Blockchain Foundations **NTUA, Sping 2026**

Problem Set #1 *

Soft Deadline: TBD
Final Deadline: TBD

Introduction

The problem sets will involve building our own blockchain, Marabu. As a student, you will implement your own Marabu full node from scratch. You are free to form groups of up to three students to work on the problems sets. Your node must connect to the nodes of other students and maintain a common blockchain with them. You will implement the protocol by following through the problem sets. The protocol is a simplified, educational protocol easy to debug and implement designed especially for this course. Even though it is simple, it implements a full blockchain which follows the same principles as a full blockchain such as Bitcoin and Ethereum, albeit with limited features (it allows only for simple payments and is not very efficient). The problem sets guide you through implementing your node in a step-by-step fashion, so each problem set builds on top of the previous one to augment your node slowly adding new features until it is complete. In case you miss one problem set or you are unable to implement the tasks in that problem set correctly, you may make use of the solutions of that problem set in order to continue with the next one.
For programming, you are free to use any programming language that you are proficient in but we recommend using TypeScript for the following reasons:

- TypeScript offers easy asynchronous programming which would be required for your node to simultaneously listen to multiple connections.
- Solutions will be provided in TypeScript.
- Course staff can only provide assistance during office hours in TypeScript. Assistance on other languages is not guaranteed.

Comment your source code well.

Submission Instructions

Please upload your code to a private Github repository. Add the course staff (`dionyziz`, `Scott-Hickmann` and `claserken`) as collaborators. You can then submit the link to your

*Version: 2 – Last update: July 1

Github repository, the commit hash corresponding to the version of your code you want to submit, and the IP address where your node is hosted on Gradescope. Please add all of your team members to the Gradescope submission to make sure everyone receives the grading results. You will need to repeat that both for the soft deadline (if you want to get ungraded feedback before the hard deadline) and for the hard deadline. This process will be the same for later programming PSETs.

1 Networking

Start coding the implementation of your Marabu node. In this exercise, you will set up your node and exchange a 'hello' message with any peer connecting to you.

1. Decide what programming language you will use. We recommend TypeScript but you are free to choose a language you are proficient in.

2. Find a nice name for your node (consisting of up to 128 printable ASCII characters).

3. Read the protocol description at [**?**].

4. Implement the networking core of your node. Your node must listen to TCP port 18018 for connections and must also be able to initiate TCP connections with other peers. For Javascript and Typescript, you may find the `"net"` module useful (see the documentation at [**?**]). Your node must be able to support connections to multiple nodes at the same time. You can check [**?**] and [**?**] for sample code to implement a TCP server and client on Javascript/TypeScript.

5. Implement canonical JSON encoding for messages as per the format specified in [**?**]. You are free to use library functions for this [**?**].

6. On receiving data from a connected node, decode and parse it as a JSON string. If the received message is not a valid JSON or doesn't parse into one of the valid message types, send an `"INVALID_FORMAT"` error to the node.

   Note that a single message may get split across different packets, e.g. you may receive `{"type":"ge` and `tpeers"}` in separate messages. So you should defragment such JSON strings. Alternatively, a single packet could contain multiple messages (seprated by `"\n"`) and your node should be able to separate them. Note that JSON strings you receive may not be in canonical form, but they are valid messages nevertheless.

7. Implement the protocol handshake

   a) When you connect to a node or another node connects to you, send a `"hello"` message with the specified format.

   b) If a connected node sends any other message of a valid format prior to the hello message, you must send an `"INVALID_HANDSHAKE"` error message to the node and then close the connection with that node. (However, if the message is not properly formatted, just send an `"INVALID_FORMAT"` error).

Note: Every message you send on the network should have a newline, i.e. `"\n"` at the end. Your node should use this to help parse and defragment valid messages.

2 Peer Discovery

In this exercise, you will extend your Marabu node so that it can exchange messages and perform peer discovery.

1. Hard-code a list of bootstrapping peers identified by their IP addresses. You can use the following addresses where the course staff have hosted their nodes:

   `[45.63.84.226:18018, 45.63.89.228:18018, 144.202.122.8:18018]`

2. Store a list of discovered peers (including the bootstrapping peers above) locally. The list should survive reboots.

3. On loading the Marabu node, connect to at least one of your discovered peers.

4. Upon connection with any peer (initiated either by your node or the peer), send a `"getpeers"` message immediately after the `"hello"` message.

5. On receiving a `"peers"` message from a connected peer, update your list of discovered peers.

6. Optional: Devise a policy to decide how many and which peers to connect to.

You can use the command `nc -vvv ip_addr port` to connect to a node at IP address `ip_addr` and port `port`, and also send and receive messages. Use this to test your node extensively. You can also use `nc -vvv -l -p port` to listen for connections on the port `port`.

3 Test Cases

IMPORTANT: Make sure that your node is running at all times! Therefore, make sure that there are no bugs that crash your node. If our automatic grading script can not connect to your node, you will not receive any credit. Taking enough time to test your node will help you ensure this.
Below is a (non-exhaustive) list of test cases that your node will be required to pass. We will also use these test cases to grade your submission.

1. The grader node "Grader" should be able to connect to your node. If you don't pass this test, Grader would not be able to grade the rest of the test cases. So make sure that you test this before you submit. Test this on the machine whose IP address you will submit.

2. Grader should receive a valid `hello` message on connecting.

3. The `hello` message should be followed by a `getpeers` message.

4. Grader should be able to disconnect, then connect to your node again.
5. If Grader sends a `getpeers` message, it must receive a valid `peers` message.
6. If Grader sends `{"type":ge`, waits for 0.1 second, then sends `tpeers"}`, your node should reply with a valid `peers` message.
7. If Grader sends any message before sending `hello`, your node should send an `"INVALID_HANDSHAKE"` error message in the format specified in the protocol description and then disconnect.
8. If Grader sends an invalid message, your node should send an `"INVALID_FORMAT"` error message. Some examples of invalid messages are:
   a) `Wbgygvf7rgtyv7tfbgy{{{`
   b) `{"type":"diufygeuybhv"}`
   c) `{"type":"hello"}`
   d) `{"type":"hello","version":"jd3.x"}`
   e) `{"type":"hello","version":"5.8.2"}`
9. If grader sends a set of peers in a valid `peers` message, disconnects, reconnects and sends a `getpeers` message, it must receive a `peers` message containing at least the peers sent in the first message.
10. Grader should be able to create two connections to your node simultaneously.