EdenNetwork Documentation

목차

EdenNetwork

EdenNetwork는 TCP/UDP 소켓통신을 쉽게 사용할 수 있게 도와주는 네트워크 라이브러리입니다. 이 문서에서는 EdenNetwork의 구조와 간단한 사용법을 소개하겠습니다.

1. EdenNetwork 소개

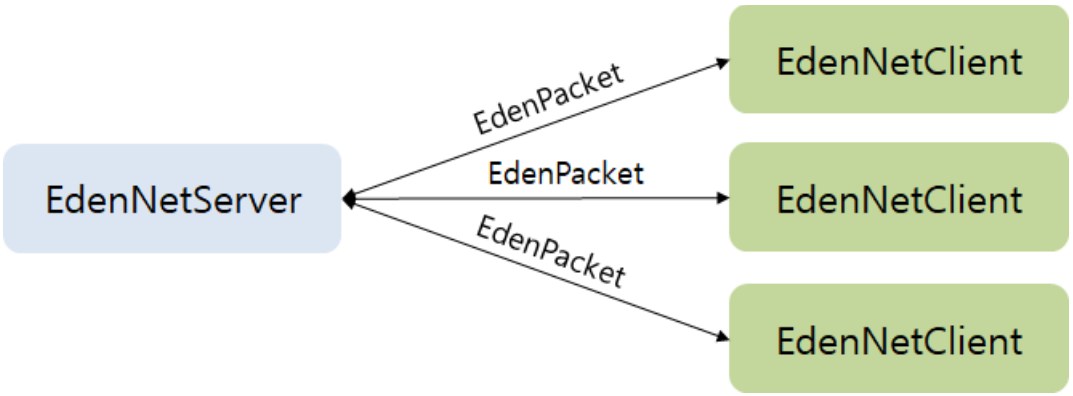

그림 1. EdenNetwork의 구조

EdenNetwork는 EdenNetServer와 EdenNetClient 두 개의 클래스가 서버와 클라이언트의 역할을 하며 1:N으로 통신합니다. 서버와 클라이언트가 모두 송수신을 할 수 있으며 송신은 Send와 Request 두가지 방법을 통해 이루어지고 수신 또한 Receive와 Response 두가지 방법을 통해 이루어집니다. 모든 데이터 교환은 EdenPacket이라는 형식으로 이루어집니다.

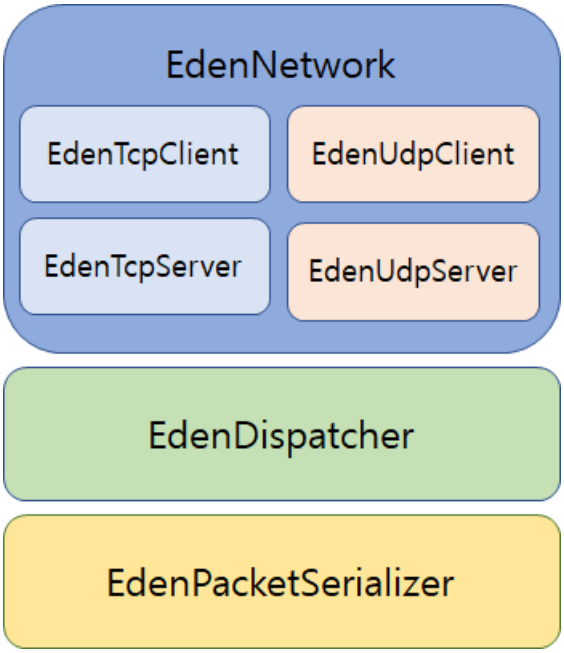

그림 2. EdenNetwork의 주요 모듈

EdenNetwork는 3가지 모듈에 의해 동작합니다. 먼저 네트워크에서 데이터를 송수신하는 EdenNetwork의 4가지 TCP/UDP 구현체 EdenTcpClient, EdenTcpServer, EdenUdpClietn, EdenUdpServer가 있습니다. 두번째는 네트워크에서 패킷을 수신했을 때 알맞은 엔드포인트 로직으로 패킷을 보내주는 EdenDispatcher입니다. 마지막으로 EdenPacketSerializer는 패킷을 송신, 수신할 때 패킷을 직렬화, 역직렬화하는 EdenPacketSerizlier입니다. EdenNetwork에서는 직렬화 방식으로 Google Protocol Buffer를 사용합니다. Serializer는 사용자가 필요에 맡게 변경할 수 있습니다.

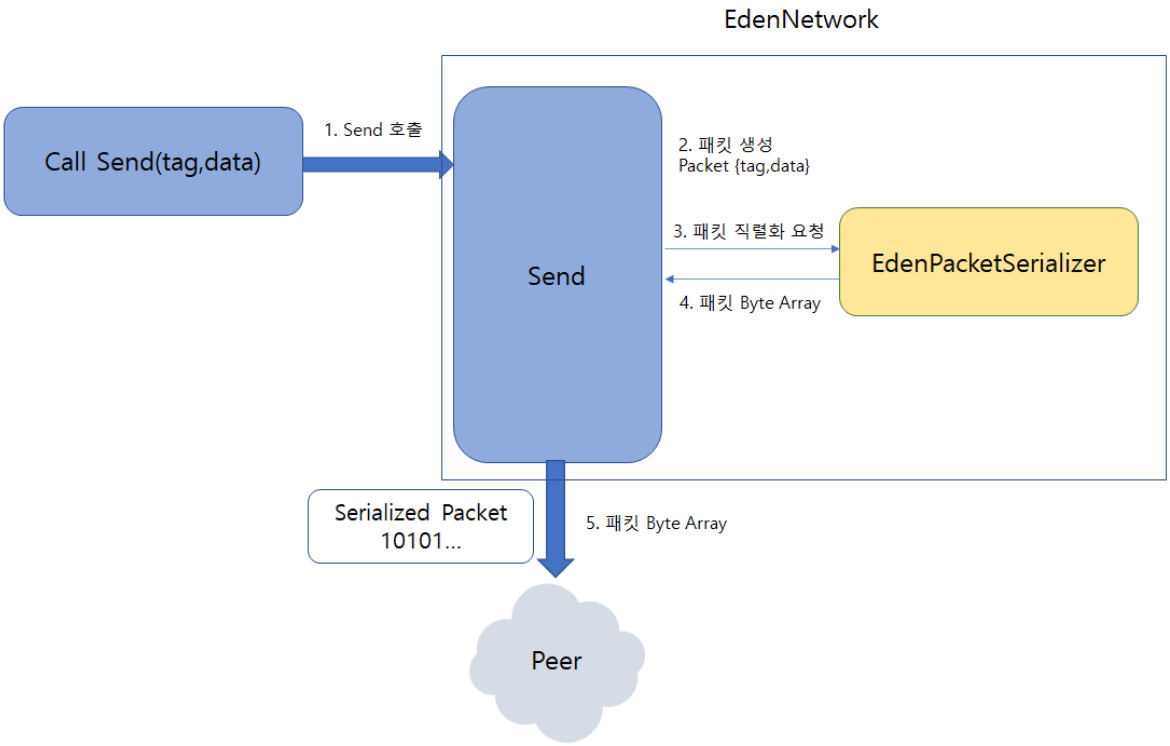

그림 3. EdenNetwork의 송신방식

EdenNetwork에서 송신 시에는 tag와 data를 보냅니다. Tag는 수신자의 Endpoint를 로직을 특정하는 값이고 data는 해당 로직에 매개변수로 전달할 값입니다. Send 메소드를 호출할 시 메소드에서 tag와 data로 패킷을 생성하고 EdenPacketSerializer를 이용해 패킷을 직렬화해 상대방에게 보냅니다.

EdenNetwork는 데이터 수신을 위해 별도의 스레드를 운영합니다. 직렬화 된 패킷이 네트워크를 통해 수신되면 수신 스레드에서 EdenPacketSerializer를 통해 수신된 패킷을 역직렬화 합니다. 이 때 패킷의 tag만 역직렬화합니다. Tag만 역직렬화 된 패킷을 EdenDispatcher에 보내면 tag를 통해 엔드포인트를 찾고 엔드포인트에서 요구하는 매개변수 타입으로 data를 역직렬화 한 후 data를 매개변수로 엔드포인트 로직을 실행합니다.

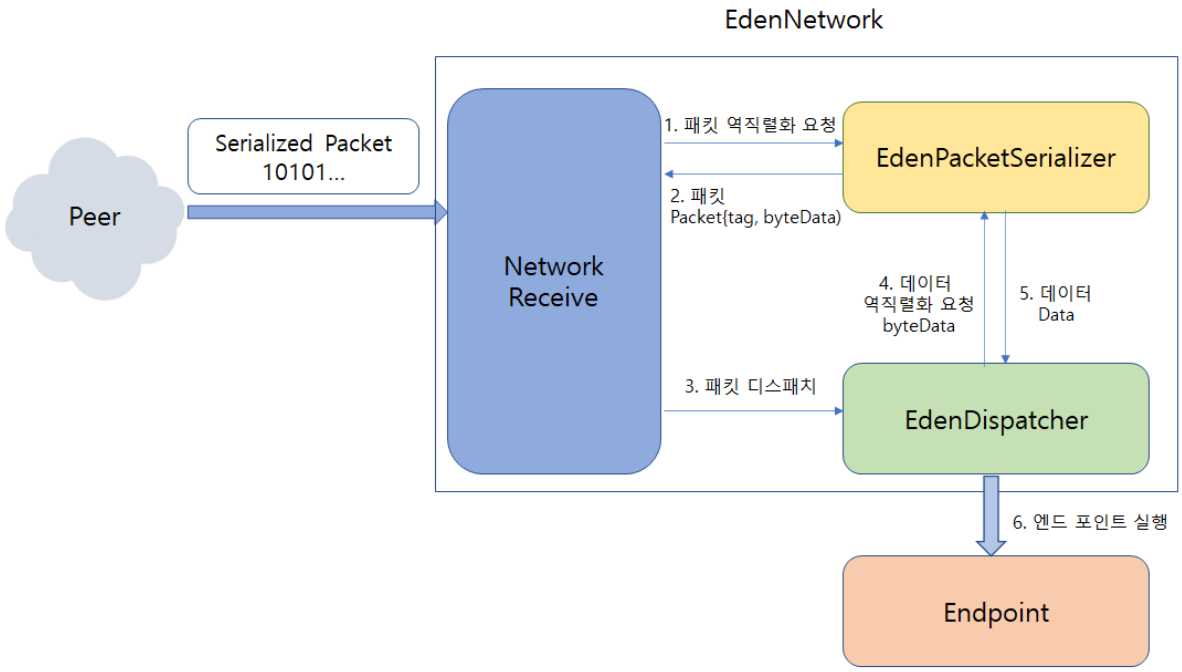

그림 4. EdenNetwork의 데이터 수신 방식

2. 주요 모듈의 구조와 기능

2.1. EdenNetwork

2.1.1. 통신방식

EdenNetwork에는 Send/Receive 방식의 통신과 Request/Response 방식의 통신 두가지가 존재합니다. Send와 Receive는 서버와 클라이언트 모두 사용할 수 있고 Request/Response의 경우 클라이언트는 Request만 서버는 Response만 사용합니다. Send에 경우 데이터를 송신 후 클라이언트는 자신의 다른 작업을 진행합니다. Request는 데이터를 송신 후 서버에서 요청에 대한 응답이 돌아올 때까지 기다리고 응답이 돌아오면 그 때서야 작업을 수행합니다.

네트워크에서 패킷 수신 시 형식에 맞지 않은 패킷일 경우 패킷을 Endpoint에 전달하지 않고 폐기합니다. 대신에 로그를 남겨 사용자가 확인할 수 있게 했습니다.

2.1.2. Endpoint

EdenNetwork는 수신 받은 패킷을 처리할Endpoint를 EdenDispatcher를 통해 등록할 수 있습니다. Endpoint는 각 API를 처리할메소드를 가진 클래스 인스턴스를 뜻합니다.

```
class Endpoint
{
    [EdenResponse]
    public int ServerResponse(PeerId clientId, int data)
    {
        ...
    }
```

```
    [EdenReceive]
    public void ServerReceive(PeerId clientId, int data)
    {
        ...
    }
}
EdenTcpServer server = new EdenTcpServer(12121);
server.AddEndpoint(new Endpoint(server));
```

위와 같은 예시 코드를 보면 좀 더 쉽게 Endpoint에 대해 이해할 수 있습니다. 패킷을 수신 받게 되면 Endpoint로 등록된 메소드들 중에 패킷의 tag와 같은 이름을 가진 메소드 이름을 찾아 해당 메소드를 실행합니다. 서버에서는 Endpoint 메소드의 첫 번째 매개변수는 패킷을 보낸 클라이언트의 ID값이고 두번째 값에 클라이언트가 보낸 데이터를 받게 됩니다. 클라이언트에서는 통신상대가 서버 뿐이니 첫번째 매개변수에 데이터가 오게 됩니다. 또한 Attribute를 통해 Receive인지 Response인지 구분하게 됩니다. Attribute가 있는 메소드만이 Endpoint로 등록됩니다.

2.1.3. TCP/UDP

EdenNetwork는 TCP와 UDP 두 프로토콜을 모두 지원합니다. TCP는 .NET의 기본 Socket 라이브러리를 이용해 구현했습니다. UDP의 경우 데이터 송수신의 신뢰성을 위해 Reliable UDP를 구현한 LiteNetLib(https://github.com/RevenantX/LiteNetLib)을 이용해 통신을 구현했습니다. TCP와 UDP 구현체 모두 같은 인터페이스를 상속받아 동일한 기능을 가지고 있습니다. UDP 구현체는 추가적으로 NAT Hole Punching을 할 수 있는 기능이 있습니다.

2.1.4 로그

로그는 Json 형식으로 로그를 작성합니다. 로그는 패킷의 송신이나 수신 시점에만 남기게 됩니다. 어떤 데이터가 송수신 되었는지 확인할 수 있고 잘못된 패킷을 수신 받았는지 확인할 수 있습니다.

2.2. EdenDispatcher

EdenDispatcher는 패킷 수신 시 Endpoint로 전달하는 역할을 가진 모듈입니다. 사용자가 Endpoint를 등록하면 EdenDispatcher가 Endpoint 정보를 가지고 있다 패킷을 수신했을 때 Endpoint 메소드를 실행합니다

2.2.1. Endpoint 등록

Endpoint에 객체를 등록하면 EdenDispatcher는 객체 내부의 메소드를 모두 스캔하여 부합하는 Attribute를 가진 메소드가 있다면 검증을 한 후 EdenDispatcher에 등록합니다. 검증 방식은 미리 정의해둔 메소드와 매개변수 개수와 반환 값이 일치하는 지 확인하는 것입니다.

각각의 Attribute당 검증 메소드는 다음과 같습니다. (Any에 경우 어떤 타입이 와도 상관없다는 뜻입니다.)

ServerDispatcher Attributes

Attribute 이름	검증 메소드	설명
EdenReceiveAttribute	void Receive(PeerId,Any)	Receive 메소드
EdenResponseAttribute	Any Response(PeerId, Any)	Response 메소드
EdenClientConnectAttribute	void ClientConnect(PeerId)	클라이언트가 서버에 접속했을 때 실행하는 메소드

EdenClientDisconnectAttribute	void ClientDisconnect(PeerId)	클라이언트가 서버와 연결이 끊겼을 때 실행하는 메소드

ClientDispatcher Attributes

Attribute 이름	검증 메소드	설명
EdenReceiveAttribute	void Receive(Any)	Receive 메소드
EdenDisconnectAttribute	void Disconnect()	서버와 연결이 끊겼을 때 실행하는 메소드

2.2.2. 패킷 디스패치

패킷이 네트워크를 통해 수신되면 Tag가 역직렬화 된 패킷을 EdenDispatcher로 넘겨줍니다. Tag를 통해 EdenDispatcher에 저장된 Endpoint를 찾고Endpoint 메소드의 매개변수 타입을 확인합니다. 그 후 매개변수 타입으로 Data를 역직렬화 하여 Endpoint의 매개변수로 실행합니다.

Packet 역직렬화를 두 단계로 나눈 이유는 패킷의 역직렬화를 EdenNetwork에서 처리하기 위해서 입니다. 일반적인 네트워크 프로그래밍에서 데이터 송수신 때마다 직렬화/역직렬화를 하는 것은 필수적인 작업이고 이 작업을 사용자 코드 레벨에서 하는 것은 코드를 길게 만들고 직관성을 떨어뜨립니다. 그렇기에 직렬화를 라이브러리에서 처리하기로 결정했습니다. 그렇기에 데이터를 역직렬화 하는 두번째 단계가 추가적으로 필요합니다.

2.3. EdenPacketSerializer

2.3.1. EdenPacket

```
public enum EdenPacketType : byte
{
    Send, Request, Response
}

public class EdenPacket
{
    public EdenPacketType Type { get; set; }
    public string Tag { get; set; }
    public object? Data { get; set; }
}
```

EdenPacket은 위와 같이 정의되어 있습니다. EdenPacketType을 통해 패킷의 종류를 구분하고 Tag를 통해 Endpoint를 찾습니다.

2.3.2 PacketSerialize

EdenPacketSerializer는 아래와 같이 패킷을 직렬화합니다.

Serialized Packet Frame

PacketLength(2B)	TagLength(1B)	Type(1B)	Tag(variable length)	Data(variable length)

패킷은 전체 길이를 2Bytes로 표현하고 Tag길이를 1Byte, Type을 1Byte로 표현하여 총 4Bytes의 패킷 헤더를 가지고 있습니다. Tag는 string 데이터를 UTF8로 인코딩하고 Data는 EdenDataSerializer를 이용해 직렬화 한 뒤 차례대로 붙여 직렬화 한 바이트 배열을 생성합니다.

역직렬화 시에도 4Bytes 헤더를 먼저 읽어 Tag와 Data의 위치를 확인한 뒤 차례대로 역직렬화합니다.

2.3.3 EdenDataSerializer

EdenDataSerializer는 사용자가 보낼 데이터를 직렬화 하는 클래스입니다. 기본적으로는 Google Protocol Buffer 직렬화 라이브러리인 Protobuf-net(https://github.com/protobuf-net/protobuf-net)을 이용해 직렬화 합니다. 사용자는 IEdenDataSerializer를 상속받은 클래스를 구현한 뒤 자신의 커스텀 직렬화 기법을 등록할 수 있습니다.

2. 통신 시퀀스

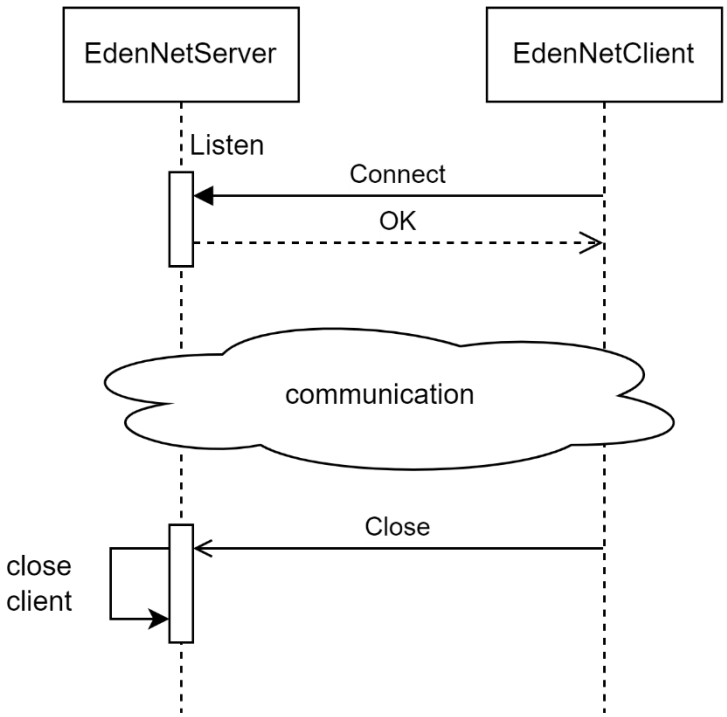

그림 5. EdenNetwork 통신 시퀀스

통신절차는 서버가 Listen을 시작한 후 클라이언트가 Connect로 서버에 접근합니다. 서버가 커넥트에 대해 OK를 반환하면 서버와 클라이언트간 통신을 수행한 후 클라이언트에서 Close를 보내면 서버는 해당 클라이언트를 닫게 됩니다.

2.1. Listen & Connect

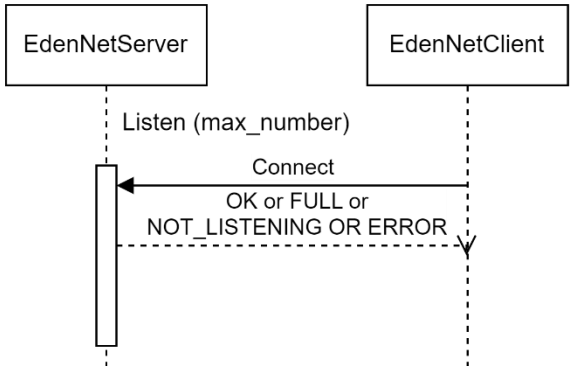

그림 6. EdenNetwork Listen & Connect 통신 시퀀스

서버는 Listen을 수행할 때 최대 몇 개까지의 클라이언트의 접속을 허용할 건지 max_number를 결정하며 시작합니다. Listen중인 서버에 클라이언트는 Connect를 통해 접속할 수 있습니다. Connect의 결과로 ConnectState를 반환합니다. ConnectState는 enum형으로 OK, FULL, NOT_LISTENING, ERROR가 존재합니다. OK를 반환받으면 서버접속에 성공한 것입니다. FULL을 반환받으면 서버에 접속된 클라이언트 수가 이미 최대이기에 더 이상 클라이언트가 접속할 수 없다는 의미입니다. NOT_LISTENING은 현재 서버가 Listen중이 아니라는 의미입니다. 그 외 서버가 Listen중인데 연결에 실패할 경우 ERROR를 반환합니다.

2.2. Send & Receive

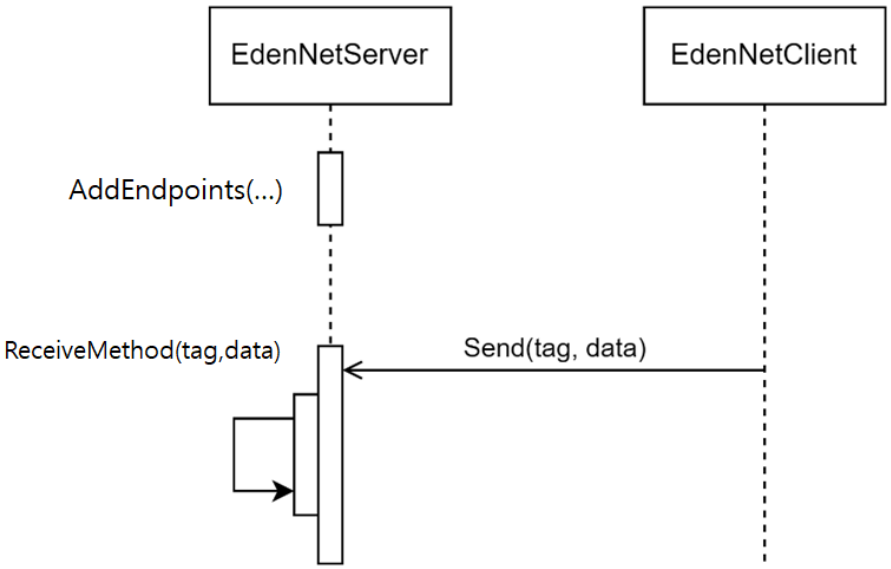

그림 7. EdenNetwork Send&Receive 통신 시퀀스

서버에 접속을 성공한 후 서버와 클라이언트는 서로 데이터를 주고 받을 수 있습니다. Send 메소드를 통해 데이터를 보낼 경우 tag와 함께 데이터를 보냅니다. 서버에서는 AddEndpoints 메소드를 통해 미리 tag를 처리할 콜백메소드를 등록합니다. 서버에서는 tag가 붙은 데이터가 소켓을 통해 들어오면 해당 tag에 매핑

되어 있는 메소드를 실행하여 데이터를 처리합니다. Send는 클라이언트에서 서버로 보낼 수 있으며 서버 또한 클라이언트에게 Send를 통해 데이터를 전송할 수 있습니다.

2.3. Request & Response

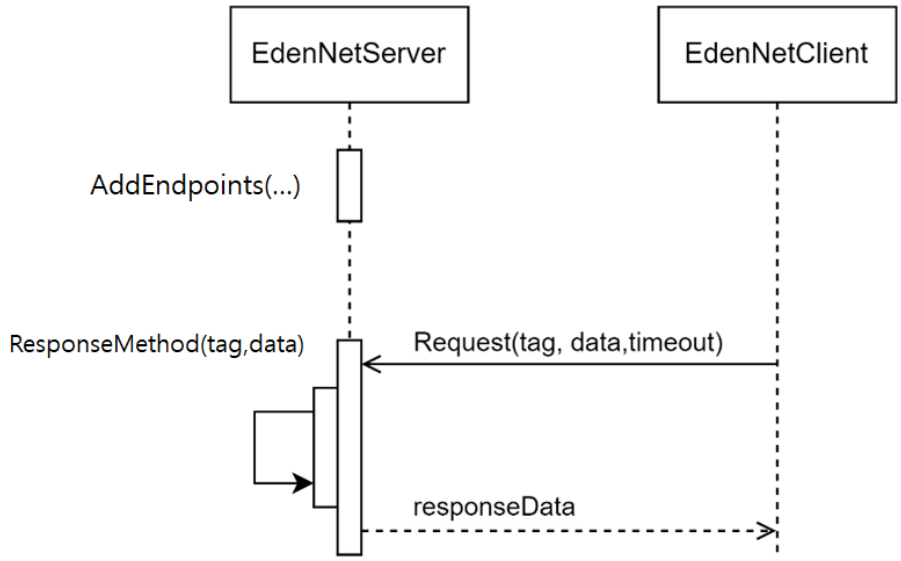

그림 8. EdenNetwork Request&Response 통신 시퀀스

Request 메소드를 통해 클라이언트가 서버에 데이터를 보낼 경우도 Send와 마찬가지로 tag와 함께 데이터를 보냅니다. Request는 서버에서 데이터를 반환하기 때문에 서버의 응답을 기다리는 시간인 timeout을 설정합니다. Request를 요청 후 서버에서 데이터를 반환할 때까지 프로세스를 블락하고 만일 timeout내에 서버가 응답을 돌려주지 않는다면 Error를 반환하며 프로세스 블락을 해제합니다. 서버에서는 AddEndpoints를 통해 tag를 처리할 콜백메소드를 등록합니다. 이 콜백메소드는 항상 클라이언트에게 응답할 데이터를 반환해야 합니다.