Raiden Train Demo

A model train is traveling the world (in our case it's just a circle but hey we all have fanatasy ;)). During its travels it passes multiple tollgates asking it to pay for the next section. The train however has a Raiden node on board and an open payment channel with it's home toll station. Since the tollgate mafia knows eachother they all have statechannels among them in such a ways as Displayed in Image 1
Image 1 - Network Topology
With the Raiden Network the train is now able to pay each and every tollage on it's travel in real time, without the need to stop at any point in time.

Setup on the "main" Computer

The "main computer" is a UDOO x84 Ultra. It has an embedded arduino 101 which is used to control the train's power supply. Therefore the arduino listens to the serial input and gives 3,3V to the GPIO pin 3. This switches the High speed switch on. As soon as the the 3,3V are set to 0V the switch turns off and the train doesn't get any power anymore. A circuit diagram like illustration can be found in Image 2 - Cicruit Diagram.
The UDOO runs a virtualenvironment with all dependencies provided in the requirements.txt as well as raiden installed via pip.

Setup on the Raspberry Pi Zero W

The Raspberypi has a Raiden Service in /etc/systemd/system/raiden.service.
It calls the shell script in ~/demo-train/StartServiceRaiden/StartService.sh during the bootup process.
The raspberry uses two virtualenvironments. One is the "raiden" virtualenv which is used by the shell script to run raiden. Ofcourse it requires raiden to be installed (In this instance done via PyPi: `pip install raiden`). The other virtualenv is called "demo" and runs the "sender_main.py" logic.

Blockchain Setup

Right now the demo uses the Kovan Testnet.
All accounts hold a certain balance of "Raiden Demo Tokens". The Raiden Demo Token as address is `0xDe99085F789f99568f891c6370fB6b7dD4C90323` and it is a minimalistic standard ERC20 Token. The channels as illustrated in Image 1 are all "unidirectional" with a balance of 1000 and a timeout of 500.

Processing Setup

Processing is used for the frontend visualisation part. All processing files are located in ./processing/sketchbook/ The main file is called railTrack.pde

Start processing without IDE

./processing-3.4/processing-java –sketch=/home/train/sketchbook/testLights –force –run

Be aware that just the folder is passed and not the actual pde-file.

Start on big screen via ssh

DISPLAY=:0.0 ./processing-3.4/processing-java –sketch=/home/train/sketchbook/testLights –force –run

There is a alias on the RPI for this which is called prorail.

Right now we work with a separate server mock, that provides an endpoint with random addresses. The endpoint will be modelled exactly like in a later production version (same endpoints, same JSON-data).

First, you need to install the python dependencies for the asynchronous server: It requires Python 3.7 (!Check if this is a problem with raiden!).

```
pip install --r requirements.txt
```

The http library for processing needs to be put in the processing library folder:

https://github.com/runemadsen/HTTP-Requests-for-Processing/releases

Then the server has to be started:

```
hypercorn server_mock:app
```

Now run the `processing/processing.pde` in Processing 3

Run the demo

Udoo

You can login to the udoo via ssh `train@demo` if you're in the bbot wifi, the password is `raiden`.
The ~/.bash.rc is configure to autmoatically activate the required virtualenv and navigate to the demo repo
There are several options to run the demo. 1) The first option is to mock raiden `hypercorn "webserver:build_app(mock='raiden')"` 2) The second option is

to run the actual demo `hypercorn "webserver:build_app(config_file='raiden_config.toml')"`

NOTE: It is required to reboot the Udoo after stopping a demo run. In theory just a master reset of the arduinio is required, but as this is not physically possible the UDOO developers implemented a master reset of the arduino on every boot of the UDOO.

RPi

You can login to the RPi via ssh `pi@raspberrypi` if you're in the bbot wifi, the password is `raiden`.
To start the sender logic on the RPi activate the virtualenv `workon demo` and execute the sender_main.py `python ~/demo-train/sender_main.py`.

Debugging

- For the majority of Raiden related issues the deletion of the raiden datadir proved to be sufficient `rm -fr ~/.raiden`.
- Sometimes the Raiden nodes on the Udoo crash during bootup. This can be checked by running `top` and see if there are 6 raiden instances active. If not reboot the UDOO and retry it
- Sometimes the connection to the Arduino breaks. Either because of a missing restart or because of the arduino not responding to the serial port anymore. This can be fixed by shutting the UDOO down and turning off the power supply for ~10s.