms
- Different versions
- Different languages
- ...

Floating Point Numbers (3/4)

Idea: Avoid problems on the precision boundary, customize the precision

- Fixed-point math library
- Look-up table (trigonometric functions, etc.)
- Amplification and truncation

Simple method

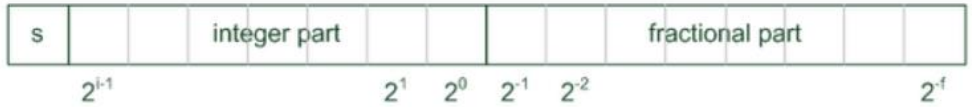
- Multiply by 1000, then divide by 1000, there is an overflow risk
- The numerator and denominator are represented by fixed-point numbers (2/3)
- ...

Floating Point Numbers (4/4)

One Solution: Fixed point math

A fixed-point number can be split into three parts

- An optional sign bit
- An integer
- A fractional part
- Need to implement addition, subtraction, multiplication and division etd.
- Implement class, class methods
- Performance needs to be considered



$$V = (-1)^{b_{f+i}}\left(\sum_{n=0}^{i} 2^n b_{n+f} + \sum_{m=1}^{f} 2^{-m} b_{f-m}\right).$$

Random Number (1/2)

- Random problems in the game
  - Trigger of random events, npc random birthplace
  - A random attribute of the attack, e.g. critical strike chance
  - ...
- These logics are generally implemented with random numbers
- How to implement random logic that is completely consistent for multiple players

Random Number (2/2)

- Random numbers are pseudorandom
- Before the game starts, initialize the random number seed
- For different players' clients, the number of random function calls is fixed, and the generated random numbers are the same

```cpp
int main() {

    std::default_random_engine e;
    std::uniform_int_distribution<int> u(0, 100);
    e.seed(80);

    for (int i = 0; i < 20; i++) {
        std::cout << u(e) << std::endl;
    }

    return 0;
}
```

```
0
52
45
69
62
52
77
31
34
78
68
55
48
77
27
37
69
62
67
35
```

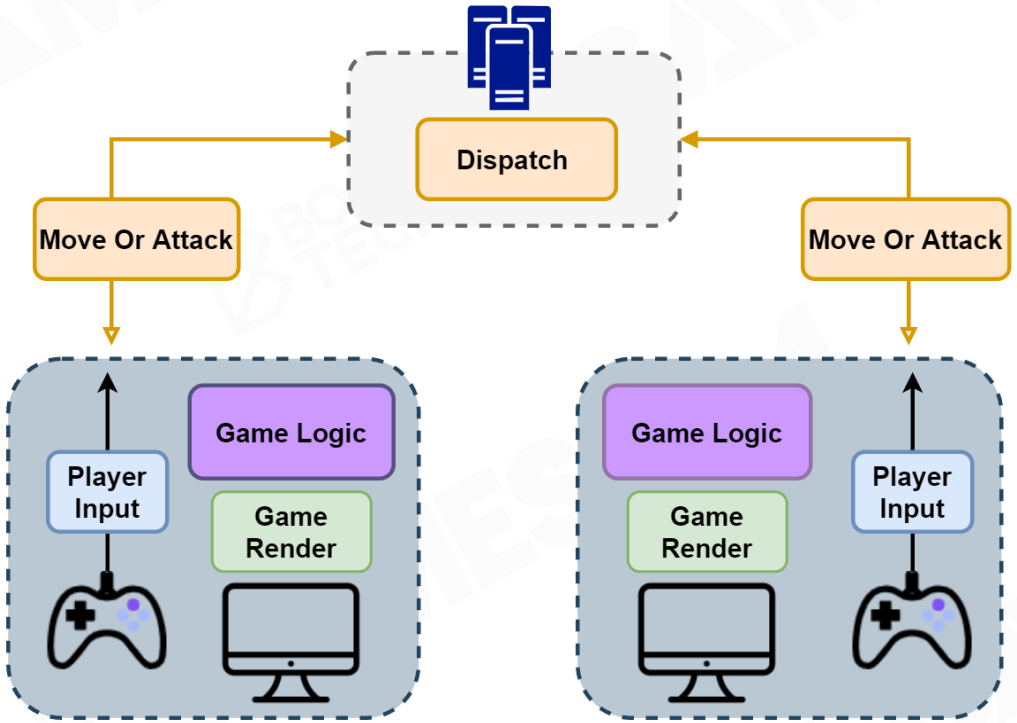
windows 11

```
0
52
45
69
62
52
77
31
34
78
68
55
48
77
27
37
69
62
67
35
```

ubuntu 20.04

Deterministic Solution

- Fixed-point numbers represent floating-point numbers in critical game logic
- Deterministic random algorithm
- Deterministic containers and algorithms (sort, add, remove, etc.)
- Deterministic math tools (vectors, quaternions, etc.)
- Deterministic physics simulation (very difficult)
- Deterministic execution order



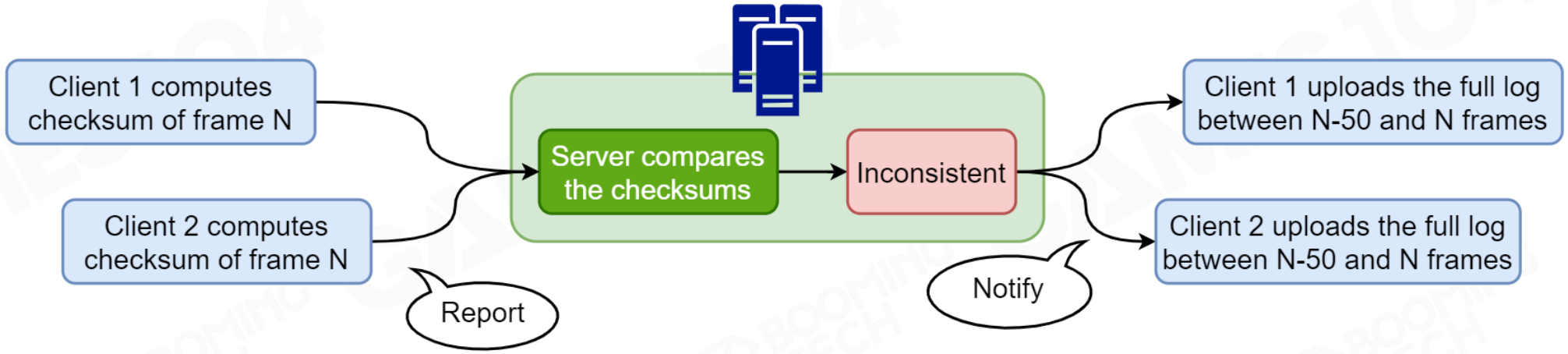
Tracing and Debugging

Method of get checksum

- All data checksum
- Key data checksum
- Other methods

Automatically locate BUG

- Server compares different client's checksums
- Client uploads 50 frames of full logs
- Find inconsistencies in the compared logs



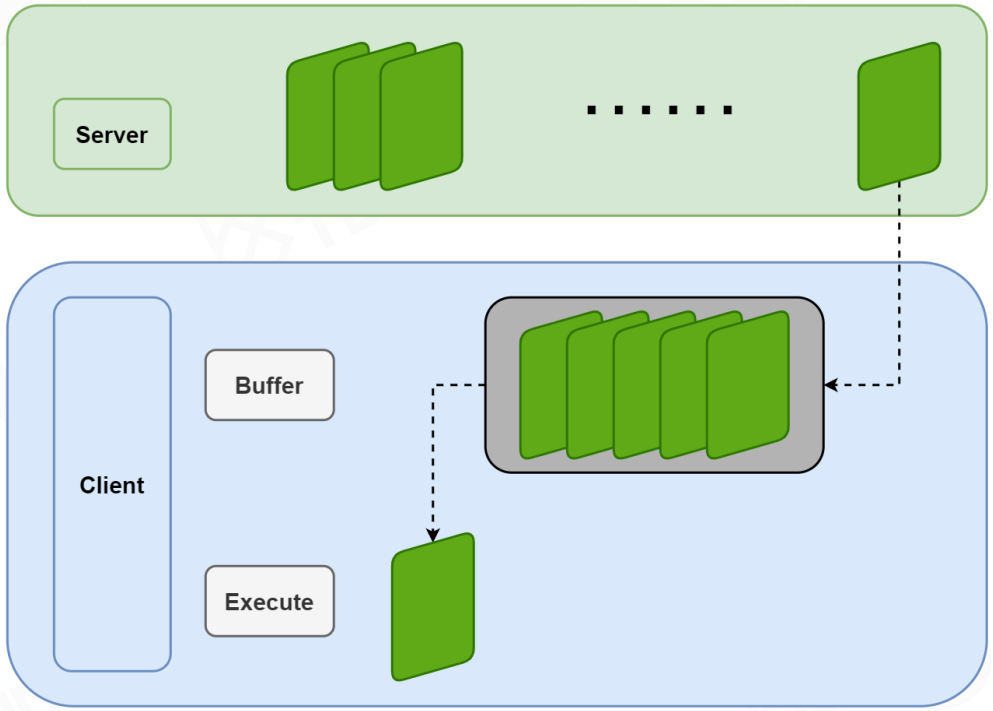
Lag and Delay

- Client send operation
- Receive the operation of this frame from the server
- execute

Lag: Network is unstable. If you wait until you receive new frame, there will be a lag

Solution

- use buffer to cache frames
  - Large buffer, large delay
  - Small buffer, sensitive to lag



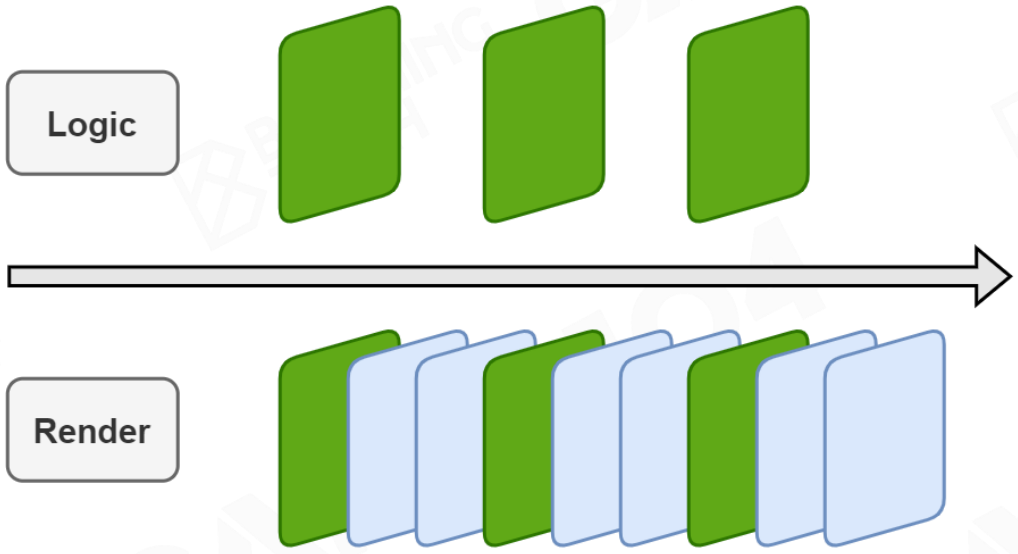
Separating Game Logic from Rendering (1/2)

Lag problem

- Separation of logic and rendering
- Local client-side interpolation smoothing

Frame rate

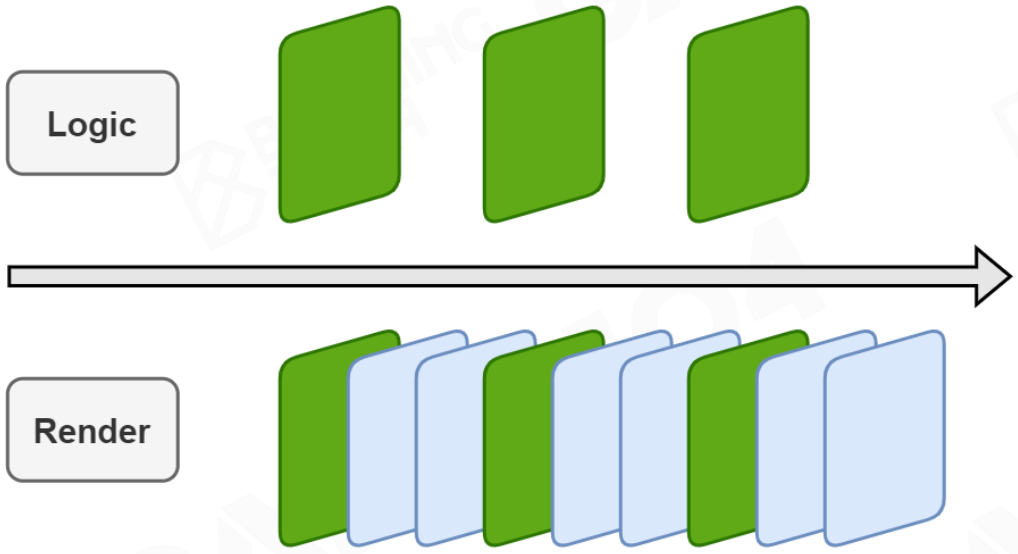
- The logical frame rate is generally 10~30 frames
- The rendering frame rate is generally higher



Separating Game Logic from Rendering (2/2)

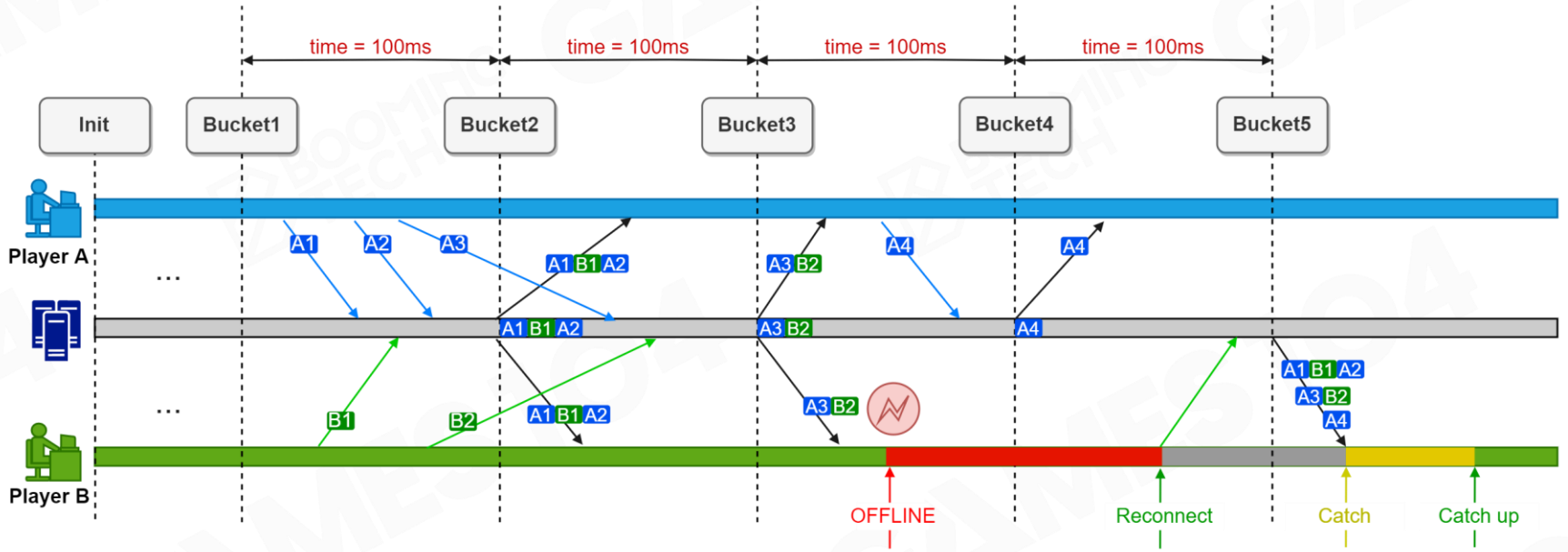
Advantage

- Different frequencies, independent operation
- Rendering separation to avoid tearing and freezing
- Rendering freezes, does not affect the operation of logical frames
- Servers can run logic frames to solve some cheating problems
- If the server runs logical frames, it can save key frame snapshots to speed up reconnection



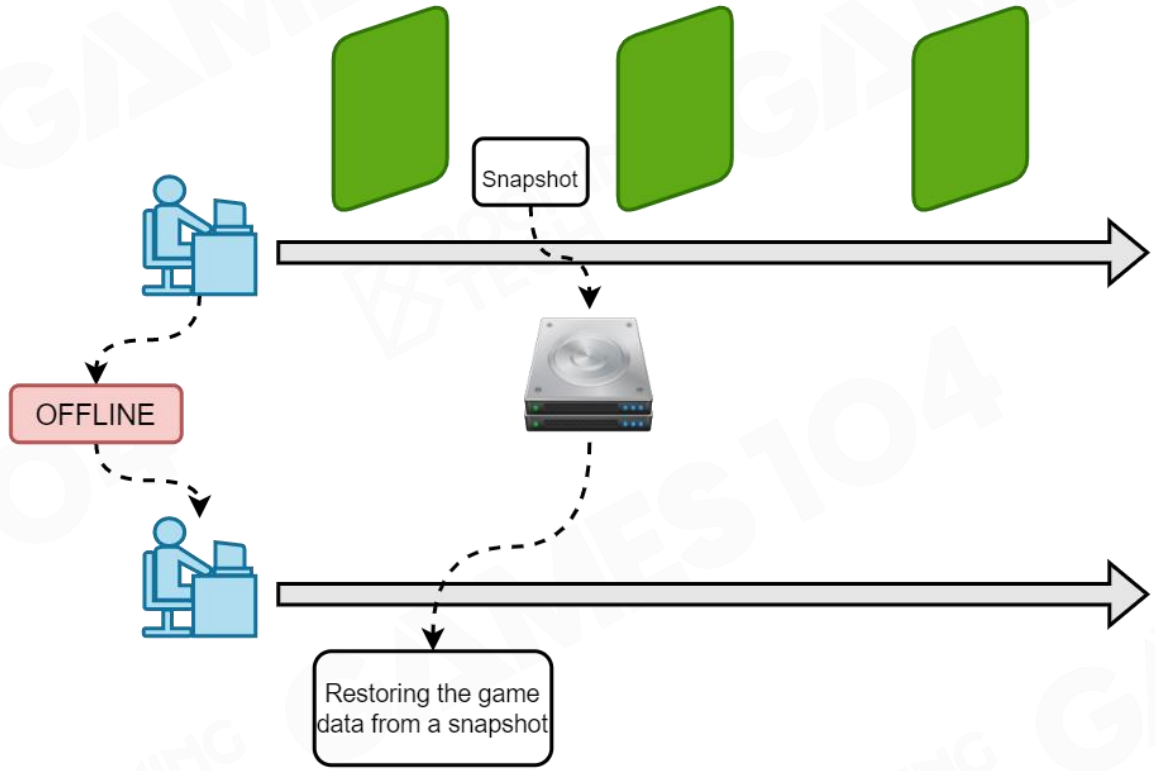
Reconnection Problem

- Offline
- Reconnect
- Catch up



Client Game State Snapshots

- Snapshots can be saved regularly on the local client and serialized to disk
- When reconnection occurs, restore the game state from the disk serialized data
- Server sends player commands after snapshot
- Accelerate to catch up with the game progress



Quick Catch Up

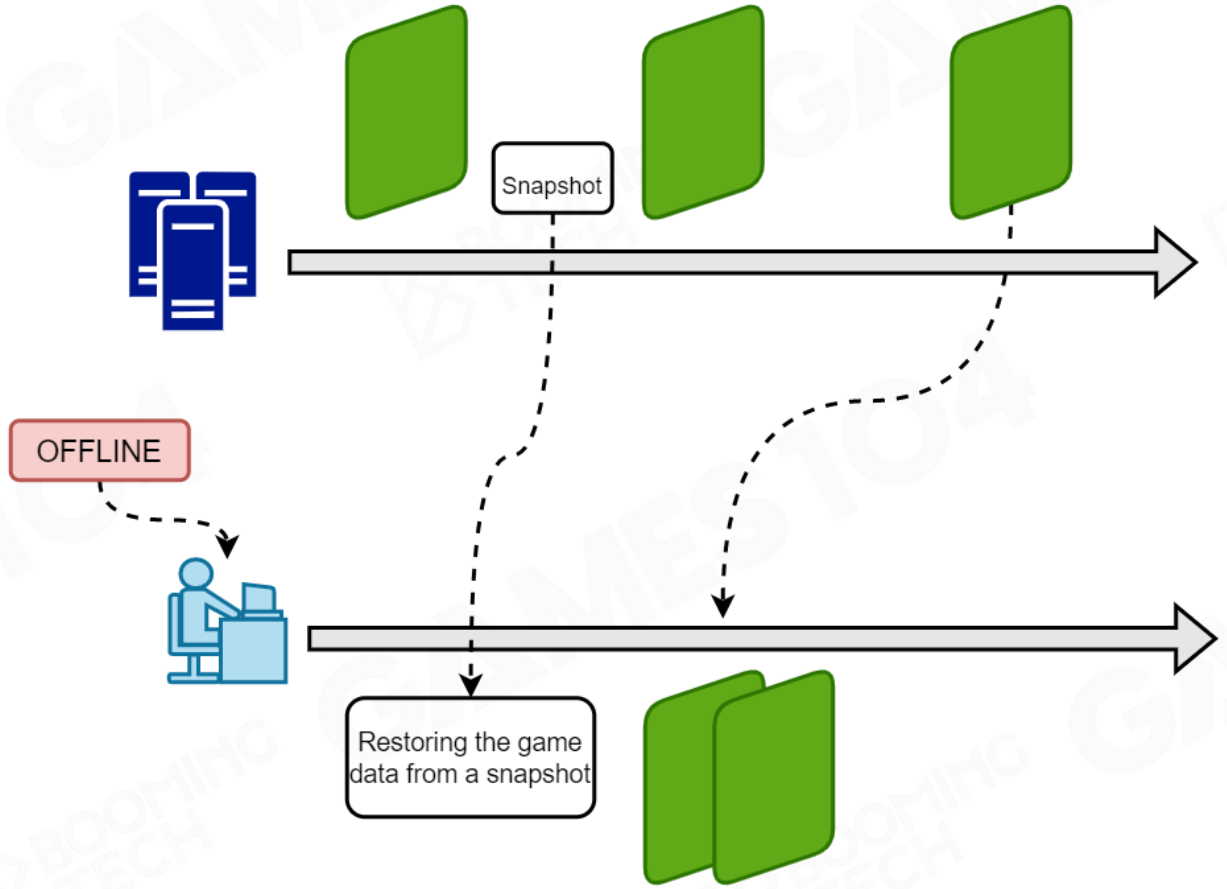
How to catch up?

- In the sample code, chasing 10 frames each time
- If originally 10 frames per second, when chasing frames, it may run 100 frames per second

```
float m_delta = 0;
float m_tick_delta = 100;
void CBattleLayer::update(float delta){
    // do something
    m_delta += delta;

    int exec_count = 1;
    while(m_delta >= m_tick_delta){
        m_delta -= m_tick_delta;

        // logic frame
        if(!logicUpdate(LOGIC_TIME)){
            return;
        }
        // catch up 10 frames at a time
        if(exec_count++ >= 10){
            break;
        }
    }
    // do something
}
```

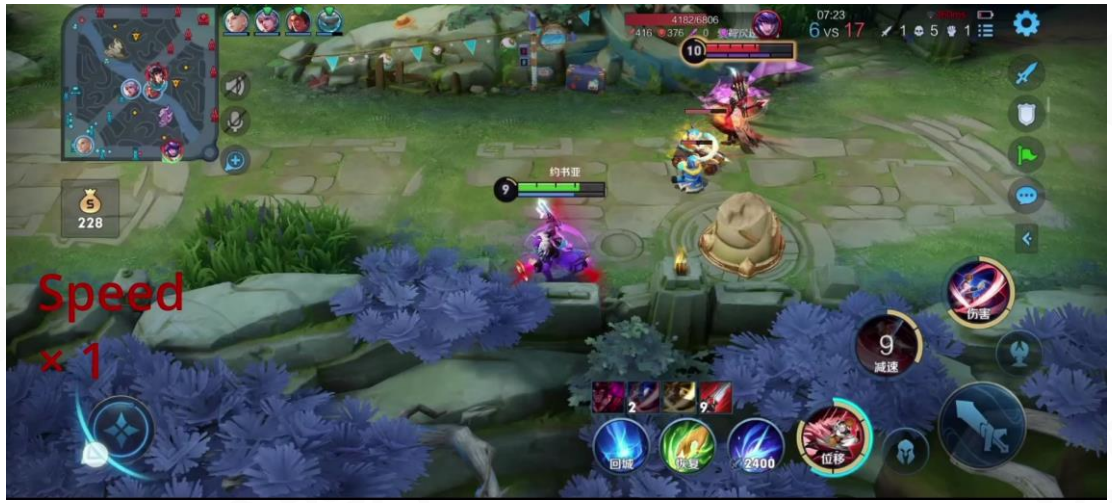
Server State Snapshot Optimization

- The server runs logical frames and saves snapshots of keyframes
- The server sends the snapshot, and the player commands after the snapshot
- Accelerate to catch up with the game progress



Temporary Offline, No Crash

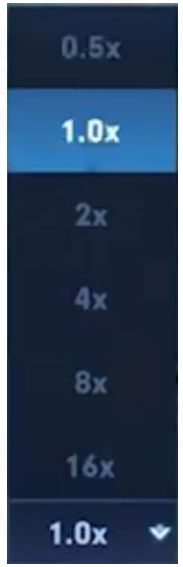
- Client also keeps game state, keyframes, deterministic-timed frames
- After reconnecting, the server sends commands to the dropped player
- Accelerate to catch up the game progress



Observing

Watching other players playing the game

- Reconnecting and watching are essentially the same
- Watching is similar to reconnecting after a client crash
- Player action command, forwarded to the player watching the game
- Watching is usually delayed for a few minutes to prevent screen peeping



Replay

Execute player commands in order which can speed up

- **Replay file**
  - Save game commands for a game
  - Files take up little space

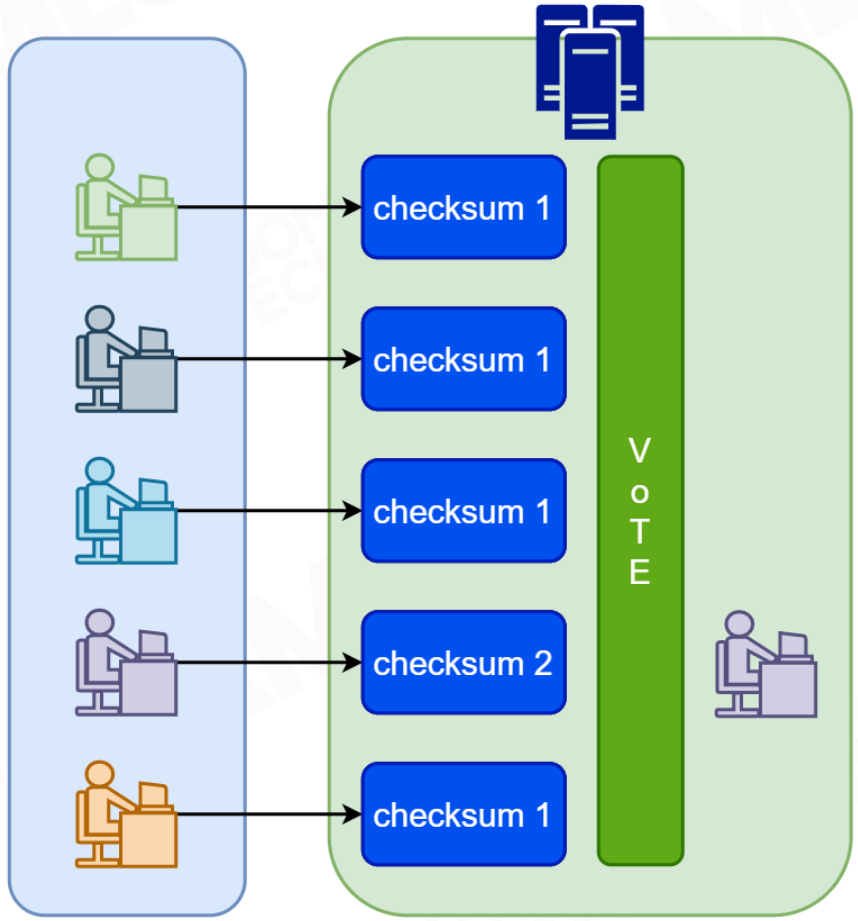
- **How to implement go back?**
  - When the client executes the replay file, it adds a key frame snapshot, which can go back to the key frame moment
  - The current version of Honor of Kings go back to the key frame before 60s

Honor of Kings

Lockstep Cheating Issues (1/3)

Multiplayer-PVP

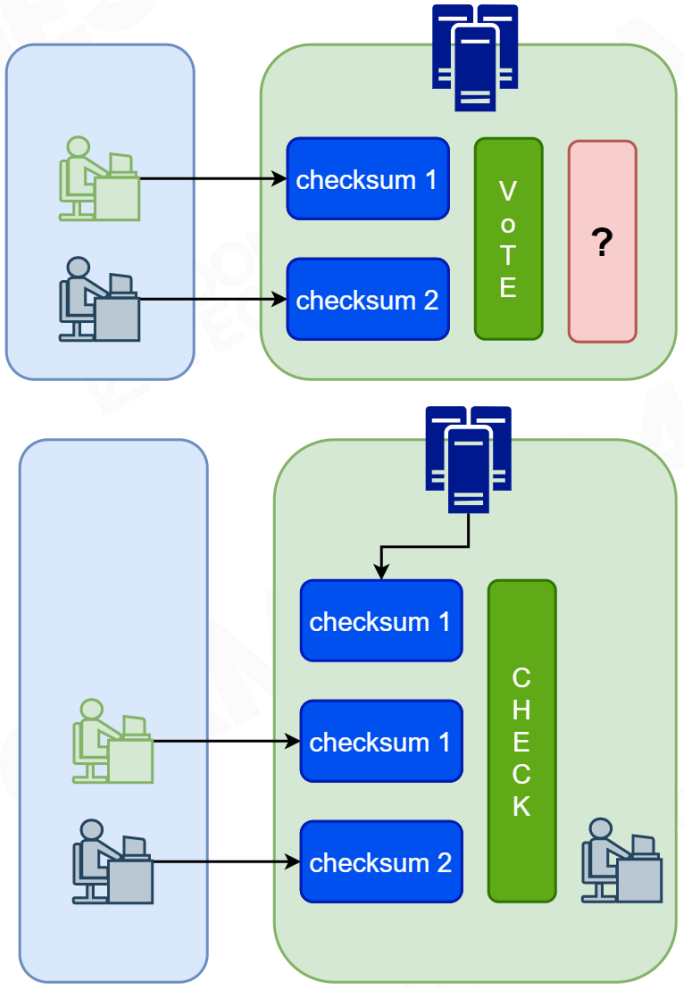
- Game over
  - The client uploads the key data checksum, the server verifys the game result
- During the game
  - Report the key data checksum
  - Cheating players are kicked out, etc.



Lockstep Cheating Issues (2/3)

2 Players

- Server can not detect who is cheating using key data checksum
- If the server is not verified, the cheating player will only affect one player in this case



Lockstep Cheating Issues (3/3)

- Difficult to avoid thirty-party plug-in to access war-fog or other hidden data
  - Game logic is performed on the client side
  - Clients have all the game data

Perspective Plug-in

Lockstep Summary (1/2)

Advantages

- Low bandwidth, only sends commands
- High development efficiency, similar to single-player game development
- Precise action/hit detection
- Easy to record games

Lockstep Summary (2/2)

Problems

- Maintain the consistency is difficult to achieve
- Hard to solve the cheat plugin to unveil all game states
- Longer disconnection and reconnection time
  - Need more complex optimization

State Synchronization

State Synchronization

State Synchronization

Replication Protocol of Halo

State Data

Guaranteed eventual delivery of most current state
- Object position
- Object health
- 150+ properties

Events

Unreliable notifications of transient occurrences
- Please fire my weapon
- This weapon was fired
- Projectile detonated
- More events

Control Data

High-frequency, the best-effort transmission of rapidly-updated data extracted from player control inputs
- Current analog stick values for all players
- Current position of client's own biped
- More properties

State Synchronization

State

- The game state is necessary to represent the game world. e.g: HP, MP

State Synchronization

- Server does not generate a single update for all clients. It sends client a customized data packet
- If the game world is too complex, you can set an Area Of Interest (**AOI**) for reducing server overhead

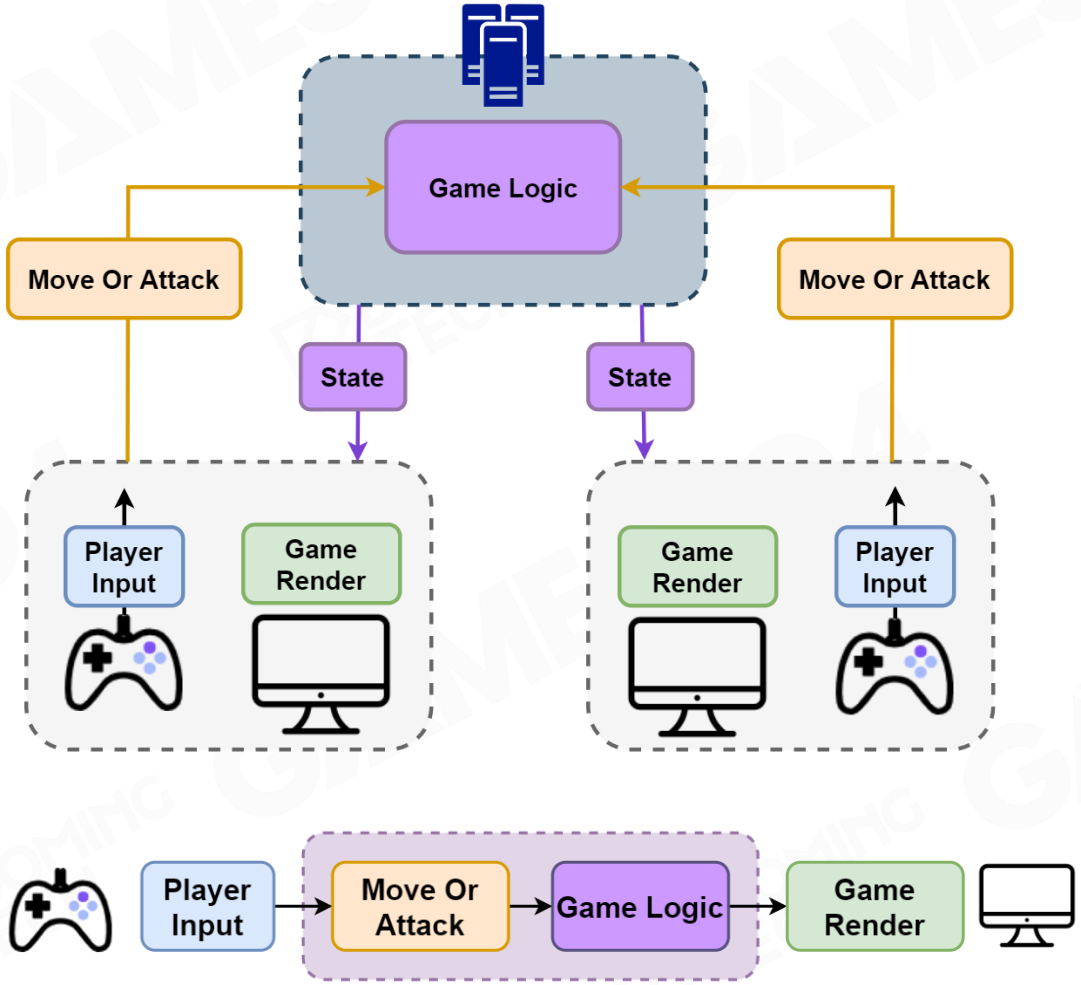
Server Authorizes the Game World

Server

- Game world is authorized
- Receive input and state from client
- Run game logic
- Send state

Client

- Receive data and simulate game world
- Game play improvement



Authorized and Replicated Clients

Authorized (1P)

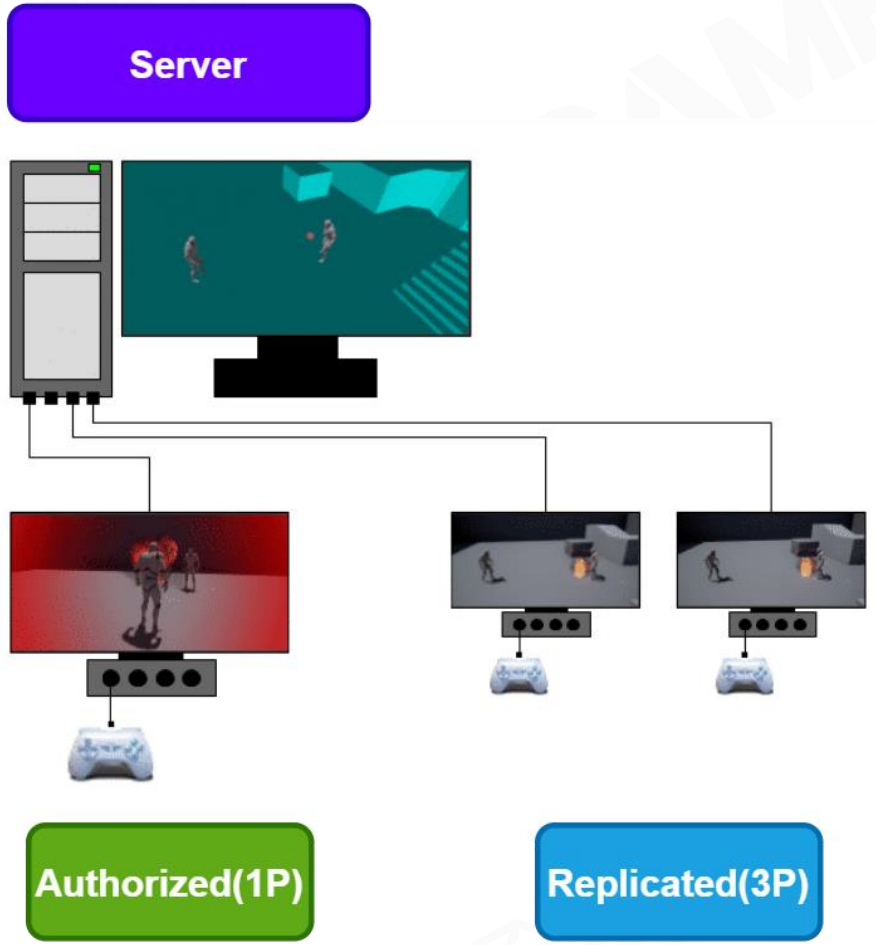
- Player's local game client

Server

- Authorized server

Replicated (3P)

- Simulated character in other player's client



State Synchronization Example (1/4)

Player1 (Authorized)

- Fire

Player2 (Replicated)

- See player1 open fire

State Synchronization Example (2/4)

Player 1 presses an input on their local machine to fire

Player1 (Authorized)
- Fire
- Send to server

Server
- Player1 fire
- Send to each client

Player2 (Replicated)
- Recieve packet
- Player1 fire

State Synchronization Example (3/4)

Server

- Tell each client to replicate the movement of Player 1's projectile

State Synchronization Example (4/4)

Server

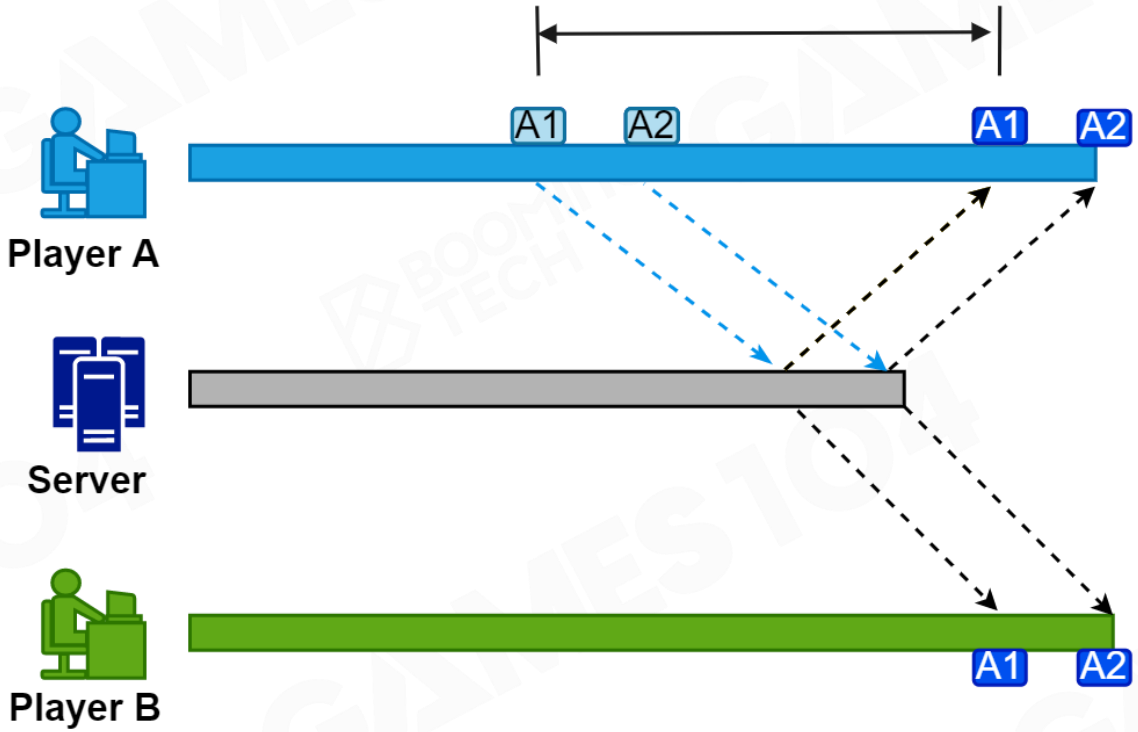
- Tell each client to destroy their copies of Player 1's projectile
- Tell all clients to response to damage of the projectile

Dumb Client Problem

Clients can not to do anything until receive server state update

How to see an immediate response?

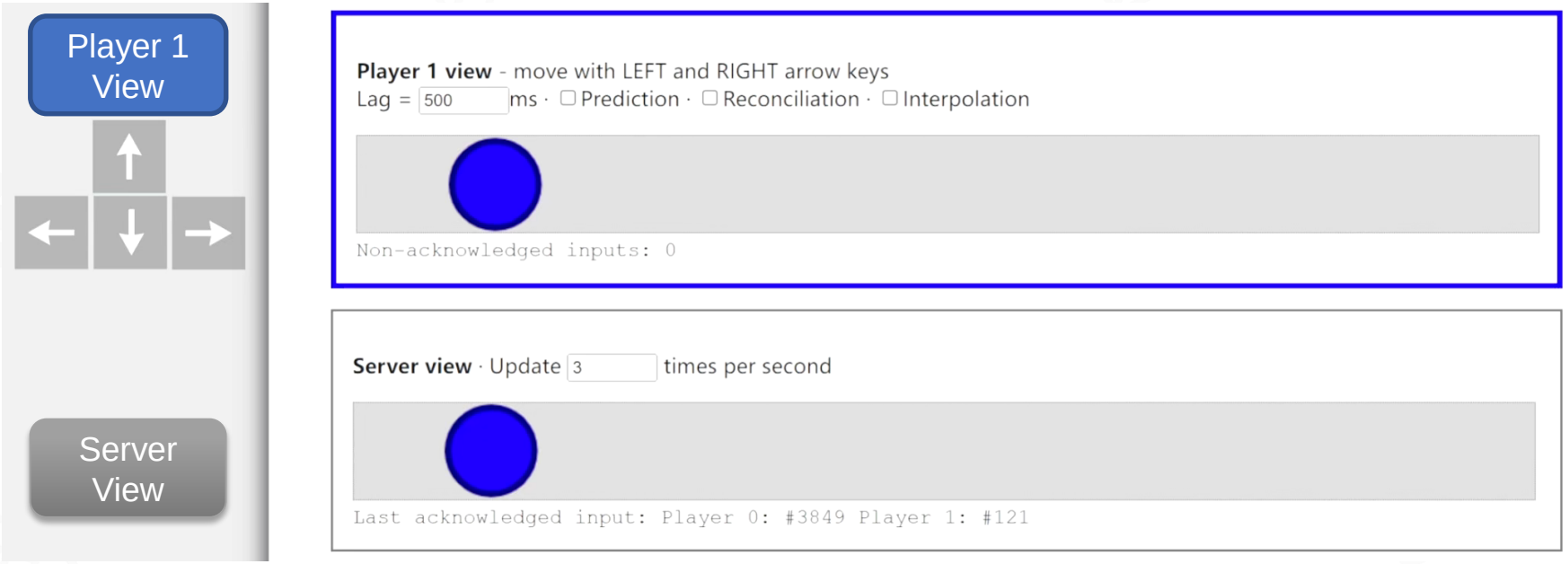
- **Client-side prediction**
- **Server reconciliation**



Client-Side Prediction

Authorized client

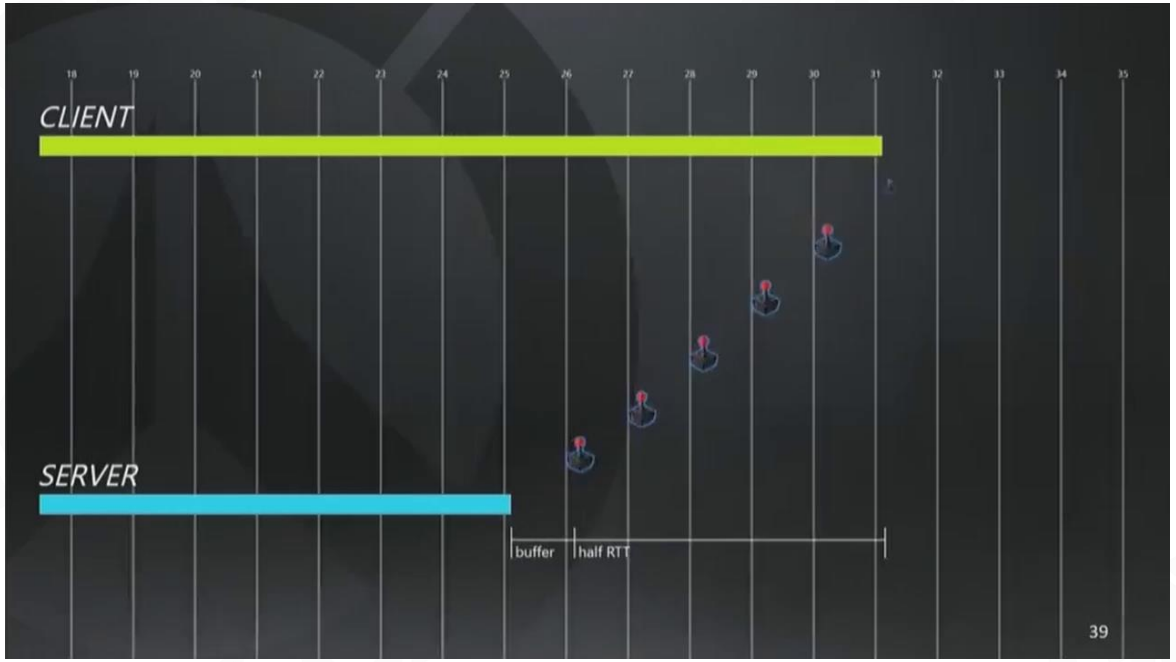
- Press “→”
- Received server message
- Start movement



Overwatch - Client-side Prediction

- RTT = 160ms
- Half RTT = 80ms
- Command frame = 16ms

The client is always ahead of the server by half RTT and one buffered command frame

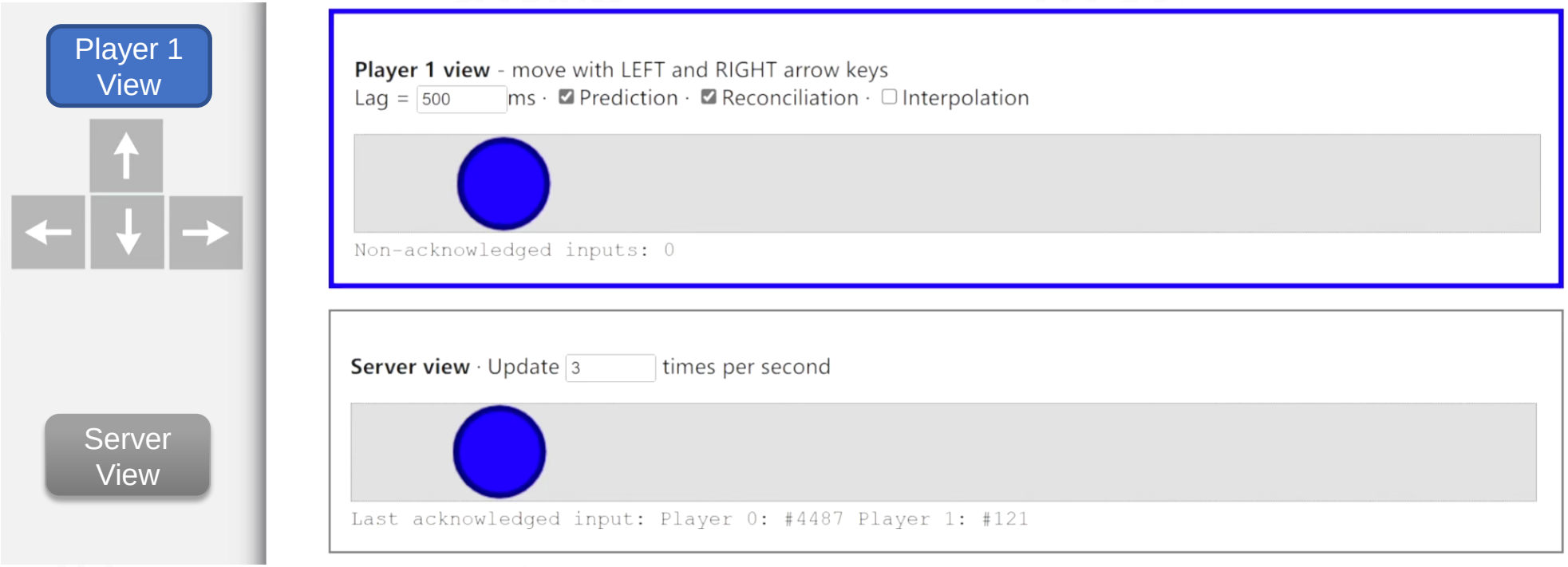


- Press key and response immediately

Server Reconciliation (1/4)

Authorized client: Buffer

- Record every state when do the client prediction
- Compare with the past server data when it was received on client side



Server Reconciliation (2/4)

Ring buffer for states

- Stores all of our states in the past several frames on client

Process

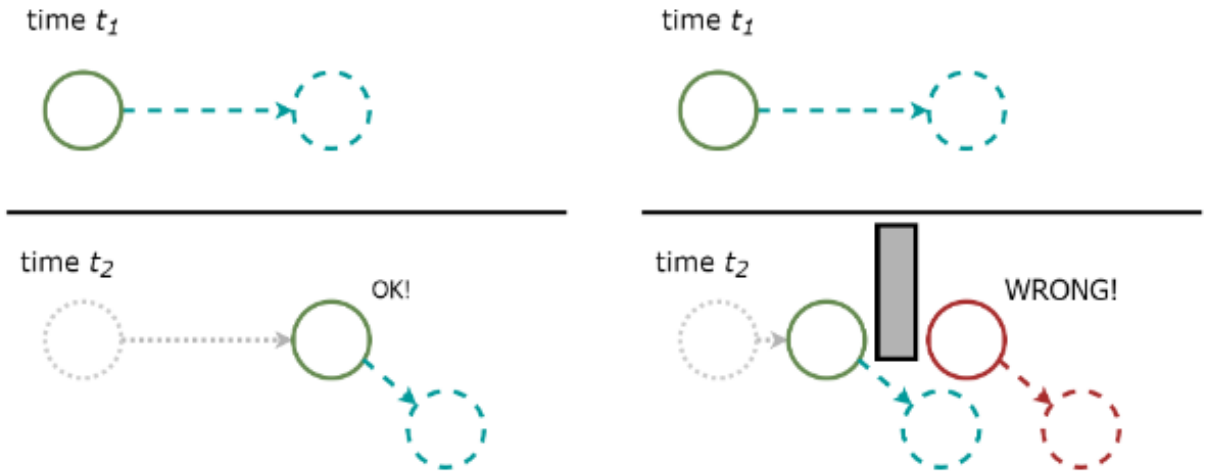
- If the client computed the same result as server, the client will continue on its merry way to simulate the next input

Problem

- If misprediction?

Server Reconciliation (3/4)

- If blocked by an obstacle at the server
- Position is wrong! (in red)
- Client must accept the new server update
- Retrace all predicted movement starting from the new confirmed position



Server Reconciliation (4/4)

- If the client and server disagree on the results, we've mispredicted
- Have to reconcile

Ring buffer for inputs

- Stores all of inputs we did in the past several frames on client.

Process

- Overwrite the clients results with the server's results
- Replay all of your inputs to catch back up to what you believe now

Server Reconciliation Example

Overwatch

- We try to move
- The server said no
- We got yanked back down to where we were before and froze

Mei's blaster unleashes a concentrated, short-range stream of frost that damages, slows, and ultimately freezes enemies in place.

Packet Loss

- Client input packages fail to reach the server
- The server tries to keep tiny **input buffer** of unprocessed input
- If the server run out of input buffer, server will **duplicate** your last input in a window
- Push client sends missed inputs asap

State Synchronization Vs. Lockstep Synchronization

	State Synchronization	Lockstep Synchronization
Deterministic Logic	Not necessary	Necessary
Response	Better responsiveness	Poor responsiveness
Network Traffic	Usually high	Usually low
Development efficiency	Much more complicated	Easy to develop, difficult to Debug
Number of players	Few players	Support small and large numbers of players
Cross-platform	Relatively easy	Relatively difficult
Reconnect	Relatively easy	Relatively difficult
Replay File Size	Big	Small
Cheat	Relatively hard	Relatively easy

Are We Ready to Make a Online Game?

To be continued…

Lecture 18 Contributor

- 德辉
- Peter
- Ximenes
- yf
- 鸭毛
- BOOK
- 伟哥
- 邓导
- Jason
- 阿鹏
- 大喷
- 爵爷
- Brooklyn

References

Network Protocols

- Internet protocol suite:
  https://en.wikipedia.org/wiki/Internet_protocol_suite
- OSI Model:
  https://www.practicalnetworking.net/series/packet-traveling/osi-model/
- Automatic repeat request:
  https://en.wikipedia.org/wiki/Automatic_repeat_request
- Network Communication and Remote Procedure Calls (RPCs):
  https://www.cs.princeton.edu/courses/archive/fall18/cos418/docs/L2-rpc.pdf
- What are TCP and UDP?
  https://www.gdyunjie.cn/showinfo-114-788-0.html

Network Synchronization (1/3)

- Synchronization Issues with Smart Solutions: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.352.6405&rep=rep1&type=pdf
- An Efficient Synchronization Mechanism for Mirrored Game Architectures: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.87.6043&rep=rep1&type=pdf
- The Brave New World of Multiplayer Online Games - Synchronization Issues with Smart Solutions, Marco Roccetti, Stefano Ferretti, Claudio E. Palazzi: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.352.6405&rep=rep1&type=pdf
- Network Time Protocol(NTP):https://www.rfc-editor.org/rfc/rfc958.html
- Algorithms for Synchronizing Network Clocks:https://datatracker.ietf.org/doc/html/rfc956
- "Minimizing Latency in RealTime Strategy Games", Jim Greer, Zack Booth Simpson, Game Progamming Gems 3 chapter 5.1, 2001: https://archive.org/details/game-programming-gems-3

Network Synchronization (2/3)

- JMP van Waveren, "The DOOM III Network Architecture", 2006: https://mrelusive.com/publications/papers/The-DOOM-III-Network-Architecture.pdf
- Christophe DIOT, Laurent GAUTIER, "A Distributed Architecture for Mult iplayer Interactive Applications on the Internet", IEEE, 1999: https://ieeexplore.ieee.org/abstract/document/777437/
- Mark Terrano, Paul Bettner "Network Programming in Age of Empires and Beyond". GDC 2001: https://www.gamedevs.org/uploads/1500-archers-age-of-empires-network-programming.pdf
- 腾讯游戏, 腾讯游戏开发精粹. 电子工业出版社, 2019.9: https://gameinstitute.qq.com/game-gems
- Cocos, 帧同步游戏在技术层面的实现细节, 2021 https://zhuanlan.zhihu.com/p/408734657

Network Synchronization (3/3)

- IEEE 754: https://en.wikipedia.org/wiki/IEEE_754
- QUAKE 3 SOURCE CODE REVIEW - NETWORK MODEL. 2012: https://fabiensanglard.net/quake3/network.php
- David Aldridge, I Shot You First - Networking the Gameplay of HALO - REACH. GDC 2011: https://www.gdcvault.com/play/1014345/I-Shot-You-First-Networking
- Timothy Ford, "Overwatch Gameplay Architecture and Netcode". GDC 2017: https://www.gdcvault.com/play/1024001/-Overwatch-Gameplay-Architecture-and
- Unreal engine Document: https://docs.unrealengine.com/5.0/en-US/networking-overview-for-unreal-engine/
- Gaffer on Games: https://gafferongames.com/#posts

Q&A

Enjoy ;)
Coding

Course Wechat

Please follow us for further information