The 2nd IEEE International Workshop on Advances in Fog/Edge Computing (AFEC 2021)
July 16th - July 19th, 2021
Athens, Greece

Towards Testbed as-a-Service: design and implementation of an unattended SoC cluster

Gabriele Proietti Mattia, Roberto Beraldi

Department of Computer, Control and Management Engineering “Antonio Ruberti”, Sapienza University of Rome, Italy

proiettimattia@diag.uniroma1.it

SAPIENZA
UNIVERSITÀ DI ROMA

DIAG
Dipartimento di Ingegneria
informatica, automatica e gestionale
Antonio Ruberti

Outline

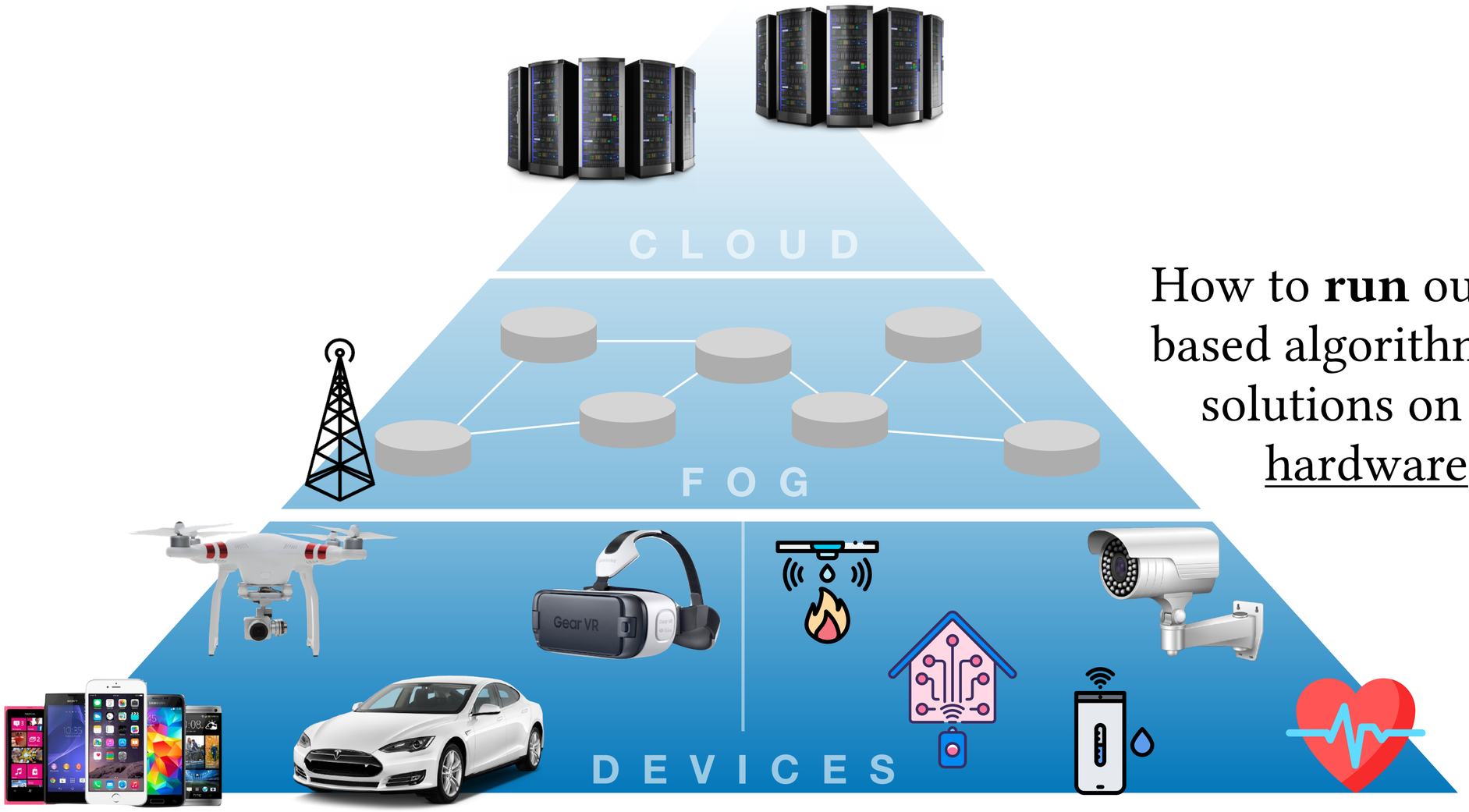
1. **Context and Challenges**
   Creating SBCs clusters for running Fog computing algorithms
2. **Hardware**
   Cluster design from the hardware point of view
3. **Software**
   Cluster design from the software point of view
4. **Results**
   Experiment results on the described cluster
5. **Conclusions**
   Final conclusions

1

Introduction

Context

Fog architecture



~5ms

Average round-trip-time fog-to-device

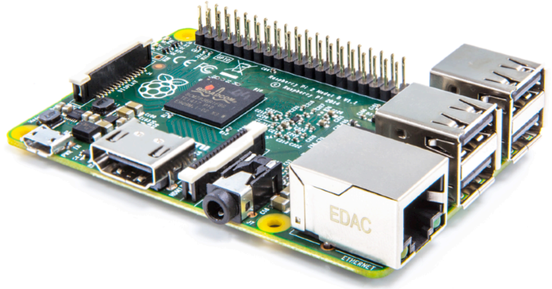
How to **run** our Fog-based algorithms and solutions on <u>real hardware</u>?

Challenge

Building a cluster of SBCs

Using SBCs (Single Board Computers — like Raspberry Pi) it is easy and inexpensive to build a cluster, but we need to take into account:

- proper **enclosure**
- **modularity** (adding/removing components)
- remote **management** (on/off)
- no human presence (**unattended** cluster)



Raspberry Pi 4

In this paper, we aim to present some guidelines for realising a *Testbed-as-a-Service* architecture also showing our own implementation.

State-of-the-art

Raspberry Pi/SBCs clusters in literature

- Doucet et al., "*Learning cluster computing by creating a raspberry pi cluster*", 2017 and K. Doucet, "*The creation of a low-cost raspberry pi cluster for teaching*", 2019 describe a Raspberry Pi cluster that are used for learning purposes, they use MPI for communication between the devices;
- Tso et al, "*The glasgow raspberry pi cloud: A scale model for cloud computing infrastructures*", 2013 and M. d'Amore et al, "*A practical approach to big data in tourism: A low cost raspberry pi cluster*", 2015 propose a Raspberry Pi cluster arranged a structured built with Lego bricks, called PiCloud;
- Cox et al., "*Iridis-pi: a low-cost, compact demonstration cluster*", 2014 presents "Iridis-Pi", a cluster of 64 Raspberry Pis and benchmark its computing capabilities addressing problems like power management, networking and shared storage
- Wilcox et al, "*Pi-crust: a raspberry pi cluster implementation*", 2015 propose the design and a benchmark of a Raspberry Pi cluster showing the total computational power of the architecture
- Mappuji et al, "*Study of raspberry pi 2 quad-core cortex-a7 cpu cluster as a mini supercomputer*", 2016 studies a Raspberry Pi cluster as an High-Performance Computing cluster

They do not address features like: self-enclosing, unattending both in hardware and software.

2

Hardware

Enclosure

Standard ATX Power Supply

Control circuitry

RPis

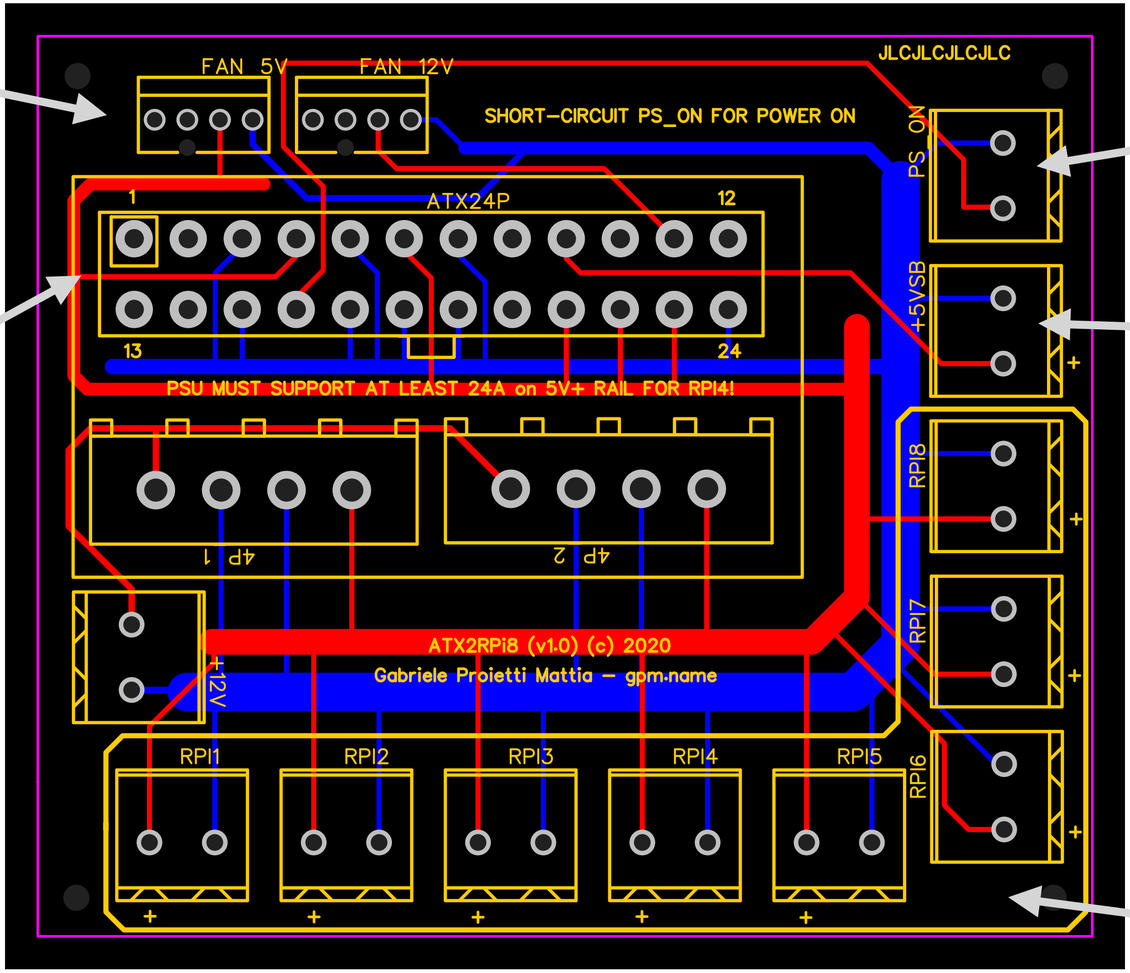
RPis

Power Board

ATX2RPi(8)

Fans Power (5V/12V)

ATX PSU 24pin/ 4pin cables (5V/ 12V)



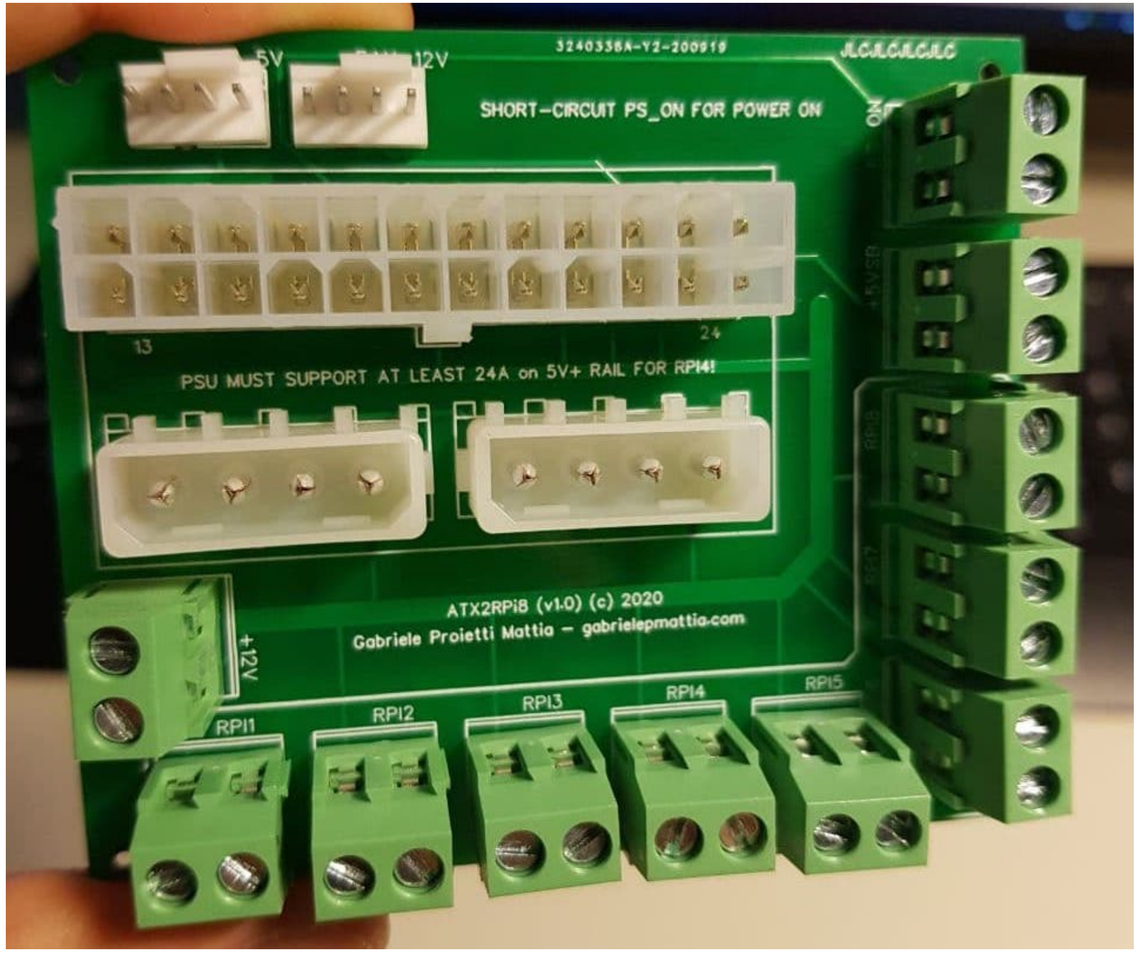
Power switch

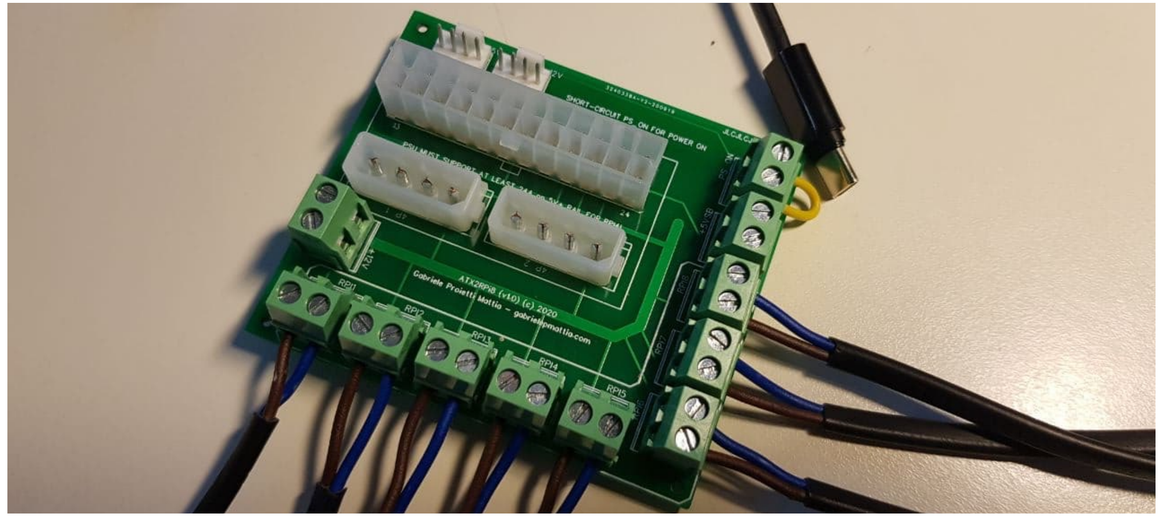
Ethernet Relay Power (5V)

RPi Power (5V)

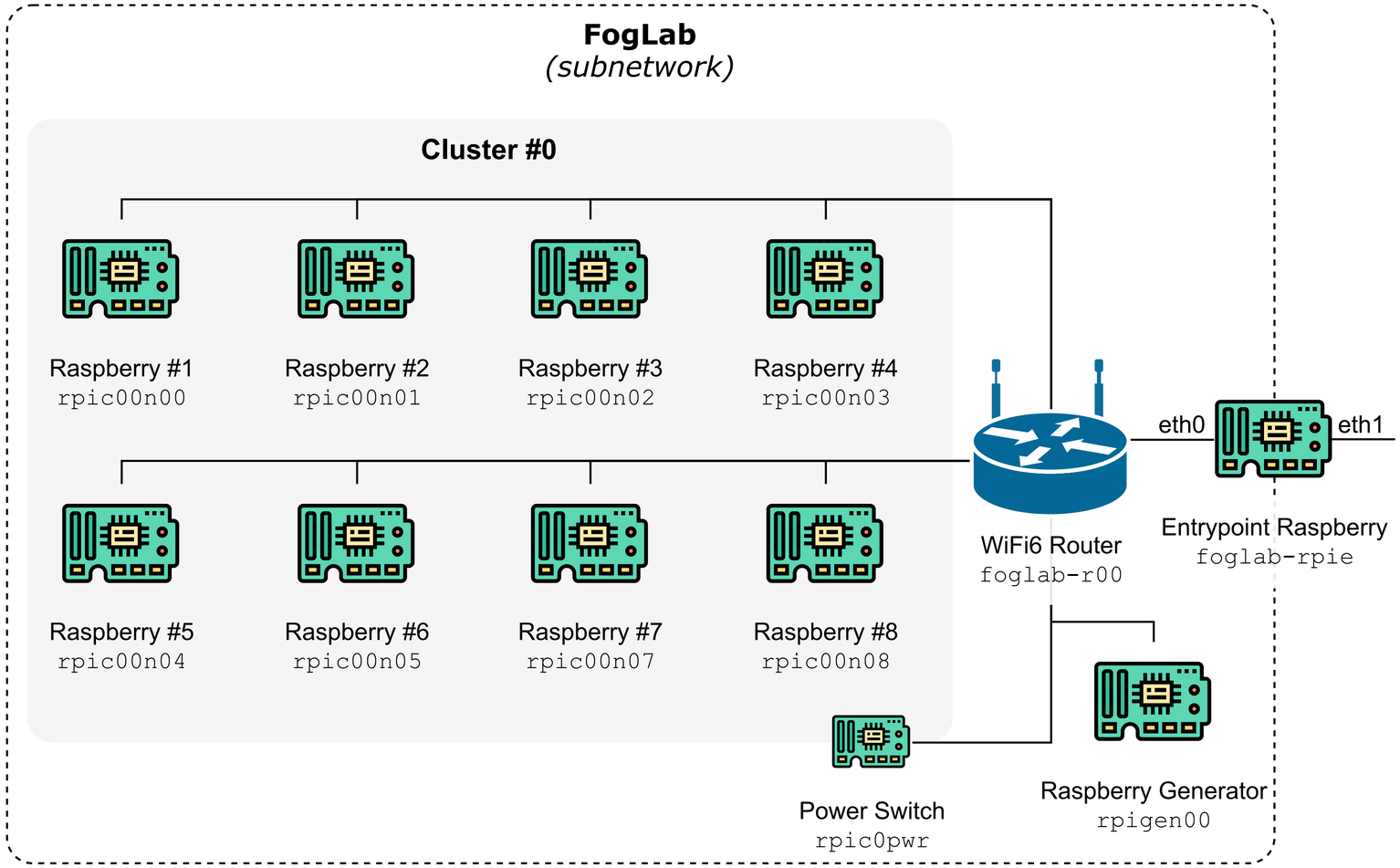
Power Board

ATX2RPi(8)





Final Deploy



3

Software

Operating System

Regarding the operating system, Hypriot OS has been used, a Linux distribution with Docker but it requested some modification:

- root partition has been made **read-only**, for avoiding corruption and manual intervention
- watchdog daemon has been enabled for **auto-rebooting** when freezing
- cloud-init is used for **auto-configuring** nodes upon boot

Then for enabling the Testbed as-a-Service paradigm the following JSON configuration files has been proposed. They represent the Testbed and the experiment to run on it. In our case, they have been implemented with the P2PFaaS (https://gitlab.com/p2p-faas) framework.

Configuration

Listing 1 Testbed JSON configuration file

```
{
  "infrastructure": {
    "nr_nodes": "3",
    "topology": {
      "0": ["1","2"],
      "1": ["0","2"],
      "2": ["0","1"]
    }
  },
  "scheduler": {
    "image": "https://...",
    "name": "SchedulerIdentificator",
    "args": ["arg1", "arg2", "arg3" ],
    "max_parallel_jobs": 4,
    "max_queue_length": 2,
  },
  "discovery": {
    "image": "https://...",
    "heartbeat": "30s"
  },
  "functions": [
    {
      "name": "My Service",
      "api": "my_service",
      "image": "https://...",
      "args": ["arg1", "arg2", "arg3"]
    },
    {
      "name": "My Service #2",
      "api": "my_service_2",
      "image": "https://...",
      "args": ["arg1", "arg2", "arg3"]
    },
  ]
}
```

Listing 2 Testbed experiment JSON configuration file

```
{
  "api": "/my_service",
  "payload": "/path/to/payload",
  "log_path": "/path/to/log",
  "arrivals": {
    "distribution": "poisson",
    "rate": 1.0,
    "rates": {
      "0": 1.0,
      "1": 2.0,
      "2": 1.5,
    },
    "max_num_request": 2000,
  },
}
```

4

Results

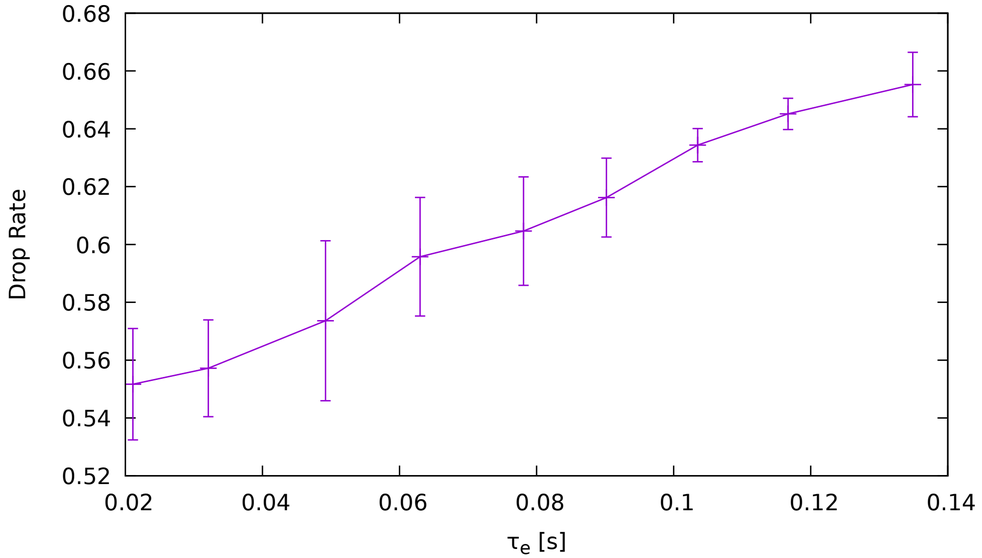
Setting

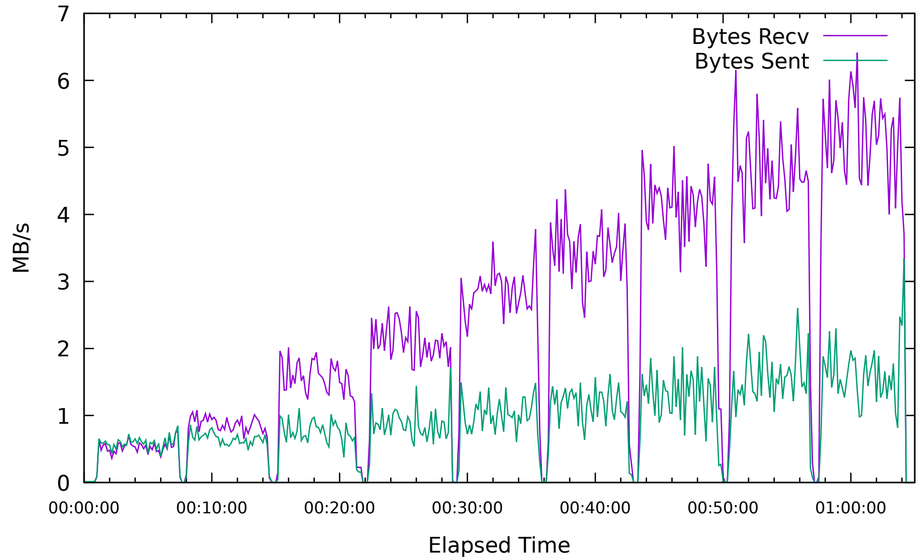
On the proposed cluster the following experiment for benchmarking a distributed load balancing algorithm has been run. The setting is the following:

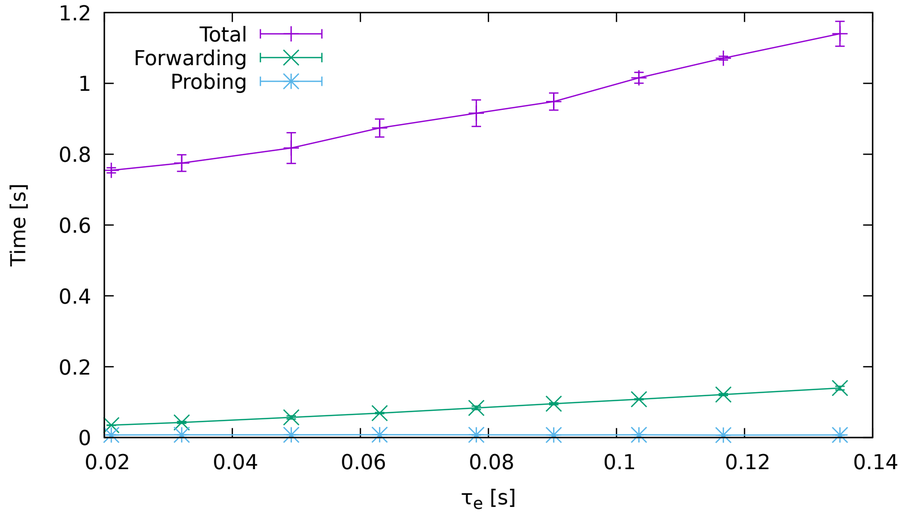
- each node receives a **traffic** of $\lambda = 5.50$ req/s, but each node is capable of executing $\mu = 5.72$ req/s
- each request is a job request to **execute** a function (FaaS), in our case image recognition, each node execute maximum $K = 4$ jobs in parallel with no queue
- each node takes takes a per-job scheduling decision according to the current load, if it is lower than a threshold Θ the job is executed locally, otherwise the node tries to cooperate by using a power-of-random choice scheme

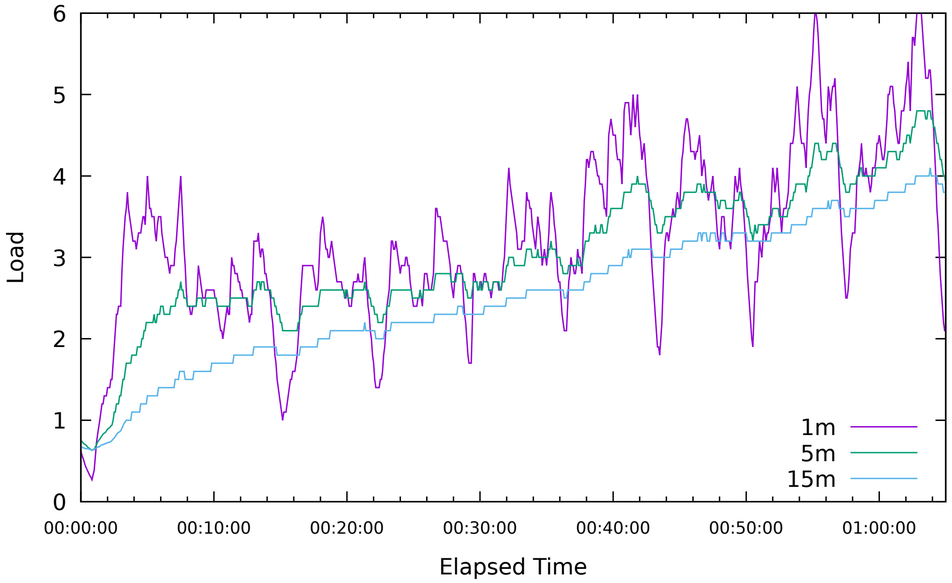
On this baseline the benchmark increases the job payload progressively (from 50kb to 800kb) for studying its impact on the algorithm performances.

Results


Drop Rate
0.68
0.66
0.64
0.62
0.6
0.58
0.56
0.54
0.52
0.02
0.04
0.06
0.08
0.1
0.12
0.14
τe [s]


Bytes Recv
Bytes Sent
MB/s
7
6
5
4
3
2
1
0
00:00:00
00:10:00
00:20:00
00:30:00
00:40:00
00:50:00
01:00:00
Elapsed Time


Total
Forwarding
Probing
Time [s]
1.2
1
0.8
0.6
0.4
0.2
0
0.02
0.04
0.06
0.08
0.1
0.12
0.14
τe [s]


Load
6
5
4
3
2
1
0
1m
5m
15m
00:00:00
00:10:00
00:20:00
00:30:00
00:40:00
00:50:00
01:00:00
Elapsed Time

5

Conclusions

Conclusions

- hardware and software **requirements** has been delineated for a long-term, unattended and remote controllable solution for implementing a Raspberry Pi cluster;
- a **power supply board** (ATX2RPi) has been designed to be compatible with a desktop computer power supply (called ATX); the board can power up to *eight* Raspberry Pi boards;
- a configuration method has been proposed for the testbed and for an experiment, it is based on **JSON** configuration files, towards a *Testbed-as-a-Service* paradigm;
- the results of a distributed scheduling algorithm for Fog computing has been discussed, they show the impact of the job payload on the algorithm performances.

Towards Testbed as-a-Service: design and implementation of an unattended SoC cluster

Gabriele Proietti Mattia*, Roberto Beraldi*

TALK & PRESENTATION

Gabriele **Proietti Mattia**

*Department of Computer, Control and Management Engineering “Antonio Ruberti”, Sapienza University of Rome, Italy