NETWORK PROTOCOL RFC DOCUMENTATION

BKFP EPITECH BERLIN

This document describes a network protocol for communication between components of a networked game engine. The protocol uses UDP (User Datagram Protocol) for efficient and low-latency communication. It is designed for educational purposes as part of a school project.

Contents

1. Introduction

This document provides an overview and detailed specifications of a network protocol used in a networked game engine. The protocol is intended for educational purposes and utilizes UDP for communication.

2. Protocol Overview

1. Components

The protocol consists of two main components:

Sender: Responsible for sending messages to a specified IP address and UDP port.

Receiver: Listens for incoming messages on a specified UDP port and processes them asynchronously.

2. Message Exchange

Messages are exchanged between the sender and the receiver using UDP datagrams. The sender sends messages to a specific IP address and UDP port, while the receiver listens on the designated UDP port for incoming messages.

3. Message Format

Messages exchanged between the sender and the receiver have a simple format:

Payload: The content of the message, typically a string.

Please see the Network Message Norm Documentation for more details on the network message norm.

4. Sender Behaviour

The sender class has a send method that allows the sending of messages. It opens a UDP socket, sends the message payload to the specified IP address and UDP port, and then closes the socket.

```
class Sender:
    constructor(ip_address, udp_port):
        this.ip_address = ip_address
        this.udp_port = udp_port

    method send(message):
        try:
            create_udp_socket()
            prepare_message_packet(message)
            send_packet_to(ip_address, udp_port)
            close_socket()
        catch error:
            log_error(error)

    method create_udp_socket():
        // Create a UDP socket for sending
        socket = create_udp_socket()

    method prepare_message_packet(message):
        // Prepare the message packet to be sent
        packet = create_packet(message)

    method send_packet_to(ip, port):
        // Send the packet to the specified IP and port
        send(packet, ip, port)

    method close_socket():
        // Close the UDP socket
```

Sender pseudo-code use-case

5. Receiver Behaviour

The receiver class sets up a UDP socket, binds it to a specified UDP port, and enters a loop to listen for incoming messages. When a message is received, it processes it asynchronously.

```
class Receiver:
    constructor(udp_port):
        this.udp_port = udp_port

    method receiver():
        try:
            create_udp_socket()
            bind_socket_to_udp_port(udp_port)
            listen_for_incoming_messages()
        catch error:
            log_error(error)

    method create_udp_socket():
        // Create a UDP socket for receiving
        socket = create_udp_socket()

    method bind_socket_to_udp_port(port):
        // Bind the socket to the specified UDP port
        bind(socket, port)

    method listen_for_incoming_messages():
        loop:
            packet, sender_address = receive_packet()
            process_received_packet(packet)

    method receive_packet():
        // Receive a UDP packet and get sender's address
        packet, sender_address = receive()
        return packet, sender_address

    method process_received_packet(packet):
        // Process the received packet and handle it accordingly
```

Receiver pseudo-code use-case

6. Error Handling

Both the sender and the receiver implement basic error handling. Errors are logged, but no sophisticated error recovery mechanisms are included in this protocol.

7. Security Considerations

This protocol does not include any security features. It is intended for educational purposes only and should not be used in production environments without implementing appropriate security measures.

8. Conclusion

The described network protocol provides a simple foundation for communication between components of a networked game engine. It uses UDP for low-latency communication and serves as a starting point for further development and learning.

9. References

Boost.Asio Documentation - Reference documentation for the Boost.Asio library.

Network Message Norm Documentation - Documentation detailing the format and norms for network messages in this protocol.