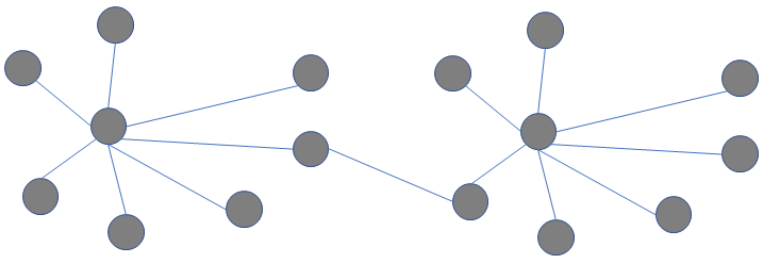
Interblockchain Dynamic Network (Version 2.8.55)

Introduction

The NTR internodes network is a dynamic peer to peer network based on the Kamdelia protocol. Each network node has a preset number of linked nodes. So, each node is to be seen as a star connected in which the node is connected to its peers neighbours (see figure below). For example, a node may have a limit of 7 peers.

- A "star" node can be connected to another "star" node.
- Each node maintains its local routing table.
- Sending a message to a particular node is performed through an algorithm calculating the shortest path between the source and destination node.

At the moment, the only supported messenging method is the `broadcast` function. Hence, each node can only broadcast a message to all nodes.



WARNING: Do not run any software using the internodes library on a computer also running an Ethereum server. This disrupts the DPT algorithm.

This library bundles different components for lower-level peer-to-peer connection and message exchange:

- Distributed Peer Table (DPT) / Node Discovery
- RLPx Transport Protocol
- Interblockchain Wire protocol

The library is based on ethereumjs/node-devp2p as well as other sub-libraries (`node-*` named).

Using the library

The first step is to include the library into your project with:

```
npm install ntrnetwork
```

Then to import the ntrnetwork class and instantiate it. To receive the network transmitted transactions, you need to subscribe a callback which will receive as parameter network transactions data.
To be added to your code:

```
const Broadcaster = require('../src/index').Broadcaster;
const broadcaster = new Broadcaster(0x016);
broadcaster.subscribe(myCallback);
myCallback(message_code, transaction) {
  // do something with the transaction
}
```

To publish a message to all peers:

```
broadcaster.publish(message_code, data);
```

Note: In the latest release, the boadcaster class has a new constrauctor requiring to specify a channel ID expressed as an hexadecimal number. The following numbers are suggested: Testnet1: 0x016; testAlpha: 0x015; production: 0x014;

example: const broadcaster = new Broadcaster(0x016);

Two network monitors are not installed respectively on 138.197.167.83 for testnet and on 138.197.156.204 for alpha testing. The monitor's status is accessed through the port 9099 on both monitors.

Message types

Note: when the data content is modified, the file src/index.js should be modified to properly catch the transactionID.

The following message types are defined in the interblockchain/index.js file

- STATUS: 0x00: each node sends a status to message to synchronize each other.
- TX: 0x02: Transaction message. This, is, restricted to the transfer request documents.
- AUDIT: 0x03: auditors send this message as the result of an audit.

These message types are handled in the Interblockchain layer - NTR class (interblockchain/index.html). The lower level messages are handled in the DPT level - the server class

- PING
- PONG
- FINDNEIGHBOURS
- NEIGHBOURS

Environment Variables

- MAXPEERS: number (maximum number of peers for a node)
- MONITOR: boolean (log output setting)

- BROADCASTER: boolean (log output setting)
- RLPX: boolean (log output setting)
- INTERBLOCKCHAIN:boolean (log output setting)
- SERVER: boolean (log output setting)
- DPT: boolean (log output setting)
- PEERS: boolean (log output setting)

TX message example

```
{
  "nodeID":"interblockchain-11b6f2b9-e7e3-42a9hdgg",
  "ip":"127.65.34.298",
  "transactionID": "12345678987654321",
  "appID": "12345678900000",
  "ticker": "itBTC",
  "sourceNetwork": "Ethereum Network",
  "sourceAddress": "0x4327DDebc9f86cb7dd8e2b899203457a2b3aef91",
  "from": "0xF4c049517e3f9c61e846887f49fb52a7f2f271ce",
  "destinationNetwork": "Bitcoin Network",
  "destinationAddress": "n33npN2oRNJFFpdoxkrm2CVyZAfRGFxALt",
  "tokenContractAddress": "0xb398cebdc41d2935a438659da3f0b01fb583f339",
  "amount": "0.5"
}
```

AUDIT message example

Note: the following properties: *transactionID*, *nodeID* and *ip* are all automatically added by the network library. Do not include them into a broadcasted message.

```
{
  "transactionID": "11b6f2b9-e7e3-42a9hdgg",
  "nodeID":"interblockchain-11b6f2b9-e7e3-42a9hdgg",
  "ip":"127.65.34.298",
  status: true,
  TR: {
    "timestamp": "Wed Oct 31 2018 12:37:17",
    "sourceKey":
"TETH:0x413e71dc2f5ad1b87967a5e01f604d3609d345a4:0x130f6562d441d25a812865527761ee7
246af0297:10000",
    "destKey": "TBCH:2MwSakwx2sy7mXUhmGeE3dKTmhYgNvkzwcR:0:10000",
    "transferRequest": {
        "amount": "0.0001",
        "appID": "382b494b-15ce-4014-8710-d9ca8060b67b",
        "destinationAddress": "2MwSakwx2sy7mXUhmGeE3dKTmhYgNvkzwcR",
        "destinationNetwork": "TBCH",
        "from": "0x130f6562d441D25A812865527761EE7246AF0297",
        "sourceAddress": "0x413e71dc2f5ad1b87967a5e01f604d3609d345a4",
        "sourceNetwork": "TETH",
        "ticker": "ITBCH",
```

```
        "tokenContractAddress": "0xd1e7560e4b9c6ab0facb450446fbf64c6bd8490a",
        "transactionID": "b4227873-e7e9-4375-b36f-3d1b010280ef",
        "brdcTender": true,
        "onlyReqConf": true
    }
}
```

REST API of the monitoring node

To get all connected peers to a particular node:

```
GET nodeDomain/peers
```

When a transaction is received by a node, it is recorded in memory and kept for 30 seconds. This is preventing a node to broadcast a transaction already received. When the 30 second is elapsed, the garbagge collector removes all transactions older than 30 seconds. The collection is transactions kept in memory is obtained by:

```
GET nodeDomain/tx
```

Run/Build

This library needs to run in an **ECMAScript 2016** and **nodeJS version 8.11.2** and up.

Steps:

- clone the source code into your project with

```
git clone https://github.com/Interblockchain/internodes.git
```

- install dependencies registered in the `package.json` file located in the root directory containing the Interblockchain wire protocol.

```
npm install
```

- Include the dependencies into your project (as illustrated in the `peer-communication.interblockchain.js`)

```
const devp2p = require('../src');
const LRUCache = require('lru-cache');
const ms = require('ms');
const assert = require('assert');
const { randomBytes } = require('crypto');
```

```
const rlp = require('rlp-encoding');
const Buffer = require('safe-buffer').Buffer;
const mac = require('getmac');
```

Publish the library on NPM

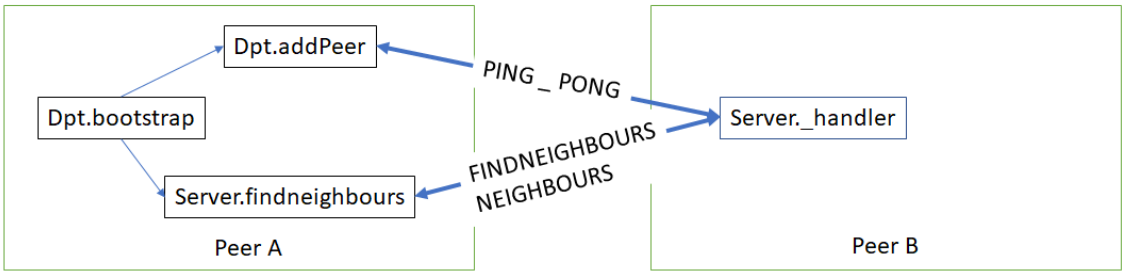
- Create an NPM account
- Login (*npm login*)
- Verify your npm local subscription (*npm whoami*)
- publish (_npm publish)

Publish the network monitor as a Docker

- Build the image (*docker build --rm -f "Dockerfile" -t interblockchainlab/internodes:latest .*)
- Publish the image on DockerHub (*docker push interblockchainlab/internodes:latest*)

Programming notes:

The main interfunction communicaiton involved in the handshake during the bootstrap stage and node discovery is illutrated below:



Usage/Examples

An example implementation is included in the `test` directory, in the file: `peer-communication.interblockchain.js`

Classes

All classes of this library are implemented as Node `EventEmitter` objects and make heavy use of the Node.js network stack.

You can react on events from the network like this:

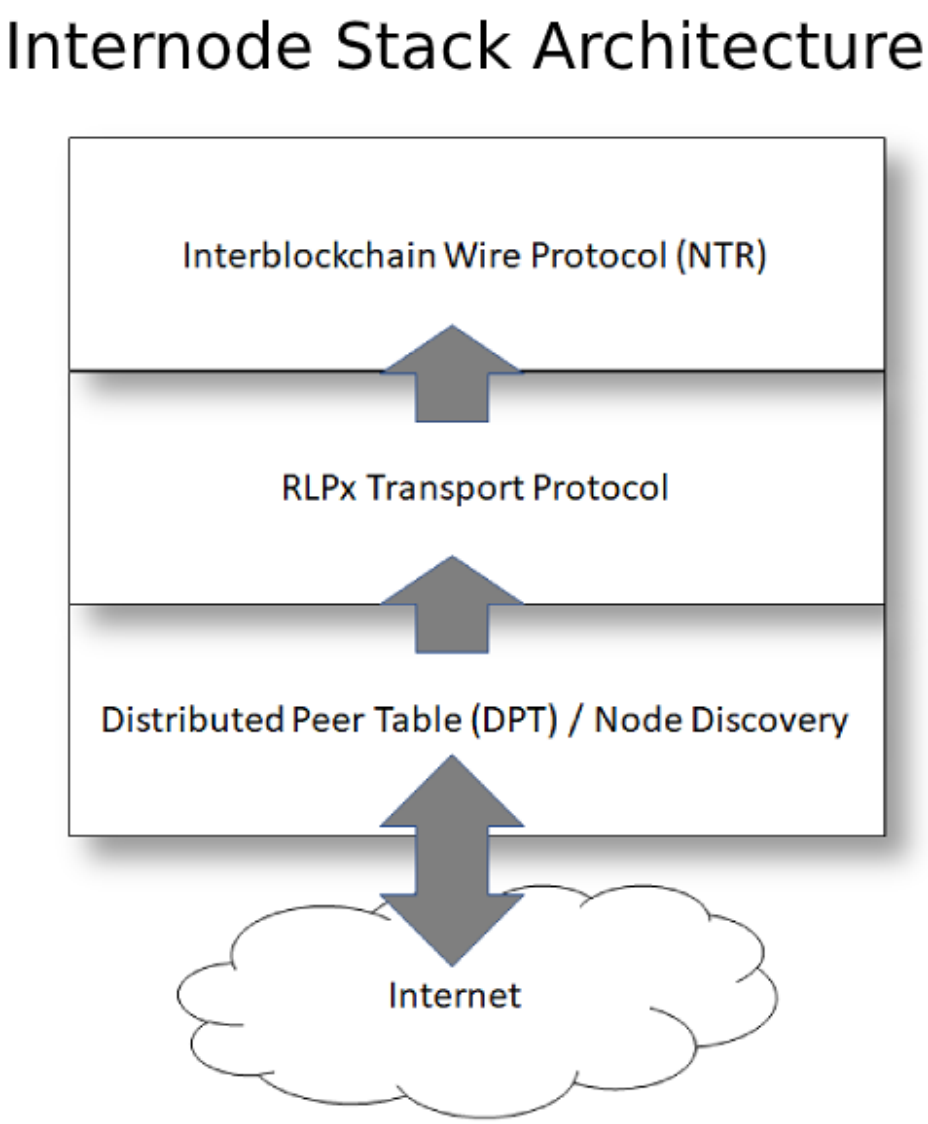
```
dpt.on('peer:added', (peer) => {
  // Do something...
})
```

Basic example to connect to some bootstrap nodes and get basic peer info:

- simple

Communicate with peers to read and write new transactions:

- peer-communication



Interblockchain Wire Protocol (NTR)

Is an upper layer protocol to transmit message among Interblockchain nodes, see ./src/interblockchain

Usage

When a new peer is added to the network, the `peer:added` event is fired. The event handler attached to this event receives as parameter the connected `peer` object. The first task of this event handler is to send a `status` object to the just connected peer. This is sent to the connected peer with the `sendStatus()` function as illustrated below.

```
rlpx.on('peer:added', (peer) => {
  ntr.sendStatus({
    networkId: CHAIN_ID,
    networkName: "interblockchain",
    Note:"test in progress"
  });
  // Do something with this messsage :-)
}
```

When other nodes send their own status message, the latter is trapped by an event handler associated to the `status` event. This event is fired only once per peer connected.

```
eth.once('status', () => {
  // Send an initial message
  ntr.sendMessage()
})
```

Each message received from other peers is handled by an event handler associated to the `message` event.

```
ntr.on('message', async (code, payload) => {
  if (code === devp2p.NTR.MESSAGE_CODES.TX) {
    // Do something with this messsage :-)
  }
})
```

API

Execution modes

The internodes can be executed as:

- an NPM library
- a nodejS application

When used as an NPM library the starting class is **Broadcaster** (located in /src/index.js). When used a nodeJS application the starting file is app.js, an express based REST endpoint.

`ntr.sendStatus(status)`

Send initial status message.

- `status` - Status message to send, format `{ networkId: CHAIN_ID}`.

`ntr.sendMessage(code, payload)`

Send a message.

- `code` - The message code, see `MESSAGE_CODES` for available message types.
- `payload` - Payload as a list, will be rlp-encoded.

Events

Events emitted:

Event	Description
message	Message received
status	Status info received

RLPx Transport Protocol

Connect to a peer, organize the communication, see ./src/rlpx/

Usage

Create your `RLPx` object, e.g.:

```
const rlpx = new devp2p.RLPx(PRIVATE_KEY, {
  dpt: dpt,
  maxPeers: 25,
  capabilities: [
    interblockchain
  ],
  listenPort: null
})
```

API

`RLPx` (extends `EventEmitter`)

Manages the handshake (`ECIES`) and the handling of the peer communication (`Peer`).

`new RLPx(privateKey, options)`

Creates new RLPx object

- `privateKey` - Key for message encoding/signing.
- `options.timeout` - Peer `ping` timeout in ms (default: `10s`).
- `options.maxPeers` - Max number of peer connections (default: `10`).
- `options.clientId` - Client ID string (default example: `ethereumjs-devp2p/v2.1.3/darwin-x64/nodejs`).
- `options.remoteClientIdFilter` - Optional list of client ID filter strings (e.g. `['go1.5', 'quorum']`).
- `options.capabilities` - Upper layer protocol capabilities, e.g. `[devp2p.ETH.eth63, devp2p.ETH.eth62]`.
- `options.listenPort` - The listening port for the server or `null` for default.

- `options.dpt` - `DPT` object for the peers to connect to (default: `null`, no `DPT` peer management).

`rlpx.connect(peer)` (`async`)

Manually connect to peer without `DPT`.

- `peer` - Peer to connect to, format `{ id: PEER_ID, address: PEER_ADDRESS, port: PEER_PORT }`.

`rplx.broadcast(code, message)`

Broadcast a message to all connected peers.

- `code` - The message code, see `MESSAGE_CODES` for available message types.
- `message` - can be an Array, a string

For other connection/utility functions like `listen`, `getPeers` see ./src/rlpx/index.js.

Events

Events emitted:

Event	Description
peer:added	Handshake with peer successful
peer:removed	Disconnected from peer
peer:error	Error connecting to peer
listening	Forwarded from server
close	Forwarded from server
error	Forwarded from server

Reference

- RLPx: Cryptographic Network & Transport Protocol
- devp2p wire protocol

Distributed Peer Table (DPT) / Node Discovery

Maintain/manage a list of peers, see ./src/dpt/, also includes node discovery (./src/dpt/server.js)

Usage

Create your peer table:

```
const dpt = new DPT(Buffer.from(PRIVATE_KEY, 'hex'), {
  endpoint: {
    address: '0.0.0.0',
    udpPort: null,
```

```
    tcpPort: null
  }
})
```

Add some bootstrap nodes (or some custom nodes with `dpt.addPeer()`):

```
dpt.bootstrap(bootnode).catch((err) => console.error('Something went wrong!'))
```

API

`DPT` (extends `EventEmitter`)

Distributed Peer Table. Manages a Kademlia DHT K-bucket (`Kbucket`) for storing peer information and a `BanList` for keeping a list of bad peers. `Server` implements the node discovery (`ping`, `pong`, `findNeighbours`).

`new DPT(privateKey, options)`

Creates new DPT object

- `privateKey` - Key for message encoding/signing.
- `options.refreshInterval` - Interval in ms for refreshing (calling `findNeighbours`) the peer list (default: `60s`).
- `options.createSocket` - A datagram (dgram) `createSocket` function, passed to `Server` (default: `dgram.createSocket.bind(null, 'udp4')`).
- `options.timeout` - Timeout in ms for server `ping`, passed to `Server` (default: `10s`).
- `options.endpoint` - Endpoint information to send with the server `ping`, passed to `Server` (default: `{ address: '0.0.0.0', udpPort: null, tcpPort: null }`).

`dpt.bootstrap(peer)` (`async`)

Uses a peer as new bootstrap peer and calls `findNeighbouts`.

- `peer` - Peer to be added, format `{ address: [ADDRESS], udpPort: [UDPPORT], tcpPort: [TCPPORT] }`.

`dpt.addPeer(object)` (`async`)

Adds a new peer.

- `object` - Peer to be added, format `{ address: [ADDRESS], udpPort: [UDPPORT], tcpPort: [TCPPORT] }`.

For other utility functions like `getPeer`, `getPeers` see ./src/dpt/index.js.

Events

Events emitted:

Event	Description
peer:added	Peer added to DHT bucket
peer:removed	Peer removed from DHT bucket
peer:new	New peer added
listening	Forwarded from server
close	Forwarded from server
error	Forwarded from server

Reference

- Node discovery protocol
- RLPx - Node Discovery Protocol
- Kademlia Peer Selection
- Safe Buffer

Todo

Add a list of connection nodes if the main connected node is off and if there not enough internodes connected.