P2P Network Basics 🌐

The `P2PNet` library provides the core functionality for building and managing a peer-to-peer network. It includes classes and methods for peer discovery, connection management, data exchange, and network routines. This document provides a broad overview of the peer network basics.

Initialization

The `PeerNetwork` class is the main entry point for initializing and managing the peer-to-peer network. It sets up the necessary configurations and services required for network operations.

1. **Configuration**: The application reads configuration settings from the provided parameters. This includes settings for designated ports, inbound peer acceptance, and trust policies.
2. **Logging**: Logging is configured to use a plain text format and is activated to capture important events and errors.
3. **Local Address Loading**: The application scans all network interface devices and collects essential information needed for the peer network, such as public IP addresses.

Operation

The `PeerNetwork` class operates by providing several key functionalities:

1. **Peer Discovery and Connection**: The peer network supports both LAN and WAN peer discovery. It uses broadcasting and multicasting to discover peers within the network and establishes connections with them.

   - **LAN Discovery**: The application can broadcast and discover peers within the local network.
   - **WAN Discovery**: The application can discover peers over the wide area network using designated ports and WAN components.
2. **Peer Management**: The peer network manages a list of known peers and active peer channels. It supports adding, removing, and elevating peer permissions.

   - **Known Peers**: A list of peers that have been discovered and are known to the network.
   - **Active Peer Channels**: A list of active peer channels that are currently connected and exchanging data.
3. **Data Exchange**: The peer network supports data exchange between peers using peer channels. It handles sending and receiving data, as well as processing incoming packets.

   - **Peer Channels**: Represents a communication channel between two peers. It handles the sending and receiving of data packets.

Routines

The `NetworkRoutines` class provides a mechanism for managing network routines. Routines are tasks that run at specified intervals to perform various network-related operations.

1. **Routine Management**: The `NetworkRoutines` class manages a dictionary of routines and provides methods for adding, starting, stopping, and setting the interval of routines.
    - **Default Routines**: The application initializes with default routines, such as rotating broadcast ports and discerning peer channels.
    - **Custom Routines**: Users can add custom routines to perform specific tasks.

Routines are accessed using their `RoutineName` property. This is automatically handled when they are added as network routines.

Trust Policies

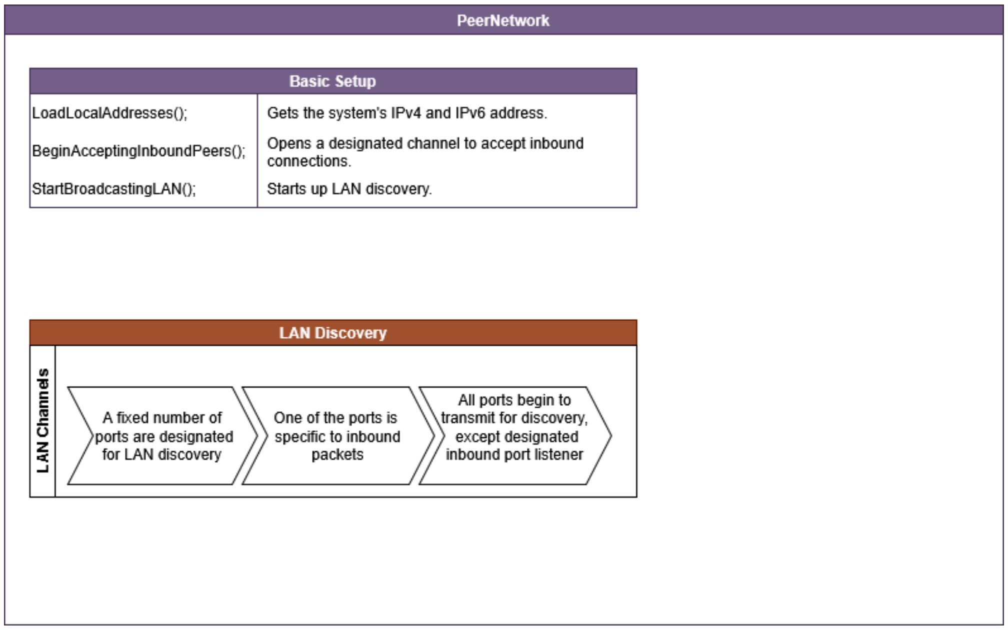
The `PeerNetwork` class includes trust policies for managing incoming peer connections and bootstrap connections.

1. **Incoming Peer Trust Policy**: Handles trust and permissions for incoming peer connections. It supports different modes for handling inbound peers, such as queue-based and event-based.
2. **Bootstrap Trust Policy**: Handles trust and permissions for bootstrap connections. It supports different trust policies, such as trustless and authority-based connections.

Diagrams

To supplement the information visually, the following diagrams are provided:

1. **Peer Network Architecture**: Shows the overall architecture of the peer network, including its interaction with peers and network routines.
2. **Peer Discovery Flow**: Illustrates the flow of peer discovery, from broadcasting to establishing connections and managing peer channels.


PeerNetwork
LAN Discovery
LAN Channels
A fixed number of ports are designated for LAN discovery
One of the ports is specific to inbound packets
All ports begin to transmit for discovery, except designated inbound port listener

Basic Setup

LoadLocalAddresses();	Gets the system's IPv4 and IPv6 address.
BeginAcceptingInboundPeers();	Opens a designated channel to accept inbound connections.
StartBroadcastingLAN();	Starts up LAN discovery.

subject to change

Peers

Peers are representative of client users within the peer network. When a connection is established to a peer through the TCP listener, the connection by default is wrapped in an instance of the `GenericPeer` implementation of `IPeer` and the `PeerChannel` which stores an `IPeer` implementation.

The `IPeer` interface defines the essential properties and methods for a peer, including the IP address, port, TCP client, network stream, and a unique identifier. The `GenericPeer` class provides a default implementation of the `IPeer` interface, encapsulating the peer's IP address, port, TCP client, and network stream. It also includes a unique identifier for the peer, which can be used for whitelisting and blacklisting peers in the network.

The `PeerChannel` class represents a communication channel with a peer in the P2P network. It manages the sending and receiving of data packets, handles connection retries, and maintains the state of the communication channel. The `PeerChannel` class also includes methods for opening and closing the channel, as well as handling incoming and outgoing data packets. The `PacketHandleProtocol` class stores the Action delegates for each packet type, and the `PeerChannel` will invoke these by default depending on the respective data packet type.

Peer Lifecycle

The lifecycle of a peer in the P2P network begins with the discovery and connection phase. When a new peer is discovered, the handler checks if the peer is already known or queued. If the peer is new, it is wrapped in an instance of the `GenericPeer` class and, depending on the `IncomingPeerTrustPolicy.IncomingPeerTrustPolicy` value, will be either enqueued and/or passed in the event `OnIncomingPeerConnectionAttempt` and finally will be added to the `KnownPeers` list.

Once the connection is established, a `PeerChannel` is created to manage the communication with the peer. The `PeerChannel` handles the sending and receiving of data packets, connection retries, and maintains the state of the communication channel. The `PeerChannel` will then be added to the `ActivePeerChannels` list. The `PeerChannel` also invokes the appropriate Action delegates from the `PacketHandleProtocol` class based on the type of data packet received. This ensures that the correct actions are taken for each type of data packet, facilitating efficient and reliable communication between peers.

There are some trust policies under `IncomingPeerTrustPolicy` that can slighly modify the initial behavior of the established `PeerChannel`

1. `AllowDefaultCommunication` - as the name implies, allows default communication between peers to exchange `PureMessagePackets`. Default is **true**.

2. `AllowEnhancedPacketExchange` - determines if peers will be trusted to exchange all other packet types, such as `DataTransmissionPackets`, which contain binary data such as files and network-related tasks. Default is **false**.

Peer Lifecycle

Bootstrap 🤝

The application serves as a bootstrap node, providing an HTTP endpoint to distribute known peers to new peers joining the network. This setup ensures seamless peer discovery and initialization, enabling efficient and secure distributed data exchange within the peer network. By containerizing the application using Docker, deployment becomes significantly easier and makes quick VPS deployments easy. Additionally, the user control panel offers finer-grained controls over the network, including scaling and monitoring, which enhances the manageability and reliability of the peer network infrastructure.

Initialization

The `P2PBootstrap` project initializes by setting up the necessary configurations and services required for the bootstrap server to operate. The main entry point is the `Program.cs` file, which configures the application and starts the web server.

1. **Configuration**: The application reads configuration settings from the `appsettings.json` file. This includes settings for encryption keys, database paths, and other essential configurations.
2. **Logging**: Logging is configured to use a plain text format and is activated to capture important events and errors.
3. **Web Server Setup**: The application uses ASP.NET Core to set up the web server. It configures the HTTP request pipeline, enabling default files, static files, and routing. This is an AOT compatible application, as is most of the P2PNet library.

Operation

The `P2PBootstrap` project operates by providing several key functionalities:

1. **Peer Distribution**: The bootstrap server provides an HTTP endpoint (`/api/Bootstrap/peers`) to distribute the list of known peers to new peers joining the network. This endpoint can operate in two modes:
   - **Trustless Mode**: Returns the list of known peers directly.
   - **Authority Mode**: Requires the client to receive and store a public key from the bootstrap server before returning the peer list.
2. **Parser Integration**: The server integrates with a parser to handle input and output operations. It provides endpoints to get parser output (`/api/parser/output`) and to submit parser input (`/api/parser/input`).

3. **Encryption Service**: The server initializes an encryption service to handle secure communication. This includes generating and loading PGP keys from the specified directory.

4. **Database Initialization**: The server initializes a local database to store necessary data. It ensures the database directory exists and sets up the required files.

User Control Panel

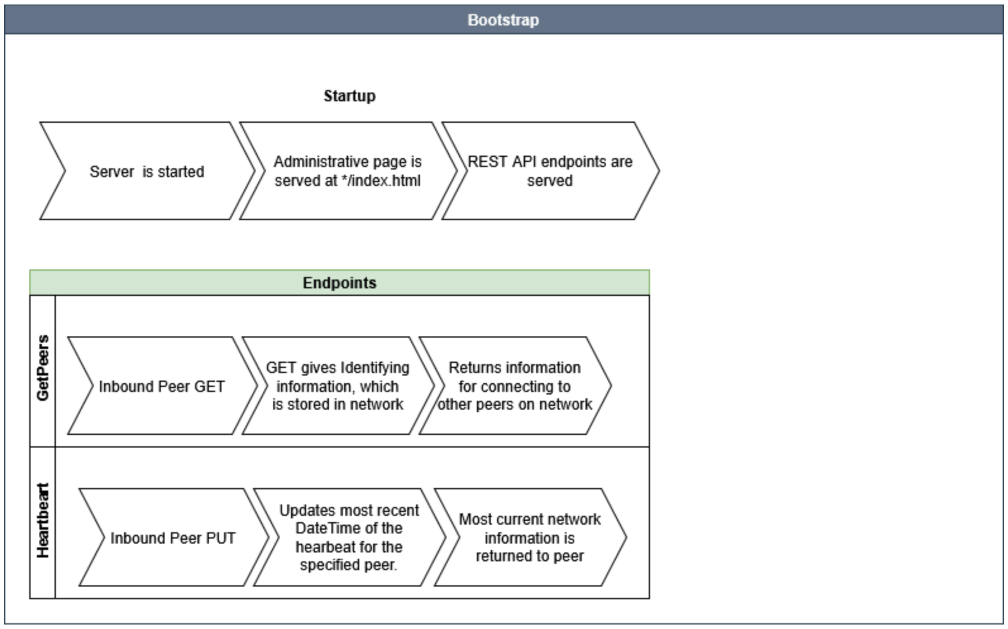
The user control panel provides a web-based interface for managing the bootstrap server. In this web-based interface is a terminal for executing commands. It includes the following pages for easier management as well:

1. **Overview**: Displays an overview of the server's status and key metrics.
2. **Settings**: Allows users to modify server settings, such as encryption keys and database paths.
3. **Peers**: Displays the list of known peers and provides options to perform actions on them, such as disconnecting or blocking peers.

Diagrams

To supplement the information visually, the following diagrams are provided:

1. **Bootstrap Server Architecture**: Shows the overall architecture of the bootstrap server, including its interaction with the P2P network and the user control panel.
2. **Peer Distribution Flow**: Illustrates the flow of peer distribution, from a new peer requesting the peer list to the server returning the list based on the configured trust policy.


Bootstrap
Startup
Server is started
Administrative page is served at */index.html
REST API endpoints are served
Endpoints
GetPeers
Inbound Peer GET
GET gives Identifying information, which is stored in network
Returns information for connecting to other peers on network
Heartbeart
Inbound Peer PUT
Updates most recent DateTime of the hearbeat for the specified peer.
Most current network information is returned to peer

Note: Bootstrap server still under construction 🏗️

Widescan 📡

The `P2PNet.Widescan` class library project is designed to facilitate mass IPv6 address generation and peer discovery within a peer-to-peer network. This project leverages hardware capabilities, such as GPUs, to efficiently generate vast quantities of IPv6 addresses. By utilizing user-defined address prefixes, it allows for a more targeted and narrow scope of addresses to ping, enhancing the efficiency of the discovery process. This can be leveraged with publicly available information on IPv6 prefix registrations, like the Ripe database, in order to refine the scope of the widescan. Additionally, the project employs lightweight ICMP packets to broadcast discovery information, ensuring minimal network overhead while effectively communicating with potential peers.

Initialization

The `P2PNet.Widescan` project initializes by setting up the necessary configurations and services required for widescan operations. The main entry point is the `Widescan` class, which configures the application and starts the widescan process.

1. **Configuration**: The application reads configuration settings from the provided parameters. This includes settings for address prefixes, hardware mode (GPU or CPU), and other essential configurations.
2. **Logging**: Logging is configured to use a plain text format and is activated to capture important events and errors.
3. **Hardware Mode Setup**: The application can be configured to use either GPU offloading or parallel CPU capabilities for address generation and peer discovery.

Operation

The `P2PNet.Widescan` project operates by providing several key functionalities:

1. **IPv6 Address Generation**: The widescan application generates vast quantities of IPv6 addresses using either GPU offloading or parallel CPU capabilities. This allows for efficient and rapid address generation.

    - **GPU Offloading**: Utilizes the processing power of GPUs to generate IPv6 addresses in parallel, significantly speeding up the process.
    - **Parallel CPU Capabilities**: Uses multiple CPU cores to generate IPv6 addresses in parallel, providing an alternative for systems without GPU support.
2. **Peer Discovery**: The widescan application pings the generated IPv6 addresses to discover potential peers within the network. It uses lightweight ICMP packets to broadcast discovery information, ensuring minimal network overhead.

3. **Address Prefix Filtering**: By utilizing user-defined address prefixes, the widescan application narrows the scope of addresses to ping, enhancing the efficiency of the discovery process. This can be refined using publicly available information on IPv6 prefix registrations, such as the Ripe database.

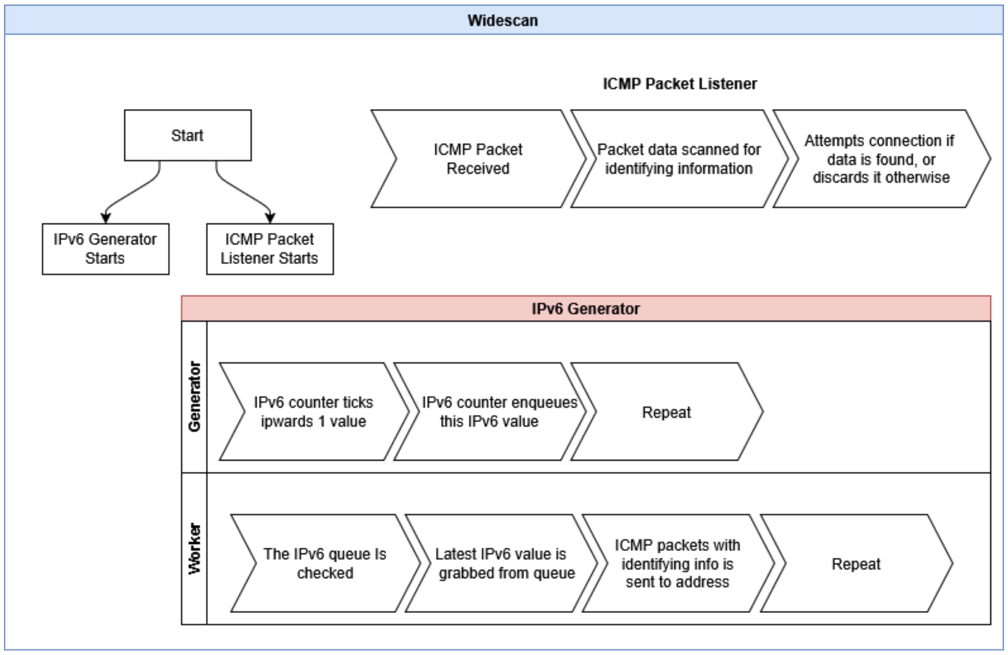
Integration

The `P2PNet.Widescan` project is designed to be a modular import that can be integrated with other P2P network applications, such as a`P2PNetwork` client application or the `P2PBootstrap` server. It does not run independently and does not use `appsettings.json` for configuration. Instead, it relies on the host application to provide the necessary configuration parameters.

Diagrams

To supplement the information visually, the following diagrams are provided:

1. **Widescan Architecture**: Shows the overall architecture of the widescan application, including its interaction with the P2P network and the hardware components (GPU/CPU).
2. **IPv6 Address Generation Flow**: Illustrates the flow of IPv6 address generation, from configuration to address generation using GPU or CPU, and finally to peer discovery. This is very much a simple consumer-producer pattern, with intermediate in-memory queues to operate as a broadcaster.
3. **ICMP Packet Listener:** Operating independently of the broadcaster, the ICMP Packet Listener operates as a listener for responses from prospective peers and other potential widescan instances. This utilizes a docile form of packet sniffing on the host machine.


Widescan
ICMP Packet Listener
Start
IPv6 Generator Starts
ICMP Packet Listener Starts
ICMP Packet Received
Packet data scanned for identifying information
Attempts connection if data is found, or discards it otherwise
IPv6 Generator
Generator
IPv6 counter ticks ipwards 1 value
IPv6 counter enqueues this IPv6 value
Repeat
Worker
The IPv6 queue Is checked
Latest IPv6 value is grabbed from queue
ICMP packets with identifying info is sent to address
Repeat

Note: Widescan project still under construction 🏗️