User Guide

1 Introduction

These POX controller programs are designed to reconfigure forwarding flows on OpenFlow-supported switches. They operate by calculating a minimum spanning tree when there is a link failure between switches, and they dynamically install flows according to this reconfiguration mechanism. The programs offer two modes of operation:

1. **Reactive Mode:** In this mode, flows are dynamically installed on switches when packets matching certain criteria hit the switch. The switch forwards these packets as a packet-in message to the controller, which then responds accordingly by installing flows.

2. **Proactive Mode:** In this mode, flows are automatically installed following the calculated spanning tree. This installation occurs immediately after all the switches are connected to the controller.

These programs are intended for use in a microgrid platform composed of four generation units, each linked to an Open vSwitch (OVS) switch. Each switch is then connected to its neighboring switches, as illustrated in Figures 1 and 2.

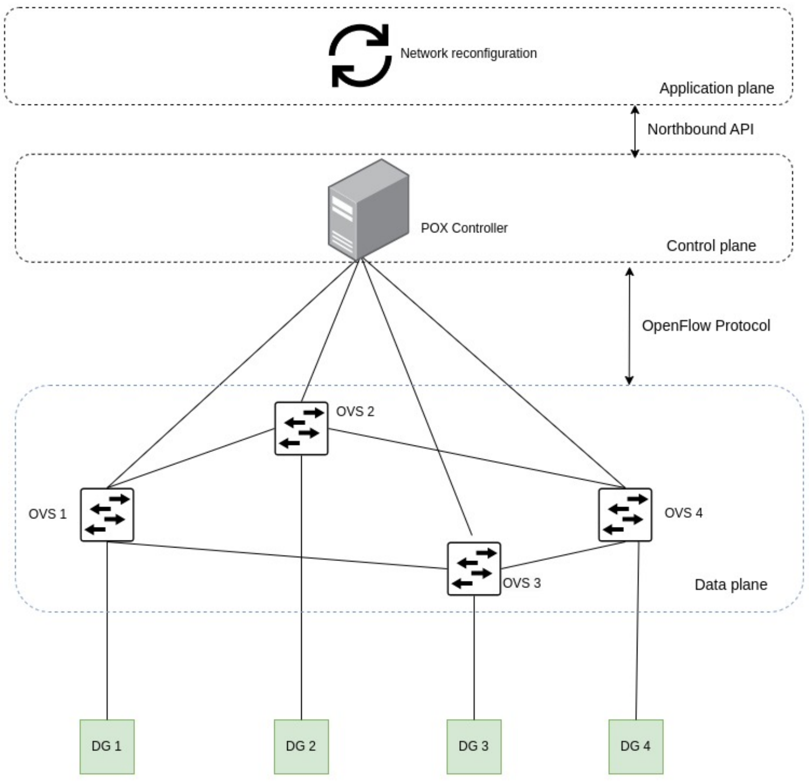

Figure 1: Testbed architecture

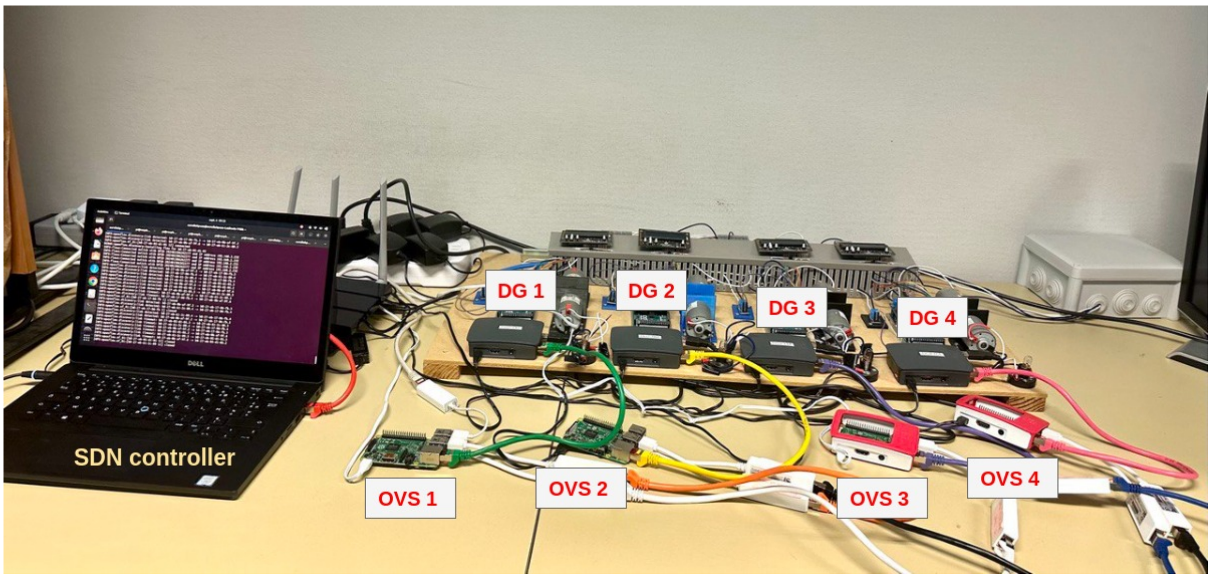

Figure 2: Testbed's experimental configuration

Key Features:

- Flow reconfiguration based on a minimum spanning tree algorithm.
- Two modes of operation: reactive and proactive.
- Suitable for microgrid environments with OVS switches.

2 Prerequisites

Before you can run this controller program, ensure you have the following prerequisites installed:
Python (Version 2.7 - 3.5)

- POX Controller[1]
- SCAPY
- NetworkX

3 Installation

- Pox controller :
  - `git clone http://github.com/noxrepo/pox`
  - `cd pox`
  - `git checkout dart`
- Create an virtual environment named poxx :
  - `python -m venv poxx`
  - `source poxx/bin/activate`
- Install the required python packages : `pip install -r requirements.txt`

4 Running programs

- Activate the "poxx" virtual environment from the Home directory : `source poxx/bin/activate`
- Start the controller for the reactive behavior : `pox/ ./pox.py log.level -DEBUG reactive`

[1]<http://github.com/noxrepo/pox>

- Wait until the switches are connected to the controller, links are collected and the spanning tree is calculated
- Connect to ovs 1 and ovs 2 using ssh, the password is "raspberry": `-ssh pi@192.168.2.232` (ovs 1) `-ssh pi@192.168.2.199` (ovs 2)
- Check flow tables for ovs 1 and ovs 2 :
  - `sudo ovs-ofctl dump-flows s1` for ovs 1
  - `sudo ovs-ofctl dump-flows S2` for ovs 2
- Connect DGs motors to raspberrys
- Connect by ssh to DG consoles the password is "raspberry":
  - `ssh pi@192.168.2.10` for DG 1
  - `ssh pi@192.168.2.11` for DG 2
  - `ssh pi@192.168.2.12` for DG 3
  - `ssh pi@192.168.2.13` for DG 4
- Start client-servers program on each DG starting by the servers :
  - On DG 4 (192.168.1.151):
    - `cd Desktop`
    - `python3.5 endservern.py`
  - On DG 3 (192.168.1.144): `python3.5 endserver.py`
  - On DG 3 (192.168.1.103): `python3.5 serv1C103.py`
  - On DG 1 (192.168.1.145): `python3.5 serv2C145.py`
- Then, open another terminal to connect to DG 1 console by ssh and go to the Download directory to start the data injection program :
  - `ssh pi@192.168.2.10`
  - `cd Download`
  - `python3.5 DataInjection.py`
- Enter an instruction value between 0 and 12
- Check if the instruction is received on terminals of DG 1, DG 2, DG 3 and DG 4.
- Check the flow tables of ovs 1 and ovs 2 :
  - `sudo ovs-ofctl dump-flows s1` for ovs 1
  - `sudo ovs-ofctl dump-flows S2` for ovs 2
- Repeat the previous actions but launch the program for proactive behavior: `pox/ ./pox.py log.level -DEBUG proactive`