Network Discovery and Secure Command Communication Overview

Overview of Network Discovery System

The **Network Discovery** system facilitates local network communication between different Unity applications, allowing them to discover and establish connections seamlessly on the same local network. It operates in two modes:

Server Mode

- When running as a server, the system waits for discovery broadcasts from potential clients.
- Upon receiving a broadcast, it validates the broadcast's authenticity by checking a shared authentication key hash and verifying the freshness and uniqueness of the request using timestamps and nonces.
- If the request is valid, the server responds with its network details (port and authentication token hash), enabling the client to establish a connection.

Key Components:

- **DiscoveryBroadcastData**: Sent by clients, includes hashed authentication tokens, timestamps, and unique identifiers (nonces).
- **DiscoveryResponseData**: Sent by the server as a response, containing authentication hash and network port details necessary for clients to connect securely.
- Security measures include hashing (SHA-512) for authentication and a NonceManager to prevent replay attacks.

Overview of Message System / Command System

The Message System (or Command System) facilitates secure communication between networked devices in Unity projects.

How It Works:

- Commands are defined by unique identifiers, ensuring stable and clear command references.
- When a client connects, the Command Coordinator handles this event and can send commands to clients, either immediately or after a specified delay.
- Commands are serialized and encrypted using AES encryption with a predefined key and Initialization Vector (IV), ensuring secure transmission.
- Messages can be sent either to specific clients or broadcasted to all connected clients.

- On reception, messages are decrypted, processed, and the corresponding actions are triggered within the game logic.

Key Components:

- `CommandMessenger`: Abstract base class responsible for sending and receiving encrypted messages.
- `CommandsCoordinator`: Extends `CommandMessenger` and manages connection events, sending delayed or immediate messages.
- `CommandHash`: Computes stable hashes for command identifiers to avoid reliance on hard-coded strings.
- `EncryptionUtils`: Handles AES encryption and decryption of command data for secure message exchange.