engineering proceedings

MDPI

Proceeding Paper

Building Application for Software-Defined Network †

Delyan Genkov *, Tsvetan Raykov and Miroslav Slavov

Department of Computer Systems and Technologies, Technical University—Gabrovo, 5300 Gabrovo, Bulgaria; tsvetan.raykov@gmail.com (T.R.); miroslav.st.slavov@gmail.com (M.S.)

* Correspondence: dgenkov@tugab.bg

† Presented at the International Conference on Electronics, Engineering Physics and Earth Science (EEPES 2025), Alexandroupolis, Greece, 18–20 June 2025.

Abstract

Software-defined networks are a modern approach to computer networks. With this concept, network devices can be monitored and configured centrally. While the lower layers of a software-defined network—devices and controllers—are relatively well known and standardized, the upper layers consist of APIs and software applications and are not standard. This article aims to propose one possible way to interact with a software-defined network and to build applications for monitoring and configuring such networks.

Keywords: software defined network; SDN; application; controller; pox; API

1. Introduction

Computer networks consist of many interconnected network devices—switches, routers, firewalls, etc. While network policy documents the rules and behavior of the whole network, the administrators need to convert these rules in particular configurations for different network devices and to apply these configurations to all the devices in the network one by one.

The software-defined networking approach suggests centralized configuration of all the network devices through a software application [1]. This application should allow definition of a policy, concerning the whole network, then to convert the policy to partial rules for different devices and to apply these rules simultaneously to all the devices. For achieving this, an intermediate software application, named controller, is used. The controller connects to all network devices. It collects information and receives events from the devices—i.e., when a new device connects to the network—along with its parameters—MAC address, IP address, etc. The controller maintains the topology of the whole network, runs the protocols and algorithms for optimal path selection and creates the forwarding tables of the devices. Network devices use the rules in their tables to forward packets.

Communication between the controller and network devices uses well known protocols. One of the most often used is OpenFlow—an open-source protocol, maintained by Open Networking Foundation [2]. There are other protocols, both open-source and proprietary. In this work, OpenFlow v.1.0 is used.

Communication with the controller is performed with the Northbound API, which is controller-specific. Most controllers have REST interfaces to interact with.

The application layer of the SDN consists of custom software components and is not standard [3,4]. There are many possible applications that could monitor and control a software-defined network. The goal of this work is to present a framework ap-

check for updates

Academic Editor: Grigor Mihaylov

Published: 11 September 2025

Citation: Genkov, D.; Raykov, T.; Slavov, M. Building Application for Software-Defined Network. *Eng. Proc.* **2025**, *104*, 92. https://doi.org/10.3390/engproc2025104092

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

plication that can interact with an SDN controller and to monitor and/or control a software-defined network.

2. Materials and Methods

There are many controllers available, both open source and commercial. In this work, the POX controller is used [5]. POX is a networking software platform written in Python 2.7.

POX supports OpenFlow protocol version 1.0.

Basically, POX is a collection of python scripts. It can be downloaded from Github repository [6]. In this work we use the mininet virtual network simulator [5]. It can create a virtual network, connecting hosts and switches, and it contains the POX controller as well as OpenFlow-enabled Open vSwitch [7]. Mininet is available as a linux virtual machine. To start it, the open-source virtualization platform Oracle VirtualBox is used [8]. After starting the virtual machine, we need to start mininet and create some virtual network topology. The following example creates a simple topology, consisting of one switch, three hosts and an external controller:

sudo mn –topo single,3 –switch ovsk –controller remote

The sample topology used is shown in Figure 1.

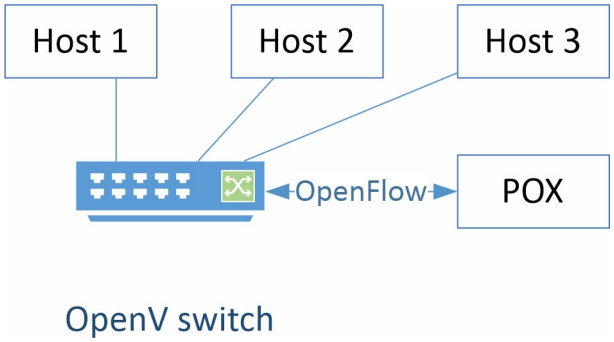


Figure 1. Sample network topology.

The POX controller code is located in ~/pox subdirectory of the default user's (mininet) home directory of the virtual machine. By default, it listens to TCP port 6653. Basically, custom source code can be located everywhere until it is reachable, but there is an "ext" subdirectory, which is the normal place for custom modules.

The custom module must contain a "launch" function, which is used for component initialization. This function can send command line parameters to the module.

Initially, we need to import some modules containing important functionality. The main POX object is named "core", and it serves as a central point for the POX's API. For interaction with the controller, we need to import the core object [4].

```
from pox.core import core
import pox.openflow.libopenflow_01 as of
import pox.lib.packet as pkt
from pox.lib.addresses import EthAddr, IPAddr
import pox.lib.revent as revent
import pox.lib.recoco as recoco
```

Other important imported objects are the OpenFlow 1.0 library, the objects for parsing and construction of network packets, the objects for handling MAC and IP addresses, the "revent" library for handling events and the "recoco" library for multitasking.

2.1. Event Handling

The event handling in POX is handled by the "revent" library and uses the publish/subscribe communication pattern. To subscribe to an event, creation of a callback function and registration with the source is necessary.

Common event types are

- packet_in(switch, port, packet)—a new packet enters the switch and it is not defined how to handle it;
- stats_in (switch, xid, pattern, packets, bytes)—statistics information about the packets and bytes sent and received from/to a switch;
- flow_removed (switch, pattern, packets, bytes)—a forwarding rule is removed from flow table for some reason, i.e., timeout expired;
- switch_join(switch)—a new switch appears in the network;
- port_change(switch, port, up)—a port of a switch changed its status.

2.2. Multitasking

When working with multiple devices, multiple events may occur simultaneously. The "recoco" library can implement multitasking using tasks and timers. A recoco task is an operation started with its own thread, and it is independent of the other tasks. Timers can start a task once at a predetermined time or periodically on a regular interval. This is useful if you must check for some conditions periodically.

2.3. Sending Messages to the Switches

In order to control the network behavior, the application needs to send messages to the network devices and to instruct them what to do in specific conditions. In POX, this is performed using the OpenFlow protocol. Some usable message types are

- ofp_packet_out—to instruct a switch where to send a packet;
- ofp_flow_mod—flow table modification;
- ofp_stats_request—requesting statistics from switches, etc.

Most OpenFlow messages are described in OpenFlow Switch Specification [9,10].

2.4. Matching Packets to Determine the Action

According to OpenFlow protocol, the matching rules for a packet can include the attributes shown in Table 1 [4].

Table 1. Attributes of a packet.

Attribute	Meaning
n_port	Switch port number the packet arrived on
dl_src	Ethernet source address
dl_dst	Ethernet destination address
dl_vlan	VLAN ID
dl_vlan_pcp	VLAN priority
dl_type	Ethertype/length (e.g., 0x0800 = IPv4)
nw_tos	IP TOS/DS bits
nw_proto	IP protocol (e.g., 6 = TCP) or lower 8 bits of ARP opcode
nw_src	IP source address
nw_dst	IP destination address
tp_src	TCP/UDP source port
tp_dst	TCP/UDP destination port

This is also known as 12-tuple and presents all the parameters of a packet that can be matched for packet type and source determination. For example, if we need to match the source IP address, we can use the following code:

my_match = of.ofp_match()

my_match.nw_src = "192.168.1.0/24"

A single parameter or a group of parameters simultaneously can be matched, and particularly matches can be made; for example, the first three numbers from the IP address must match, and the last one is not important.

2.5. Working with Packets

The "packet" library in POX is used for reading the packet's header parameters or to construct and send some packets. As different packet types have different structures, the library contains common types of packets and their fields for the most common protocols, like ethernet, IPv4, ICMP, TCP and others.

For every protocol, the containing fields are defined; for example, for an Ethernet packet, the fields are

- dst (EthAddr)—the destination MAC address;
- src (EthAddr)—the source MAC address;
- type (int)—the ethertype or ethernet length field.

3. Results and Discussion

In this work, a framework application is created, which communicates with the POX controller, captures events from the example network and simply registers the events in the system log. The application start in the mininet virtual machine is shown at Figure 2.

```
mininet@mininet-vm:~/pox/ext$ ../pox.py sample_app
POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al.
WARNING:sample_app:The sample application is running...
WARNING:version:Support for Python 3 is experimental.
INFO:sample_app:application ready
INFO:sample_app:Packet received:
INFO:sample_app:Source IP:10.0.0.1
INFO:sample_app:Destination IP:10.0.0.2
INFO:sample_app:Source MAC:00:00:00:00:00:01
INFO:sample_app:Destination MAC:00:00:00:00:00:02
INFO:sample_app:Packet Type:ICMP
INFO:sample_app:In Port:1
INFO:core:POX 0.7.0 (gar) is up.
```

Figure 2. Starting the sample application.

The application is created using python programming languages and the libraries described above—libopenflow_01, packet, recoco and revent. In the initialization phase, it registers functions to handle the necessary events—packet_in, stats_in, flow_removed, switch_join and port_change. Then, when an event occurs, the callback function is invoked, and the corresponding logic is executed. The logic depends on the purpose of the application created, it will be different for a layer 2 switch and for a layer 3–4 firewall, for example.

The application discovers the network topology and creates a table in memory, containing all the switches, ports and connections. It detects network changes—when a new switch or host is connected, or an existing one is removed and updates the topology table accordingly. When a packet is received, the application recognizes all the twelve attributes, shown in Table 1, and increments the counters for the corresponding flow. In case of a new flow recognition, it creates a new entry in the flow table and then forwards the packet

processing to the normal POX logic. It also periodically requests and receives statistics from every switch and stores the information in the statistics table.

There is also a rules table, which at this stage is created by reading an external file during initialization stage. This table contains the disallowed communications between host pairs, based on MAC or IP addresses, thus allowing the application to execute functions of a simple layer 2 or layer 3 firewall.

The application's simplified logic is shown in Figure 3.

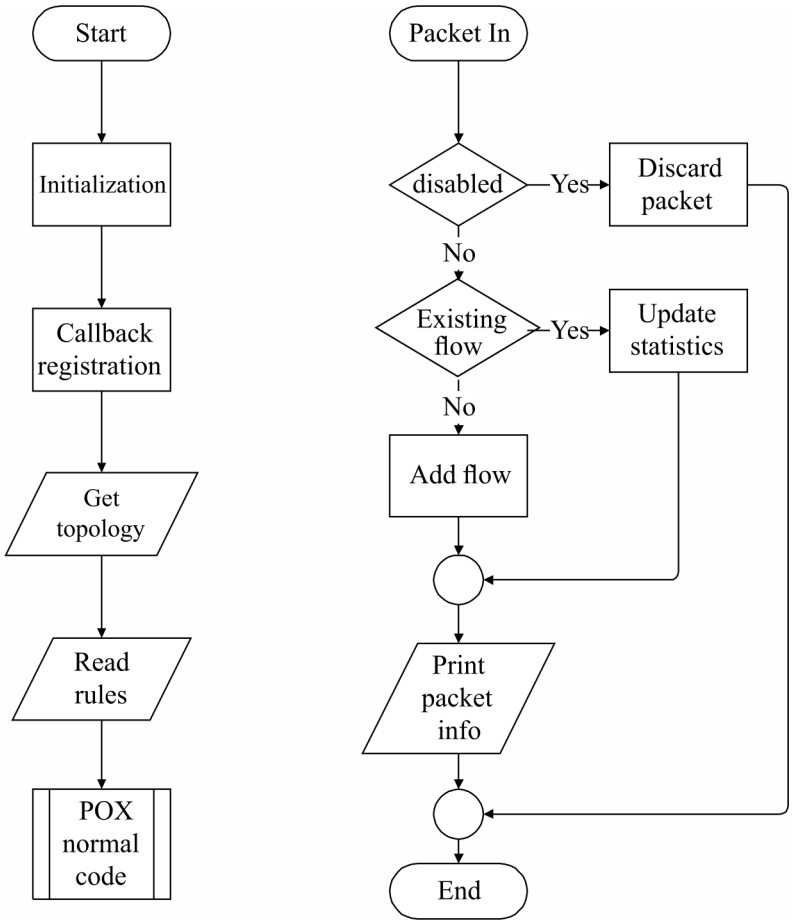


Figure 3. Application flowchart.

On the left side of the diagram, the main application logic is shown. On the right side, the logic of the callback function for packet_in event is shown. The other events are captured in the same way. Functions for sending commands to the switches are also implemented for instructions to the switch on where to send a packet, inserting, changing or removing flow table entry and for requesting statistics from switches.

4. Conclusions

A sample software-defined network application is created that implements the major functionality necessary for developing various SDN applications. It currently uses the POX controller, which is a significant limitation for future usage.

Future plans of the team include creating different applications for software-defined networks, like network monitoring applications, network firewalls, load balancers and quality-of-service applications, which will use this application as a foundation. We also plan to work with different controllers to implement networks with redundant controllers and to use real devices, supporting SDN, to create a real, scalable, centrally monitored and controlled software-defined network.

Author Contributions: D.G., T.R. and M.S. were involved in the full process of producing this paper, including conceptualization, methodology, programming and preparing the manuscript. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the University Center for Research and Technology at the Technical University of Gabrovo, project NIP2025-9, "Study and Development of Software Defined Networks".

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The raw data supporting the conclusions of this article will be made available by the authors upon request.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

SDN	Software-Defined Network
API	Application Programming Interface
MAC	Media Access Control
IP	Internet Protocol
REST	Representational State Transfer

References

1. Hussain, M.; Shah, N.; Amin, R.; Alshamrani, S.S.; Alotaibi, A.; Raza, S.M. Software-Defined Networking: Categories, Analysis, and Future Directions. *Sensors* **2022**, *22*, 5551. [CrossRef] [PubMed]
2. Open Networking Foundation. OpenFlow. Available online: https://opennetworking.org/sdn-resources/customer-case-studies/openflow/ (accessed on 10 April 2025).
3. Open Networking Foundation. SDN Architecture. Available online: https://opennetworking.org/wp-content/uploads/2013/02/TR_SDN_ARCH_1.0_06062014.pdf (accessed on 10 April 2025).
4. Bonanni, M.; Chiti, F.; Fantacci, R.; Pierucci, L. Dynamic Control Architecture Based on Software Defined Networking for the Internet of Things. *Future Internet* **2021**, *13*, 113. [CrossRef]
5. POX Documentation. Available online: https://noxrepo.github.io/pox-doc/html/ (accessed on 10 April 2025).
6. POX. Available online: https://github.com/noxrepo/pox (accessed on 10 April 2025).
7. Mininet. Available online: https://mininet.org/ (accessed on 10 April 2025).
8. Linux Foundation. Open vSwitch. Available online: https://www.openvswitch.org/ (accessed on 10 April 2025).
9. Oracle. Virtual box—Powerful Open-Source Virtualization. Available online: https://www.virtualbox.org/ (accessed on 10 April 2025).
10. Open Networking Foundation. OpenFlow Switch Specification. Available online: https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf (accessed on 10 April 2025).

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.