SDN Network Performance Monitor and Visualizer using Raspberry Pi

Fabrizio Diaz
Department of Computer Science
National Taiwan University of Science and Technology
Taipei, Taiwan
fabrizio.diaz@uptp.edu.py

Letizia Candia
Department of Computer Science
National Taiwan University of Science and Technology
Taipei, Taiwan
letizia.candia@uptp.edu.py

Violeta Torres
Department of Computer Science
National Taiwan University of Science and Technology
Taipei, Taiwan
celina.torres@uptp.edu.py

Victoria Villamayor
Department of Computer Science
National Taiwan University of Science and Technology
Taipei, Taiwan
victoria.villamayor@uptp.edu.py

***Abstract*—This project presents a Network Performance Monitor and Visualizer that transforms Software-Defined Networking (SDN) concepts into tangible, interactive experiences. The system integrates an Arduino Uno with sensors and LEDs, a Raspberry Pi running Mininet, and Python middleware to create real-time bidirectional communication between physical inputs and virtual network behavior. Users control network traffic patterns via joystick, trigger link failures with button presses, and observe network state through color-coded LED indicators. The implementation demonstrates core SDN principles including centralized control, dynamic flow rule installation, and automatic path recalculation. This platform provides an educational tool that bridges the gap between theoretical SDN knowledge and hands-on understanding through immediate visual feedback.**

***Index Terms*—ONOS, Raspberry Pi, Arduino, SDN, Mininet**

I. Introduction

Software-Defined Networking (SDN) has fundamentally transformed modern network architecture by decoupling the control plane from the data plane, enabling centralized network management and programmability. However, despite its growing adoption in enterprise and data center environments, SDN concepts remain abstract and challenging for students and practitioners to fully grasp without hands-on experience. Traditional learning approaches—lectures, simulations confined to computer screens, and theoretical exercises—often fail to convey the dynamic nature of how SDN systems respond to network events in real-time.

This Network Performance Monitor and Visualizer combines an Arduino-based control panel with Mininet physical-virtual network managed by ONOS. Users can select network elements via joystick movements and trigger topology changes with buttons presses. The system responds with LED feedback, transfroming invisible network state into observable phenomena.

II. Components

A. Hardware

TABLE I
Hardware Components and Specifications

Component	**Model**	**Quantity**
Microcontroller	Arduino Uno R3	1
SBC	Raspberry Pi 4	1
Joystick	HW-504	1
Button	HW-483	1
RGB LED	HW-479	1
Green LED	–	2
Temp Sensor	HW-506	1

B. Software

TABLE II
Software Components and Tools

Software	**Purpose**
ONOS	SDN controller for network management
Mininet	Network topology emulation
Python	Integration and communication layer
PlatformIO	Arduino development environment
pySerial	Serial communication library
Requests	HTTP library for ONOS REST API

III. System Architecture

The system consists of three layers that communicate to create real-time interaction between physical hardware and virtual network behavior.

A. Physical Interface – Arduino

This is the connection between the real world and the Network layer. It takes inputs from three types of sensors: the joystick that selects which network device to control (left for h1, right for h2, up for switch, down for both hosts), the button for toggling the state of the selected device, and the temperature sensor for environmental monitoring.

The outputs are visualized using two types of LEDs: an RGB LED which shows the switch status through color coding, and two single-color LEDs that show the status of the host connections (on for connected and off for disconnected).

The Arduino constantly reads the signals from the sensors and sends them via USB serial to the Raspberry Pi while listening for LED control commands from the Python script.

B. *Integration Layer*

A single Python Script is used to translate between hardware events and network commands. It retrieves data from the Arduino and sends them over to ONOS REST API as HTTP requests like such as POST (to install traffic flows), GET (to check link status), and POST /intents (to create traffic intents between hosts).
The layer constantly processes the networks state by monitoring the traffic, the link status and the congestion levels from ONOS. Based on these metrics, it determines the correct output and sends the visual feedback back to the Arduino.

C. *Network Control Layer –ONOS*

It is the infrastructure where ONOS manages the network flow and Mininet recreates the physical network topology. ONOS provides centralized SDN control over Minintet, maintaining a global-view of the network. When there is a request to change the topology or traffic, ONOS translates these into OpenFlow messages and sends them to Mininet switches for execution.

Mininet emulates the network with virtual switches (Open vSwitch), hosts and links connecting them. These virtual components behave like physical network devices, allowing realistic testing of SDN concepts without requiring actual hardware.

IV. System Implementation and Operation

A. *Hardware Assembly*

The Arduino was configured through the three LEDS, the joystick, button, and temperature sensor according to Table III. The push button was connected to digital pin 2, while the joystick's X and Y axes were connected through analog pins A0 and A1 respectively, with the joystick switch on digital pin 3. The temperature sensor is connected to digital pin 7 and lastly, the RGB LED is connected to digital pins 8 (red), 9 (green), and 10 (blue), while the host status LEDs (h1 and h2) are connected to digital pins 5 and 6.

B. *Sofware components*

Mininet and Platform IO were installed in the Raspberry Pi for network simulation and Arduino development respectively. ONOS runs on a separate laptop due to ARM64 compatibility limitations on the Raspberry Pi.

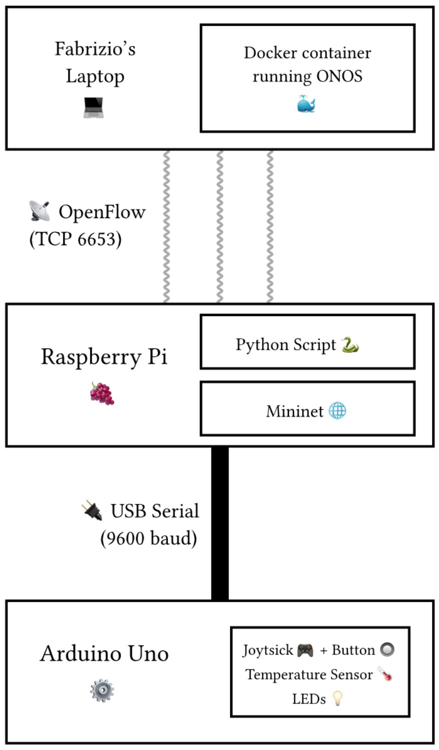


Fig. 1. Simple Flowchart of the system

TABLE III
Pin Assignment

Pin	Assignment	Function
2	INPUT	Button
4	INPUT	Joystick Switch
5	OUTPUT	LED h1
6	OUTPUT	LED h2
7	INPUT	Temp Sensor
9	OUTPUT	LED3 Green (RGB)
10	OUTPUT	LED3 Blue (RGB)
11	OUTPUT	LED3 Red (RGB)
A0	INPUT	Joystick X-axis
A1	INPUT	Joystick Y-axis

1) Mininet Installation (Raspberry Pi): Mininet was installed on the Raspberry Pi to create a virtual SDN network. It simulates network switches and hosts that connect to the ONOS controller via OpenFlow.

```
sudo apt update
sudo apt install mininet
```

Listing 1. Mininet Installation

To verify installation:

```
sudo mn --test pingall
```

Listing 2. Mininet Verification

2) Platform IO Configuration (Arduino Development): Platform IO was used for Arduino firmware development with the following configuration:

```
[env:uno]
platform = atmelavr
board = uno
framework = arduino
upload_port = /dev/ttyACM0
lib_deps = milesburton/DallasTemperature@
    ^4.0.5
monitor_speed = 9600
monitor_filters = default, send_on_enter
monitor_echo = yes
```

Listing 3. platformio.ini Configuration

3) ONOS Controller (Laptop): ONOS runs on the laptop via Docker since ARM64 is not supported on Raspberry Pi:

```
docker run -t -d \
  -p 8181:8181 \
  -p 8101:8101 \
  -p 6653:6653 \
  -p 5005:5005 \
  -p 830:830 \
  --name onos \
  onosproject/onos:2.7.0
```

Listing 4. ONOS Docker Deployment

Required ONOS applications were activated via the ONOS CLI:

```
docker exec -it onos /opt/onos/bin/onos

# Inside ONOS CLI:
app activate org.onosproject.openflow
app activate org.onosproject.fwd
exit
```

Listing 5. ONOS App Activation

The Web UI is accessible at `http://localhost:8181/onos/ui` with credentials: `onos/rocks`.

4) Python Dependencies (Raspberry Pi): The PySerial library was installed for serial communication with the Arduino:

```
pip3 install pyserial
```

Listing 6. Python Dependencies

C. Hardware Integration

1) Wiring Connections: The system uses the following communication links:

- **Arduino** ↔ **Raspberry Pi**: USB Serial cable (`/dev/ttyACM0` at 9600 baud)
- **Raspberry Pi** ↔ **Laptop**: WiFi network (both devices on same LAN)
- **Mininet** ↔ **ONOS**: OpenFlow protocol on port 6653

2) Starting Mininet with Remote Controller: To connect Mininet to the ONOS controller running on the laptop:

```
sudo mn --controller=remote,ip=LAPTOP_IP,
    port=6653 --topo=single,2
```

Listing 7. Mininet with Remote Controller

Replace `LAPTOP_IP` with the actual IP address of the laptop running ONOS.

D. Operational Behavior

The Python bridge script (`bridge.py`) runs on the Raspberry Pi and coordinates communication between the Arduino sensors and the Mininet virtual network. The script reads sensor events from the Arduino via serial communication and translates them into network commands that affect the virtual topology.

1) Communication Protocol: Table IV describes the message formats used for communication between the Arduino and the Raspberry Pi.

TABLE IV
COMMUNICATION PROTOCOL

Direction	Message	Action
	Arduino → Raspberry Pi	
Input	`JOY_UP`	Select Switch (s1)
Input	`JOY_LEFT`	Select Host h1
Input	`JOY_RIGHT`	Select Host h2
Input	`JOY_DOWN`	Select Both Hosts
Input	`BUTTON`	Toggle selected device state
Input	`TEMP:XX.X`	Temperature reading
	Raspberry Pi → Arduino	
Output	`LED1:GREEN`	h1 connected (LED ON)
Output	`LED1:RED`	h1 disconnected (LED OFF)
Output	`LED2:GREEN`	h2 connected (LED ON)
Output	`LED2:RED`	h2 disconnected (LED OFF)
Output	`LED3:GREEN`	Switch normal
Output	`LED3:BLUE`	Switch congested
Output	`LED3:RED`	Both hosts disconnected

2) Running the System: The system startup sequence is as follows:

1) **Start ONOS on laptop**:

```
docker start onos
```

2) **Start Mininet on Raspberry Pi**:

```
sudo mn --controller=remote,ip=
    LAPTOP_IP,port=6653 --topo=single
    ,2
```

3) **Run bridge script on Raspberry Pi** (in a new terminal):

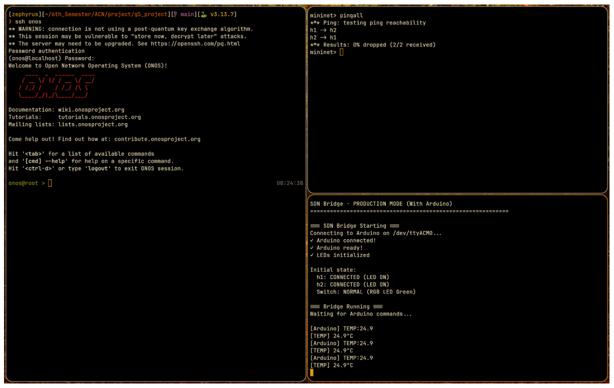
```
python3 bridge.py
```

3) Test Scenarios: The following scenarios can be tested to verify system functionality:

- **Scenario 1 - Link Failure (h1)**: Move Joystick left to select h1 → Press Button → LED1 turns OFF, `pingall` fails for h1
- **Scenario 2 - Link Restore (h1)**: With h1 selected → Press Button again → LED1 turns ON, `pingall` succeeds
  FOTO
- **Scenario 3 - Link Failure (h2)**: Move Joystick right to select h2 → Press Button → LED2 turns OFF, `pingall` fails for h2

- **Scenario 4 - Congestion**: Move Joystick up to select Switch $\rightarrow$ Press Button $\rightarrow$ LED3 turns BLUE, `iperf` shows reduced bandwidth (1 Mbps)
- **Scenario 5 - Clear Congestion**: With Switch selected $\rightarrow$ Press Button again $\rightarrow$ LED3 turns GREEN, bandwidth restored
- **Scenario 6 - Both Hosts Down**: Move Joystick down to select both hosts $\rightarrow$ Press Button $\rightarrow$ LED1 and LED2 turn OFF, LED3 turns RED
- **Scenario 7 - Temperature Monitoring**: Temperature readings are displayed in the console every 2 seconds

V. Prototype Demonstration



```
[zephyrus][~/6th_Semester/ACN/project/g5_project][ main][ v3.13.7]
❯ ssh onos
** WARNING: connection is not using a post-quantum key exchange algorithm.
** This session may be vulnerable to "store now, decrypt later" attacks.
** The server may need to be upgraded. See https://openssh.com/pq.html
Password authentication
(onos@localhost) Password:
Welcome to Open Network Operating System (ONOS)!

Documentation: wiki.onosproject.org
Tutorials:     tutorials.onosproject.org
Mailing lists: lists.onosproject.org

Come help out! Find out how at: contribute.onosproject.org

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'logout' to exit ONOS session.

onos@root >                                                    08:24:38
```

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet>
```

```
SDN Bridge - PRODUCTION MODE (With Arduino)
============================================================

=== SDN Bridge Starting ===
Connecting to Arduino on /dev/ttyACM0...
✓ Arduino connected!
✓ Arduino ready!
✓ LEDs initialized

Initial state:
  h1: CONNECTED (LED ON)
  h2: CONNECTED (LED ON)
  Switch: NORMAL (RGB LED Green)

=== Bridge Running ===
Waiting for Arduino commands...

[Arduino] TEMP:24.9
[TEMP] 24.9°C
[Arduino] TEMP:24.9
[TEMP] 24.9°C
[Arduino] TEMP:24.9
[TEMP] 24.9°C
```

Fig. 2. Workflow on the host laptop running docker

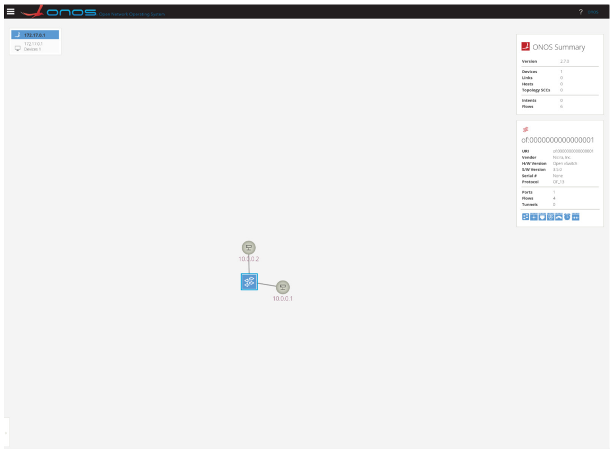
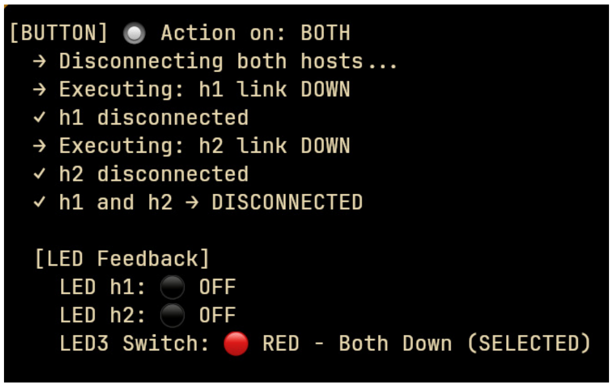


Fig. 3. Topology of the mininet network showed in ONOS web UI



```
[BUTTON] ⚪ Action on: BOTH
  → Disconnecting both hosts...
  → Executing: h1 link DOWN
  ✓ h1 disconnected
  → Executing: h2 link DOWN
  ✓ h2 disconnected
  ✓ h1 and h2 → DISCONNECTED

  [LED Feedback]
    LED h1: ⚫ OFF
    LED h2: ⚫ OFF
    LED3 Switch: 🔴 RED - Both Down (SELECTED)
```

Fig. 4. Output of the Python Script when taking action on both hosts

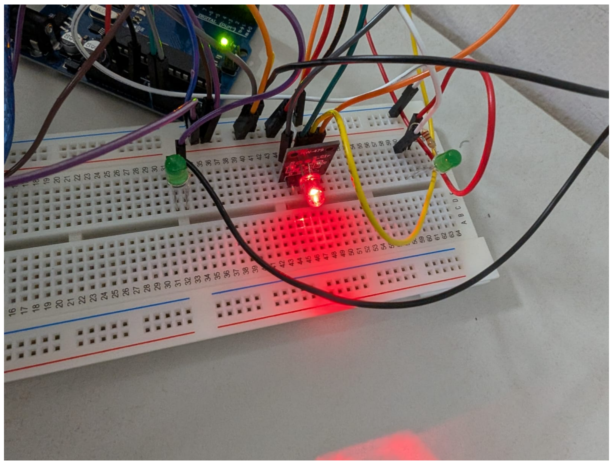
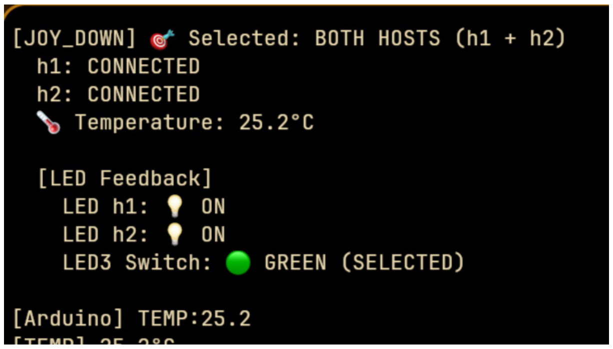


Fig. 5. Red LED indicating that there is no connection between hosts



```
[JOY_DOWN] 🎯 Selected: BOTH HOSTS (h1 + h2)
  h1: CONNECTED
  h2: CONNECTED
  🌡 Temperature: 25.2°C

  [LED Feedback]
    LED h1: 💡 ON
    LED h2: 💡 ON
    LED3 Switch: 🟢 GREEN (SELECTED)

[Arduino] TEMP:25.2
```

Fig. 6. Output of the python script showing the selection of both hosts

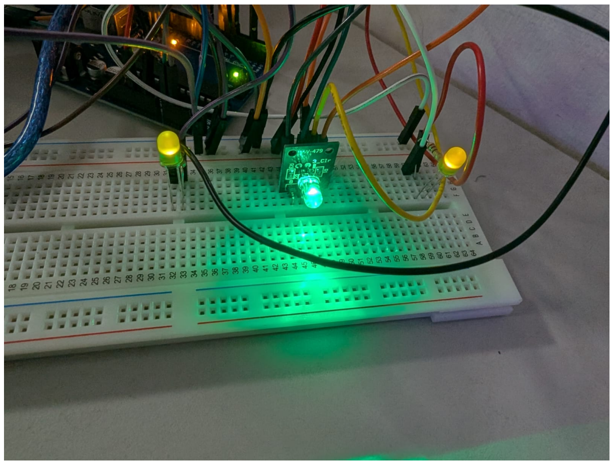


Fig. 7. Green LEDs showing that both hosts are connected

VI. Conclusion

This project successfully demonstrated the integration of physical hardware with Software-Defined Networking concepts through a functional prototype that connects an Arduino-based control panel to a Mininet virtual network managed by an ONOS controller.

The implemented system allows users to interact with network topology in real-time: selecting devices with a joystick, toggling link states with a button, and receiving immediate visual feedback through LEDs. This tangible interaction transforms abstract networking concepts—such as link failures, topology changes, and bandwidth congestion—into observable, hands-on experiences.

The modular architecture separating concerns between the Arduino (physical I/O), Raspberry Pi (bridge and network simulation), and laptop (SDN controller) proved effective for rapid prototyping and team collaboration. Each component could be developed and tested independently before integration.

With additional time and resources, this architecture could be extended into a comprehensive educational tool for networking courses. Potential enhancements include: expanding the topology to multiple switches, adding more sensors for diverse network scenarios, implementing a web dashboard for real-time monitoring, and creating guided laboratory exercises. The immediate cause-and-effect relationship between physical actions and network behavior makes this approach particularly valuable for students learning SDN fundamentals.

Source Code

The complete source code and setup instructions are available at:
https://github.com/Okikulo/SDN-Physical-Controller.git

Acknowledgments

The authors would like to thank the Department of Computer Science and Information Engineering at National Taiwan University of Science and Technology (NTUST) for providing the hardware resources used in this project. Special thanks to our professor for guidance throughout the Advanced Computer Networks course.