A Framework for Swarm Robotics Experimentation with Pi-puck Robots and an Ethereum-based Blockchain

A. Pacheco, V. Strobel, and M. Dorigo
A Framework for Swarm Robotics Experimentation with Pi-puck Robots and an Ethereum-based Blockchain

Alexandre Pacheco, Volker Strobel, and Marco Dorigo
IRIDIA, Université Libre de Bruxelles, Brussels, Belgium
Contact: alexandre.melo.pacheco@gmail.com
January 2020

Abstract
This technical report documents the process used to establish a testbed for running blockchain experiments on the Pi-pucks, starting from a blank SD card and a Pi-puck equipped with only bare-bone components. A local Ethereum blockchain is created for this purpose and is synchronized among the swarm using the integrated Wi-Fi abilities of the Pi-puck. Decentralized Wi-Fi communication (i.e., independent of any router/host/DNS server) is achieved by establishing a Mobile Ad-Hoc Mesh Network between the robots.
1 Introduction

The Pi-puck [7] (fig. 1) is a Raspberry Pi extension board for the e-Puck [5], a popular research mini-robot. Interfacing the e-puck with a Raspberry Pi Zero W (fig. 2) improves the robot communication capabilities and computational power at a low cost, at the same time adding only minimal additional power consumption. The use of the Raspberry Pi extension allows the implementation of more complex features in robot swarms while complying with the standard swarm intelligence requirements of simplicity, robustness and low cost. For these reasons, the Pi-puck is an ideal platform for swarm robotics research.

Recently, the implementation of blockchain-based consensus in robot swarms has been shown to be a novel and very promising approach to the solution of security related problems [10]. The Pi-pucks are the ideal platform for blockchain experiments in swarm robotics since the Raspberry Pis are frequently used in blockchain projects as a single-board computer capable of supporting a fully-fledged Linux operating system.

This technical report walks the reader through all the steps required to run blockchain experiments on the Pi-pucks. In section 2 the hardware and prerequisites for running the experiments are established. The installation process is done in parallel for a Linux/MAC/Windows computer and an SD card running the Pi-puck operating system. Section 3 explains how to install the Ethereum client and how to configure the local blockchain network. A local Ethereum network facilitates running a testbed without concerns of spending valuable tokens. In section 4, we install the Python packages required for robots to communicate with the local Ethereum network in a clean and flexible manner. Python was the chosen language for the implementation of the controllers since the Pi-pucks support Python controllers natively and ethereum.org officially supports the web3.py console which can be used to communicate with the blockchain using Python. In section 5, we establish the communication protocol that robots use to exchange blockchain information. For this setup we establish a Mobile Ad-Hoc Mesh Network (MAMN) which is decentralized in the sense that it requires no router/host/DNS server. Instead, any participating node can send data which is automatically relayed between nodes using a routing algorithm. Using digital signatures in the blockchain we can verify the identity of the message sender, and also that the contents of the message have not been altered in transit. Section 6 is reserved for examples and running final tests on the setup. Finally, in section 7 we conclude with possible directions of research using this blockchain experiments testbed, as well as pointing limitations and elements to be improved on the setup.

Figure 1: Pi-puck extension board mounted on an e-puck. Credit: GCtronic

Figure 2: Raspberry Pi Zero W
2 Hardware and Pi-puck Operating System

In this section we configure the hardware required to create the blockchain robot experiments testbed. To recreate the setup of $n$ nodes as described in this technical report it is required:

- $n$ Pi-pucks
- $n$ SD Cards with at least 16 GB
- a computer (for SSH access, managing SD cards and auditing the network)
- a Wi-Fi dongle (if the computer being used is a desktop with no Wi-Fi board)
- peripherals for the RPi (screen, keyboard, charger, mini-HDMI)

The last item may be omitted by connecting to the Pi-puck using VNC remote access which runs on boot by default with the Pi-puck operating system. More details on this setup are available on the GCTronic Pi-puck Wiki\(^1\).

An image of the Pi-puck operating system can also be downloaded from the GCTronic Pi-puck Wiki\(^2\).

To format the SD card, the SD Card Formatter tool can be used on Windows or Mac. On Linux the same can be achieved using the `parted` tool.

Finally, flashing the SD card is possible via the command line or via the GUI tool balenaEtcher\(^3\). Only one SD card should be pre-installed with the Pi-puck OS image since the remaining cards will be installed from a custom image created at the end of the installation steps explained in this report. Once everything is configured in the single SD card, an image file can be created from it by running the following command on a Linux machine:

```
computer$ sudo dd bs=4M if=/dev/sdb of=image.img
```

The path `/dev/sdb` can be found by running `lsblk` and looking at the available memory of each device in order to identify the SD card.

Throughout this technical report `$computer$` refers to the monitoring computer where we do general setups, and `$pi$` refers to the Pi-puck equipped with the SD card we wish to clone once all installation steps are concluded.

\(^1\)https://www.gctronic.com/doc/index.php?title=Pi-puck
\(^2\)gctronic-stretch-ros-kinetic-opencv3.4.1.img.tar.gz
\(^3\)https://www.balena.io/etcher/
3 Local Ethereum Network

Go-Ethereum (geth) \[2\] is the official implementation of the Ethereum Protocol in the Go language. There are other implementations of the protocol such as parity \[11\] which is written in the Rust language. For this setup we chose to use geth since it has been around for longer and has a larger supporting community.

The tool geth acts as a command line interface used to run an Ethereum node and to perform actions such as mining blocks, sending transactions, creating smart-contracts, or exploring the blockchain. After installing geth it becomes possible to connect to the main Ethereum network (mainnet) and begin synchronization of the blockchain. This process can be lengthy since it involves downloading and performing cryptographic proofs for each block on the longest, current blockchain.

On the Raspberry Pi it is theoretically possible to connect to the mainnet, although certain actions will be limited: mining is not be possible since Ethereum uses the resource heavy Proof-of-Work (PoW) consensus algorithm, and synchronizing a full node on an SD card could take multiple days (compare an SD card write speed of 12.5 MB/s versus a modern SS drive’s 1200 MB/s). At time of writing, the blockchain size is 120.8 GB so available disk space is also a factor to keep in mind in addition to the computational resources required to verify the hash of each block on the current chain. For these reasons only a --light synchronization of the mainnet is realistic for a Raspberry Pi Zero W. A light node allows actions such as broadcasting transactions and smart contracts to the network but does not inherit the trustless and tamper-proof characteristics of full nodes.

In this project we instead create a local Ethereum network called pipuck and run geth to initialize the blockchain from a custom genesis block tailored for the abilities of the Pi-pucks and goals of the experimental setup.

The first step is to install geth on both the monitoring computer and on the Pi-puck. This installation is performed a single time on the Pi-puck since its SD card will be cloned for each robot.

3.1 Install geth on the computer

It is useful to install geth on the computer in order to perform maintenance actions such as generating the genesis block and auditing the network during experiments. On Ubuntu the latest version of geth can be installed via the PPA repository.

```
computer$ sudo add-apt-repository -y ppa:ethereum/ethereum
computer$ sudo apt-get update
computer$ sudo apt-get install ethereum
```

And finally, verify that the installation of geth was successful

```
computer$ geth version
Geth
Version: 1.9.10-unstable
Git Commit: e9e69d6e296d21a7c9c66024c9888a862027252e
Git Commit Date: 20200113
Architecture: amd64
Protocol Versions: [64 63]
Go Version: go1.13.6
Operating System: linux
```
3.2 Install geth on the Pi-puck

The most recent version of Ethereum can be found at https://geth.ethereum.org/downloads/. The Raspberry Pi Zero W has an ARMv6 processor. The following commands download and install geth version 1.9.9 for the ARMv6 processor.

```bash
pi$ wget https://gethstore.blob.core.windows.net/builds/geth-linux-arm6-1.9.9-01744997.tar.gz
pi$ tar zxvf geth-linux-arm6-1.9.9-01744997.tar.gz
pi$ cd geth-linux-arm6-1.9.9-01744997
pi$ sudo cp geth /usr/local/bin/
```

And finally, verify that the installation of geth was successful on the Pi-puck.

```bash
pi$ geth version
geth version 1.9.9.
```

3.3 Generate a custom Genesis Block

The genesis block is the first block of the blockchain and is typically referred to as block number 0. This block contains all the hard-coded specifications of the blockchain and can be tailored to the goals of the local network prior to its deployment. A new genesis block can be created with the Ethereum command-line tool `puppeth` which is installed alongside geth on $computer. During the creation of the block, the tool asks for all the relevant configurations of our local Ethereum network `pipuck` such as:

- network name
- consensus protocol
- block time
- sealing accounts
- pre-funded accounts
- chain/network ID

The following command creates a new account:

```bash
computer$ geth --lightkdf --datadir=~\wherever account new
```

The flag `--lightkdf` reduces the memory requirements for the creation of the cryptographic keys. For our experiments we have generated 20 accounts which all have the same password “123456”. It is useful to have pre-generated accounts which can be reused as many times as wanted. In section 3.4 is shown how to download and use one of these addresses for local network experiments.

When running `puppeth`, we have used the following options:
Please specify a network name to administer (no spaces, hyphens or capital letters please)
> pipuck

What would you like to do? (default = stats)
1. Show network stats
2. Configure new genesis
3. Track new remote server
4. Deploy network components
> 2

What would you like to do? (default = create)
1. Create new genesis from scratch
2. Import already existing genesis
> 1

Which consensus engine to use? (default = clique)
1. Ethash - proof-of-work
2. Clique - proof-of-authority
> 2

How many seconds should blocks take? (default = 15)
> 15

Which accounts are allowed to seal? (mandatory at least one)
> 0xee3a20530907f8478e736b2f54b91c82751a8702
> 0x57047c7e2775cb7cb7d1638442756e8ea251dd74
> 0x6bb61b721ec45e899ac60921efe947c114e814
> 0x8aadcbc427c41a4413b4b5b6cf5d45b4a77ee20
> 0x2363472dfc512bca641f5dc0738a199f173491b
> 0x9b15ace96d8fff3af94c28da152cf465f6cb97c3801
> 0xe02e8b748b04ca1535e052695fca7ae083ab
> 0x2c564a95f7ab2d844b4bba1f31ba07c08cd2c5b
> 0x8ffaa878633e7f1f50a3e949c5714dc2b15663a
> 0x0d9c3610504e02b577195f7b9239c177d29ac
> 0x0c534f56285eca04c7eaf3f96514a2dce9813675
> 0x7777f0c0a887a0ce00c43d7b810a58a9eef23323
> 0xb3833a2063836639ef7d8530c7594a19086644
> 0x4d4263a96349d88456229accda251d18e0c39d
> 0x5843e31ecca1156e22bd3752d6f60665316e51
> 0x31dee0b02efica7e9fa186655e7ef889f239f
> 0xf43a566f484e1b1e2e2b9e6504ac2dd05863a470
> 0x25545e345c863f34a595d6148cb0ee70801ee0
> 0xd6db0c2f32e3ab18e5783043019e28e050984a
> 0x75a9d3e944e9b02f6e079793f96512bb34e0d2e
> 0x

Which accounts should be pre-funded? (advisable at least one)
> 0xee3a20530907f8478e736b2f54b91c82751a8702
> 0x57047c7e2775cb7cb7d1638442756e8ea251dd74
> 0x6bb61b721ec45e899ac60921efe947c114e814
> 0x8aadcbc427c41a4413b4b5b6cf5d45b4a77ee20
> 0x8aadcbc427c41a4413b4b5b6cf5d45b4a77ee20
> Should the precompile-addresses (0x1 .. 0xff) be pre-funded with 1 wei? (advisable yes)
> yes

Specify your chain/network ID if you want an explicit one (default = random)
> 1515

What would you like to do? (default = stats)
1. Show network stats
2. Manage existing genesis
3. Track new remote server
4. Deploy network components
> 2

1. Modify existing configurations
2. Export genesis configurations
3. Remove genesis configuration
> 2

Which folder to save the genesis specs into? (default = current)
Will create pipuck.json, pipuck-aleth.json, pipuck-harmony.json, pipuck-parity.json
> [enter]

### 3.4 Start the local Ethereum Node

Now that the genesis block is configured for our experiments the next step is to initialize the blockchain and its path. The directory `~/ethereum` is the default folder for the Ethereum mainnet and contains the folders keystore and geth which hold the keys for accounts and the chaindata respectively.

In this case, we choose to create a directory `~/mygethnode` and initialize geth using this data directory and the genesis file created in 3.3.

```
$ mkdir ~/mygethnode
```
Additionally, the user accounts can be added by downloading from github the accounts we have generated, pre-funded and added to the block sealers list.

```bash
$ mkdir ~/mygethnode/keystore
$ wget <ADD LINK WHEN GIT IS MADE PUBLIC>
```

Now at any point the local Ethereum node can be hosted by calling `geth` with the following options:

```bash
$ geth --datadir ~/mygethnode/ --syncmode 'full' --port 30311 --rpc --rpcaddr 'localhost' --rpcport 8501 --rpcapi 'personal,db,eth,net,web3,txpool,miner' --networkid 1515 --gasprice '1' --allow-insecure-unlock --nodiscover --verbosity 5
```

Should the `chaindata` folder be out-of-sync with the latest blocks on the network (for example, if the node was disconnected for a while) the network will automatically synchronize by downloading subsequent blocks and verifying the hashes.

In case it is necessary to reset the blockchain to block 0, or initialize with a new genesis block, the folder containing `chaindata` needs to be removed.

```bash
$ rm -r geth
$ geth --datadir ~/mygethnode/ init ~/mygethnode/<YOURGENESISBLOCK>.json
```

### 3.5 Interact with the local Ethereum Node

There are various ways to interact with the running `geth` node. The option that is native to the installation of `geth` is the Javascript Console\(^4\). To connect to the Ethereum mainnet using this tool one would run `geth attach`. However, to connect to other instances of `geth` it is required to input the path to an IPC file, an HTTP address or a websocket. In our case we use the IPC file that is automatically generated upon initialization of the blockchain.

```bash
$ geth attach ~/mygethnode/geth.ipc
```

Welcome to the Geth JavaScript console!

```
instance: Geth/v1.9.10-unstable-e9e69d6e-20200113/linux-amd64/go1.13.6
coinbase: 0xee3a20530907f8478e736b2f54b91c82751a8702
at block: 0 (Sun, 26 Jan 2020 19:05:34 CET)
datadir: /home/iridia/mygethnode
modules: admin:1.0 clique:1.0 debug:1.0 eth:1.0 miner:1.0 net:1.0 personal:1.0 rpc:1.0 txpool:1.0 web3:1.0
```

In the next section we present a Python alternative to the Javascript Console.

\(^4\)https://github.com/ethereum/go-ethereum/wiki/javascript-console
4 Interacting with the Blockchain using Python

In the previous section we showed how to interact with the pipuck network using Javascript Console commands. Now the goal is for robots to trigger blockchain events as a part of their high-level control functions. In order to establish a clean interface from which robots can send transactions, evaluate smart contracts and/or audit the blockchain data, it makes sense that the interface is written in the same language as the robots controllers.

For this framework we have chosen to implement the Pi-puck controllers in Python since it is a well-known high level language among researchers. In fact, the Pi-pucks feature a complete Raspbian Stretch operating system which comes with Python/Python3 pre-installed. The low level interface to the sensors and actuators of the e-puck base is handled using the `smbus` package which includes a library of SMBus (System Management Bus) functions. With this setup, researchers can implement controllers using Python at a high level.

It then makes sense to use a Python wrapper for the Ethereum console, such as the web3.py\(^5\) \(^3\). web3.py is the most developed and active geth console repository based on Python.

In the end, we achieve a clean and flexible setup for blockchain experiments with robots where high-level controllers are implemented solely in Python. In the following subsections we walk the reader through the steps required to install web3.py on Raspbian Stretch.

4.1 Pre-configuration

Raspbian Stretch comes pre-installed with Python3.5. Unfortunately, the most recent version of web3.py only offers support starting from Python3.6. Older versions of web3.py are no longer supported and are limited when it comes to interfacing with a local network running Proof-of-Authority. For this reason, we choose to install Python3.6.

**Note:** The following process can take multiple hours on a Raspberry Pi Zero W. We recommend to do it a single time before cloning the SD card for other nodes.

First, update all the required packages:

```
pi$ sudo apt-get update
pi$ sudo apt-get install -y build-essential tk-dev libncurses5-dev libncursesw5-dev libreadline6-dev libdb5.3-dev libgdbm-dev libsqlite3-dev libssl-dev libbz2-dev libexpat1-dev liblzma-dev zlib1g-dev libffi-dev
```

The following snippet installs Python3.6 from source. This step can take up to 2 hours to complete on a Raspberry Pi Zero W.

```
pi$ wget https://www.python.org/ftp/python/3.6.10/Python-3.6.10.tgz
pi$ sudo tar xzf Python-3.6.10.tgz
pi$ cd Python-3.6.10
pi$ .configure
pi$ make
pi$ make altinstall
```

To verify that the installation succeeded, type:

```
pi$ python3.6 -V
pi$ pip3.6 -V
```

\(^5\)https://github.com/ethereum/web3.py
The package installer pip should have been installed as well. The system is ready to install web3.py as well as an updated version of smbus which works with Python3.6.

```
sudo pip3.6 install web3
sudo pip3.6 install smbus2
```

## 4.2 Proof-of-Work

Start the Python3 terminal by typing $ python3.6 and then write

```
>>> from web3 import Web3
>>> my_provider = Web3.IPCProvider('~/mygethnode/geth.ipc')
>>> w3 = Web3(my_provider)
```

It is possible to check if the connection succeeded by checking the console client version.

```
>>> w3.clientVersion
'Geth/v1.9.9-stable-01744997/linux-arm/go1.13.5'
```

## 4.3 Proof-of-Authority

In order to interface with a local Proof-of-Authority (PoA) network it is necessary to use geth_poa_middleware. This is required due to lack of consensus in a single PoA protocol. To do this the user must type in a Python terminal:

```
>>> from web3 import Web3, IPCProvider
>>> w3 = Web3(IPCProvider('~/mygethnode/geth.ipc'))
>>> from web3.middleware import geth_poa_middleware
>>> w3.middleware_onion.inject(geth_poa_middleware, layer=0)
>>> w3.clientVersion
'Geth/v1.9.9-stable-01744997/linux-arm/go1.13.5'
```

To facilitate access to the terminal window we have made a console.py file accessible on github. This file is used in section 6 when performing tests and running examples on the setup.
5 Mobile Ad-Hoc Mesh Network

In order to take full advantage of the Wi-Fi communication abilities of the Pi-puck without compromising the decentralization of the swarm we establish a Mobile Ad-hoc Mesh Network (MAMN).

The advantage of an MAMN is that it does not rely on any central hubs (such as routers or master servers) nor assumes global connectivity. Instead, each node participates in routing by forwarding data for other nodes. In Linux-based systems the determination of which nodes forward data and to where can be done dynamically via the routing algorithm of the batman-adv module which has been part of the Linux kernel tree since version 2.6.38.

Establishing an MAMN network between Pi-pucks, or any other Linux-based machine, using this module is a simple process.

5.1 Pre-configuration

Despite batman-adv being part of the Linux kernel, the userspace driver, batctl, as well as the user daemon alfred must be installed on both the Pi-pucks and the monitoring computer.

```
$ sudo apt install libnl-3-dev libnl-genl-3-dev
$ git clone https://git.open-mesh.org/batctl.git
$ cd batctl
$ sudo make install
$ sudo apt-get install alfred
```

5.2 Setup Network

To setup the network, first we must select a web interface. This interface could be an Ethernet port, but that would defeat the purpose of an MAMN. The interfaces available can be identified by running `iwconfig`. In the Raspberry Pi Zero W the default Wi-Fi interface is `wlan0` which will be used for all nodes, whereas on the computer it is necessary to search for the interface corresponding to the Wi-Fi dongle extension.

```
$ iwconfig
wlan0     IEEE 802.11  ESSID:"AdHocPi"
         Tx-Power=31 dBm
         Retry short limit:7  RTS thr:off  Fragment thr:off
         Power Management:on
lo no wireless extensions.
```

Create a file titled `setup-adhoc` which contains the following code:

```
#!/bin/bash
# Running this script will disable internet connection

# Disable wpa_supplicant and dhcpcd
systemctl stop wpa_supplicant.service
systemctl stop dhcpcd.service

# Setup Ad-Hoc Network
```
ip link set wlan0
iw wlan0 set type ibss
ifconfig wlan0 mtu 1499
iwconfig wlan0 channel 3
ip link set wlan0 up
iw wlan0 ibss join AdHocPi 2432

# Configure batman-adv virtual network
modprobe batman-adv
batctl if add wlan0
ip link set up dev wlan0
ip link set up dev bat0
ifconfig bat0 172.27.0.2/16

Each node in the network should have an unique IP address in the last line of setup-adhoc. To run the network automatically on boot, make the file executable and place it in /usr/local/bin/

$ chmod a+x setup-adhoc
$ sudo mv setup-adhoc /usr/local/bin/

Create a file called adhoc.service and place it in the folder /lib/systemd/system/. Copy the following contents to the file:

[Unit]
Description=Set up Ad Hoc network (B.A.T.M.A.N.)

[Service]
ExecStart=/usr/local/bin/setup-adhoc

[Install]
WantedBy=multi-user.target

Finally, enable and start the service.

$ sudo systemctl enable adhoc.service
$ sudo systemctl start adhoc.service

If you want to disable the service (for example, to access the node via a regular Wi-Fi network) type

$ sudo systemctl disable adhoc.service
$ reboot now

5.3 Troubleshooting

It may be relevant to double check if wpa_supplicant.service and dhcpcd.service are indeed inactive. These services interfere with the network configuration.

$ systemctl status wpa_supplicant.service
$ systemctl status dhcpcd.service

12
If there is an error related to RF-kill when running `set wlan0 up` it can be fixed as follows:

```
RTNETLINK answers: Operation not possible due to RF-kill
$ sudo rfkill unblock all
```
6 Testing the Setup

6.1 Start the network

Once all Pi-pucks have booted they should automatically connect to the ad-hoc network if the procedure in section 5 was followed. On the monitoring computer (or any Pi-puck) you can verify which nodes are connected by calling `batctl`.

```
computer$ sudo batctl n
0/d2:5e:62:5c:69:99 BATMAN_IV)]
IF Neighbor last-seen
wlx801f029ba623 b8:27:eb:30:e2:9a 0.372s
```

The Ubuntu machine is communicating with the Pi-puck (MAC address b8:27:eb:30:e2:9a) over the interface wlx801f029ba623, which corresponds to the USB Wi-Fi dongle. The batman-adv module verifies connectivity approximately every second and works with graphviz to generate connectivity diagrams. This tool can be a useful feature to evaluate the effect of network connectivity in consensus, for example.

Now it is possible to access the terminal on any Pi-puck using `ssh`. We will do this now in order to test other elements of the setup such as running a geth node.

```
computer$ ssh pi@172.27.1.101
pi$
```

To synchronize geth between machines, the network must be initialized using the same genesis block as presented in section 3. The file 2peers.json available on Github was created for a network of two nodes. The following snippet initializes the network and starts the geth instance on either machine.

```
$ cd ~/mygethnode
$ rm -r geth
$ wget -O 2peers.json https://raw.githubusercontent.com/Pold87/geth-on-epuck/master/2peers.json?token=AOJYXZCA25GhJYPKHHQ1E2K61264Y
$ geth --datadir ~/mygethnode/ init ~/mygethnode/2peers.json
$ geth --datadir ~/mygethnode/ --syncmode 'full' --port 30311 --rpc --rpcaddr 'localhost' --rpcport 8501 --rpcapi 'personal,db,eth,net,web3,txpool,miner' --networkid 1515 --gasprice '1' --allow-insecure-unlock --nodiscover --verbosity 5
```

To communicate with the geth instance using Python3.6, open a new terminal window and download the file from GitHub which boots the Web3 Python Console.

```
$ cd ~/mygethnode
$ wget -O console.py https://raw.githubusercontent.com/Pold87/geth-on-epuck/master/console.py?token=AOJYXZHI41ZWSWHPLQCMFGS6IV0CK
$ python3.6 -i console.py
Welcome to the geth Python Console
Version: Geth/v1.9.10-unsafe-e9e69d6e-20200113/1inux-amd64/go1.13.6
Your public address is: 0xEe3a20530907F8478E736b2f54b91C82751A8702
```

[https://github.com/Pold87/geth-on-epuck/blob/master/2_setup_geth](https://github.com/Pold87/geth-on-epuck/blob/master/2_setup_geth)
Your enode is:
enode://d3b88bd83a69246ab6e8542ee84b6c14fde1e3f0470eb41c37a99cd0ddff1fa1ed3a
15234ac75439aa18abe17d8c6b059a2aa078266da629654a5ba46895f3570
10.129.240.219:30311?discport=0

```python
>>> w3.geth.admin.addPeer('<<<ENODE OF PEER>>>)
True
>>> w3.geth.miner.start()
>>> w3.eth.getBlock('latest').number
```

Note: Sometimes the enode IP is automatically selected from the available network interfaces. It may be required to manually correct the enode IP to the one associated with batman-adv

### 6.2 Listen for blockchain events

```
$ cd ~/mygethnode
$ mkdir control
$ wget -O listen.py https://raw.githubusercontent.com/Pold87/geth-on-epuck/master/control/listen.py?token=AOJYXZGK257H6SQVJNAFSR26IVQGQ
$ python3.6 -i console.py
>>> listen.py
```

The Python script `listen.py` flashes the front LED lights of the Pi-puck whenever a block is mined.

### 6.3 Trigger blockchain events

```
$ wget -O groundsensor.py https://raw.githubusercontent.com/Pold87/geth-on-epuck/master/control/groundsensor.py?token=AOJYXZBUN55CLQK71B7HTN26IVD6K
$ python3.6 -i console.py
>>> groundsensor.py
```

The Python script `groundsensor.py` sends a transaction on the blockchain whenever the ground sensors on the Pi-puck return black.
7 Conclusions

The setup described in this technical report is aimed at researchers who intend to run blockchain experiments on a group of robots. Although the report targets specifically the Pi-pucks, it can easily be extended to any robot capable of running a Linux based OS, and can be useful for researchers who wish to conduct real-world experiments in fields such as swarm robotics, distributed robotic systems, cyber-physical systems and Internet-of-Things, among others.

Research addressing the applications of blockchain technology for robotics is a hot research topic and, as such, there is a high demand for platforms to run blockchain experiments on real-world robots. Since 2018 there has been a number of simulation results for blockchain applications in robots such as: the achievement of consensus in robot swarms in the presence of Byzantine robots [1, 10]; the improvement of communications and performance in industrial robots [4]; the formation of coalitions in cyber-physical systems [6]; the management of collaboration in heterogeneous multi-robot systems [9]; the secure collection of data from robots [13]; and path planning in multi-robot systems [8].

One of the main advantages of the setup presented in this technical report is that it only requires basic knowledge of Python programming, and cheap components such as the Raspberry Pi’s Zero W. One of the drawbacks is that for the moment the only protocol implemented is Ethereum, which was designed as a global platform for distributed computing and not specifically for implementation in robotic systems.

The setup we describe in this technical report can be used to conduct research in the applications of blockchain for swarm intelligence systems, specifically, the role it can play in consensus, decision making, security and control of robot swarms. In future work we will be looking to use the platform to implement fault-tolerant best-of-n solutions to classical consensus problems [12] such as those introduced in [10]. Other directions of robot swarm related research may involve, for example, blockchain for secure mapping, blockchain for secure data logging, blockchain for secure flocking or the use of the blockchain for high level decision making.
References


