LAB 09: DHT implementation with Flask (part 2)

Define Useful Tasks for PyInvoke

- Define common tasks:
  - NETWORK BOOT –> Boots two nodes thanks to the bootloader, wire them together manually
  - NETWORK KILLALL –> (VERY!!) useful for debugging purposes... when you want to restart the network after bug-fixing :)

NB:

- biggest troubles come from checking outputs of multiple nodes.
- Output must necessarily be redirected to files (goodbye consoles)
- Popen inside tasks 4 redirecting output to logfiles
- Tweak Flask to redirect stdout to your logfile

3rd Checkpoint

Check that py-invoke is really able to boot two nodes which are pred and succ of each other; Check that you can kill all your DHTnode processes (and clean consoles/output files) when you need to do so.

Find Responsible Node with iterative approach

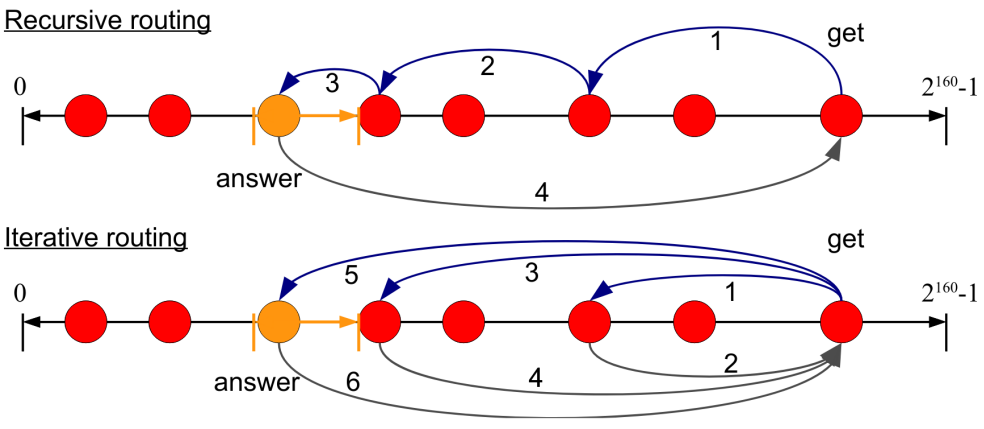

Figure 1: Recursive VS iterative approach

- Implement findNode(key) local method

I mean, a Flask application (which is a DHT node), when asked to serve a JOIN / STORE / LOOKUP request should be able to identify the node responsible for the key associated to the received request.

We will use an **iterative routing** approach, i.e., the "initiator" personally contacts all the many nodes that may be involved in the "redirects-chain" which leads to the discovery of the searched (responsible) node

GOOD TO KNOW: How to send web-requests via python-code (not anymore only through POSTMAN)

Requests is an elegant and simple HTTP library for Python

For example, if we want to implement the "LINEAR" findNode, then we must keep finding info of successor nodes, until we find the good one...

Snippet to:

1. send a request to a given url and then
2. parse the JSON response we get that contains, wishfully, all data we need to know

```
def processFromNodeInfo(host, port):
    url = "http://{}:{}/nodeinfo".format(host, port)
    respJSON = requests.get(url).json()
    name, pred, succ = respJSON['name'], respJSON['pred'], respJSON['succ']
    return Process(host, port, name, pred, succ)
```

4th Checkpoint

Boot a network; add some nodes; add a GET method that issues a findNode for a given key; check output...

Implement JOIN

- Implement JOIN (only correct placement of joinerNode)
- add a task that creates a joiner node, joiner should then ask to some already deployed node to welcome his join request

```
joinerJSON = joinerProcess.toJSON()
requests.post("http://{}:{}/dht/join".format(proxyip, proxyport), json=joinerJSON)
```

At the end of JOIN you should make rewirings via... PUT requests :) For example:

```
predurl = "http://{}:{}/pred".format(joiner.host, joiner.port)
succurl = "http://{}:{}/succ".format(joiner.host, joiner.port)

requests.put(predurl, data=predOfResp.hostportname())
requests.put(succurl, data=resp.hostportname())
```