ACTA TECHNICA NAPOCENSIS
Electronics and Telecommunications

DEVICE LEVEL PROGRAMMABILITY USING RESTCONF PROTOCOL: AN INTRODUCTORY APPROACH

Vlad-Andrei MONORANU[1], Tudor-Mihai BLAGA[1], Virgil DOBROTA[1]
[1]*Communications Department, Technical University of Cluj-Napoca, Romania*
Corresponding author: Virgil Dobrota (e-mail: Virgil.Dobrota@com.utcluj.ro)

Abstract: **This paper presents an implementation of a virtualized testbed emulated with GNS3 version 2.2.19, controlled by Postman API client and by Python3 scripts. A GNS3 VM (virtual machine) running Ubuntu acted as remote server for hosting the network devices images. The aim was to involve RESTCONF (Representational State Transfer Configuration Protocol) which uses HTTP requests, as an alternative solution for NETCONF. The scenario included one Cisco IOS-XE-based virtual router CSR1000v 16.6.7 with RESTCONF enabled. By using a GET request on a specified URL, a list of capabilities of the router could be displayed. In the first scenario different other types of requests (PUT, POST, PATCH, and DELETE) have been checked, too. Another two Cisco CSR1000v routers were added in the network topology and their manual configuration was approached in the second scenario. Following static IPs allocations the behavior of the RESTCONF configured router was tested. As proof of feasibility, we enabled OSPF (Open Shortest Path First) routing protocol on all routers, assisting the discovery of neighboring devices. The major outcome of this project was that we were able to get long lists of capabilities that can be split in individual items and reused later for automatic configuration.**

Keywords: *Cisco Router; GNS3; RESTCONF.*

I. INTRODUCTION

The network configuration management is the continuing process of supervising the installation and upkeep of all network devices which may be administrated through several options. Most of the network engineers are familiar with CLI (Command Line Interface). This is an interface found on network devices (e.g. routers, switches) and allowing their management and control. It processes commands to a computer program in the form of lines of text. In the case of automation, the CLI does not meet the requirements for strict data structures. NETCONF (Network Configuration Protocol) is an alternative solution for it as it allows the management of network devices through RPC-s (Remote Procedure Calls) which contain well-defined XML (Extensible Markup Language) messages [1]. In this paper we preferred to use HTTP that provides a programmatic interface for accessing data defined in YANG, using the datastore concepts defined in the NETCONF [2]. This protocol is called RESTCONF (Representational State Transfer Configuration Protocol). We enabled it on a Cisco CSR1000v router which runs on a VMware virtual machine through GNS3 emulator. Its functionality was presented with the help of Postman. The objective of this work is to get a list of capabilities of the network devices, in order to use them later for automatic configuration,

The rest of the paper is organized as follows: Section II discusses the related work, followed by an overview of RESTCONF. Section IV presents the implementation and experimental results. Last section includes conclusions and future work.

II. RELATED WORK

The IT infrastructure scalability may be helped by network automation even if it executes complicated analyses based on inputs from the many devices accessible inside your network. The practice of using software to automate the network, the security provisioning, and the administration in order to continually enhance network efficiency and functionality is known as network automation. Network virtualization is frequently used along network automation. When it comes to deploying and administrating both conventional and cloud-native applications, IT departments are looking for scalability, agility, and consistency [3].

Data centers, service providers, and businesses can use hardware and software-based solutions to automate their networks, increasing productivity, reducing human error, and lowering operational costs. The rise in IT expenditures for network operations is one of the most pressing concerns for network administrators. Data and device growth are outpacing IT capabilities, rendering manual procedures practically unfeasible. Despite this, up to 95% of network modifications are done manually, resulting in operating expenses that are two to three times greater than the network's cost. Businesses must increase their IT automation, which must be handled centrally and remotely, in order to keep up with the digital world [4].

Paper [5] discusses how embedded device resource constraints, heterogeneity, and network dynamics must all be taken into account while developing Internet of Things (IoT) management systems. From the network provider's point of view conventional standards-based solutions may be insufficient, the use of proprietary platforms being requested. On the other hand developers must integrate many platforms and work with a range of APIs. This creates significant issues, such as supporting heterogeneous devices and integrating open and proprietary management solutions, as well as managing multi-technology, multi-vendor, and multi-standard

Manuscript received August 1, 2021; Revised December 2, 2021

systems. With these considerations in mind, the Internet Engineering Task Force established a number of standards aimed at integrating and interoperating heterogeneous devices, including the Representational State Transfer Configuration Protocol. This article concludes that alongside other standard management protocols that are applicable to IoT Systems, RESTCONF might potentially be a useful alternative in the case of non-constrained devices.

A two-stage technique for automated RESTCONF agent conformity testing is provided in [6]. To generate and to issue requests, as well as to receive and to interpret the answers, the Python curl package was utilized. A flaw in unittest (i.e. a unit testing framework) was discovered is the absence of capability for constructing data-driven generic tests. A technique for creating unittest cases based on the metadata required to be supplied. The tool was tested against an agent under development, and the results were quite positive, since it was able to find errors at a very cheap cost.

In [7] the authors describes a multi-layer Programmable Optical Network with SDN capabilities (PROnet). This is a two-layer Research and Education Network (REN) deployed at and around the UT Dallas campus. It uses the PROnet Orchestrator, having a Gaussian-Noise-model-based quality of transmission estimator module incorporated. It demonstrated its capacity to forecast signal to noise ratio and optimal transmission power. The PROnet Orchestrator involves the quality of transmission estimator module to quickly and efficiently provide light paths in the optical network for host-to-host large data transmission. In order to identify network topology and nodes, the Resource Management module interacts with Ethernet and optical nodes using Rest APIs supplied by Ethernet controllers and RESTCONF optical plugin. The Orchestrator uses this protocol to detect the WDM topology and supply light paths.

Paper [8] demonstrates that Network Functions Virtualization (NFV) created a new approach to develop and deploy network security services, but without a standard interface between them. Thus it may fail to create a viable ecosystem that seamlessly integrates network security services. The proposed architecture enables users to define their security needs in a user-friendly way by offering high-level security interfaces that do not need detailed knowledge of network resources and protocols. A web server is designed to offer to the administrator a more accessible solution. A few web sites provide administrators the ability to define high-level security rules using user interfaces. A new network protocol such as RESTCONF provides a programmatic interface through HTTP to retrieve data described in a YANG model [19]. Note that YANG is a data modeling language for designing configuration and operating functions. It determines the scope and the type of functions that can be performed by NETCONF and RESTCONF APIs. Back to [8], a communication channel based on RESTCONF is built to allow interaction between the NSF client and the Security management system. Furthermore, because the NSF client is built on web applications in this design, RESTCONF is favored over the Network Configuration Protocol (NETCONF).

Authors in [9] present a technique for subscribing and pushing updates to the YANG datastore, which improves the performance of prior technologies. RESTCONF is used to standardize the REST API structure amongst SDN controllers, which is specified by the underlying YANG data stores. There are articles which show that these components can widely extend their applicability in domains other than networking. In [10] the authors are implementing a way of remotely controlling a car where the data related with the automobile was modelled, managed, and controlled using NETCONF/ RESTCONF and YANG.

In paper [11] it is specified that the open source operating system OpenWrt has been a popular choice for replacing proprietary firmware on networking devices like residential routers and access points in past years. To customize an OpenWrt system, such as putting up firewall rules, the user must either login in to the web interface or use SSH to alter configuration files on the device manually. RESTCONF is the chosen solution for managing the configurations, this being compared to other similar implementations.

III. OVERVIEW OF RESTCONF

REpresentational State Transfer (REST) is an architectural approach for establishing standards across web-based computer systems, allowing them to interact more easily. REST-compliant systems, often known as RESTful systems, are distinguished by their statelessness and separation of client and server concerns.

Separation of Client and Server

The client and server implementations under the REST architectural style can be managed separately of one another, without knowing about each other. This implies that the client's code can be updated at any moment without impacting the server's operation, and the server's code may be changed without affecting the client's operation.

They may be maintained modular and independent as long as one side knows what type of communications to transmit to the other. It increases the flexibility of the interface over platforms and improve scalability by isolating the user interface issues from the data storage issues by simplifying the structuring. Furthermore, the feature enables each component to develop independently. Multiple users access the same REST endpoints, execute the same operations, and receive the same replies when exploiting a REST interface.

Statelessness of REST

The REST architecture is stateless, which means the server does not have to recognize what state the client is in or the client does not have to know about the state of the server. Both the server and the client may interpret any message received in this form, even if they have not seen prior messages. The use of resources, rather than instructions, enforces the statelessness condition. The resources are nouns of the Web for communicating, which represent any item, document, or thing that the user would want to save or transfer to other services. REST systems do not require the development of interfaces because they communicate through normal operations on resources. These restrictions enhance RESTful applications achieving stability, speed, and scalability by acting as elements that can be controlled, changed, and reused without impacting the system as a whole, even while it is in use.

In order to communicate with the server, the clients submit requests to retrieve or change resources, and servers respond to these requests using the REST architecture. RESTful APIs are web service APIs that follow the REST architectural restrictions. The characteristics of HTTP-based RESTful APIs [12] are the following:

- http://api.example.com/ is an example of a base URI.
- GET, POST, PUT, and DELETE are examples of typical HTTP methods.
- A media type that defines data components for state transitions

RESTCONF provides RESTful APIs using structured data (XML or JSON) and YANG, allowing programmatic access to various network devices. HTTPS approaches are used by RESTCONF APIs, while YANG is being used as data modeling language for designing configuration and operating functions. YANG determines the scope and the type of functions that can be performed by the RESTCONF APIs [12]. The RESTCONF major components are depicted in Figure 1 [3].

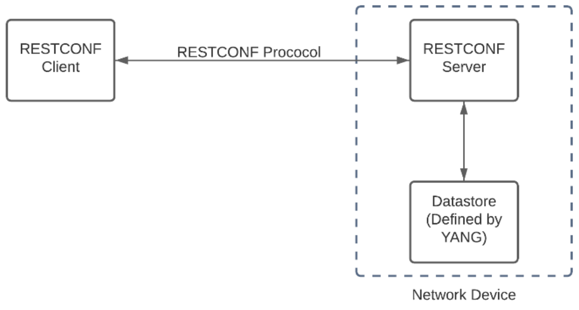


Figure 1. RESTCONF components

Being a stateless protocol that provides CRUD (CREATE, READ, UPDATE, DELETE) operations it uses HTTPS methods which are shown in Table 1 [6].

Options	**Methods**
GET	Read
PATCH	Update
PUT	Create or Replace
POST	Create or Operations (reload, default)
DELETE	Deletes the targeted resource
HEAD	Header metadata (no response body)

Table 1. RESTCONF operations

Using Postman, these HTTPS requests can be executed with less tedious work. Postman is a popular, free of charge API client, which offers a simple user interface for the consumers that would like to dissect different RESTful APIs without the need for coding [13]. This tool also has the capacity of storing environments for later use and of displaying the responses back. In this way, the process of semi-automating the router configurations is enhanced.

IV. IMPLEMENTATION AND EXPERIMENTAL RESULTS

This paper presents an implementation of a virtualized testbed emulated with GNS3 version 2.2.19 [20], controlled by Postman API client and by Python3 scripts. A GNS3 VM (virtual machine) running Ubuntu acted as remote server for hosting the network devices images. The aim was to involve RESTCONF as an alternative solution for NETCONF. The principle is presented in Figure 2.

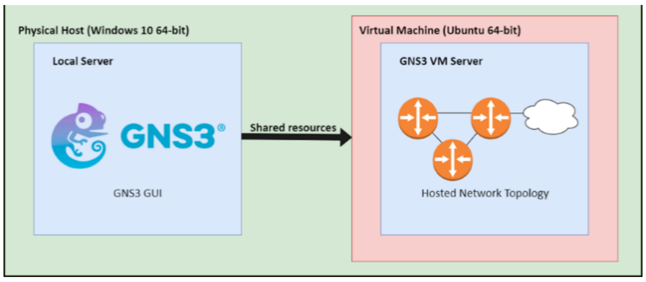


Figure 2. Principle of the testbed scenario

The detailed scenario (see Figure 3) included one Cisco IOS-XE-based virtual router CSR1000v 16.6.7 with RESTCONF enabled [18]. By using a GET request on a specified URL, a list of capabilities of the router were obtained and displayed. In the first scenario different other types of requests (PUT, POST, PATCH, and DELETE) have been checked, too. Another two Cisco CSR1000v routers were added in the network topology and their manual configuration was approached in the second scenario.

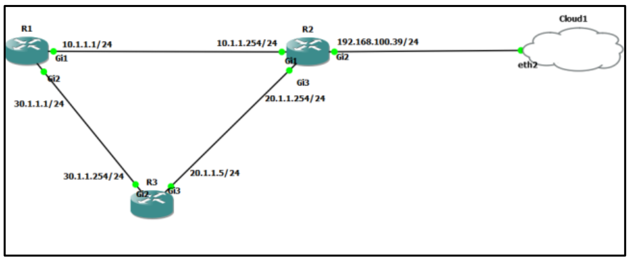


Figure 3. GNS3 network topology

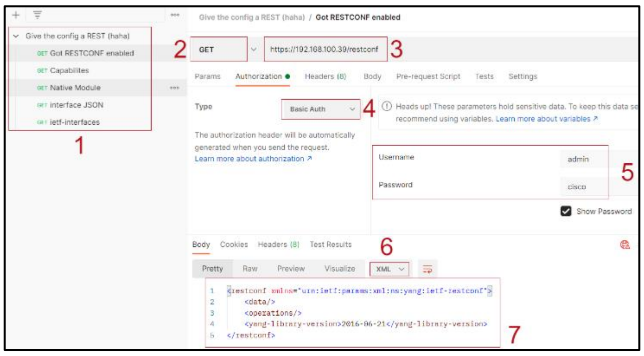
It is out of the scope of this paper to discuss step-by-step the installation of GNS3, VMware Workstation 16 Player and GNS3 VM, Cisco CSR1000V virtual routers etc. Details are presented in [15], [16].

This section presents now the scenarios that demonstrated how RESTCONF protocol works, various request types using PyCharm [21] and Postman, how to check the connectivity between network devices, and packet analysis using Wireshark.

A. Experiments with Postman and Python HTTP requests

When it comes to RESTCONF, there are several modules to choose from. Information can be obtained from each of these modules. For example, we may view all interfaces, including those that are based on the IETF standard, or we can establish a new interface. As an alternative, CLI is available, but more convenient is to use a GUI-based tool allowing to call these URLs. Postman allows us to accept URLs and then send instructions from the GUI. It also has great capacity to let us modify and tune what goes in the header for those requests or what data types we are expecting. Before we proceeded, a new free account had to be registered on the Postman official website: https://www.postman.com/. Figure 4 offers a brief explanation of the Postman GUI:

1. A new workspace and a new collection were created. The collection was used to keep together the request and the data received as a response.
2. The type of request can be selected from a drop-down menu (e.g. GET herein).
3. Regarding of what information we want to retrieve, the specific URL was introduced.
4. The type of authentication can be selected from a drop-down menu (in this case Basic Authentication).
5. The username and the password are introduced just as in CLI, mentioning that they are stored.
6. The encoding format can be selected (i.e. XML).
7. The body is the response received after calling the method on the specified URL (herein the response was provided by a GET request).

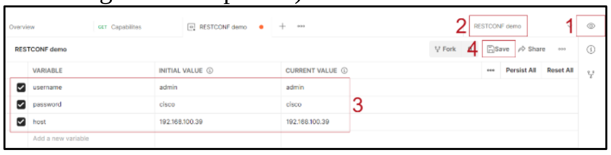

Give the config a REST (haha) / Got RESTCONF enabled
Give the config a REST (haha)
GET Got RESTCONF enabled
GET Capabilites
GET Native Module
GET Interface JSON
GET ietf-interfaces
1
2
GET
https://192.168.100.39/restconf
3
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Type
Basic Auth
4
Heads up! These parameters hold sensitive data. To keep this data se
recommend using variables. Learn more about variables
The authorization header will be automatically generated when you send the request.
Learn more about authorization
Username
admin
Password
cisco
5
Show Password
Body
Cookies
Headers (8)
Test Results
6
Pretty
Raw
Preview
Visualize
XML
7

```
<restconf xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf">
    <data/>
    <operations/>
    <yang-library-version>2016-06-21</yang-library-version>
</restconf>
```

Figure 4. Postman GUI components

There are hundreds of modules that may be called by using RESTCONF. Some are Cisco-specific, others are versions of the IOS that we are running, and some are IETF generic. The number of things we can call on or seek for is virtually endless. A GET request was used again to obtain all the capabilities of our router. In addition, the environment variables option provided by Postman were leveraged to ease the job for automation. This option allowed us to create variables that stored values. Later on they were used to replace the repetitive tasks (typing username, password or host etc.). The environment was called RESTCONF demo (see Figure 5):

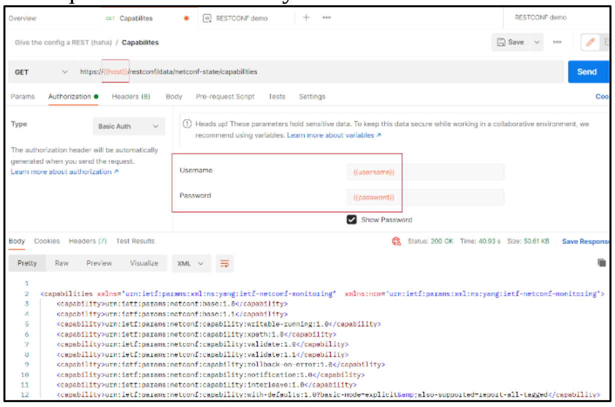
1. The environment was created from "eye".
2. The needed was selected from the list of environments.
3. The variables were created and their values were obtained from the user input.
4. The environment was then saved (otherwise the settings are not updated).



	VARIABLE	INITIAL VALUE	CURRENT VALUE
✓	username	admin	admin
✓	password	cisco	cisco
✓	host	192.168.100.39	192.168.100.39

Figure 5. Creating an environment in Postman

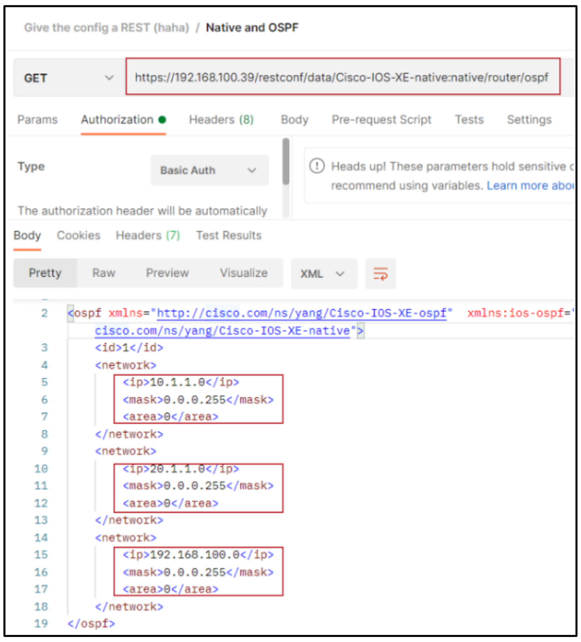
The way the variables are used is shown in Figure 6. The variable's name is put between double curly brackets to reference the value (e.g. {{variable}}). There was a significant number of modules, therefore the outcome of this request was not entirely visible.


Overview
GET Capabilites
RESTCONF demo
Give the config a REST (haha) / Capabilites
Save
GET
https://{{host}}/restconf/data/netconf-state/capabilities
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Type
Basic Auth
Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. Learn more about variables
The authorization header will be automatically generated when you send the request.
Learn more about authorization
Username
{{username}}
Password
{{password}}
Show Password
Body
Cookies
Headers (7)
Test Results
Status: 200 OK
Time: 40.93 s
Size: 50.61 KB
Save Response
Pretty
Raw
Preview
Visualize
XML

```
<capabilities xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-monitoring"  xmlns:ncm="urn:ietf:params:xml:ns:yang:ietf-netconf-monitoring">
    <capability>urn:ietf:params:netconf:base:1.0</capability>
    <capability>urn:ietf:params:netconf:base:1.1</capability>
    <capability>urn:ietf:params:netconf:capability:writable-running:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:xpath:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:validate:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:validate:1.1</capability>
    <capability>urn:ietf:params:netconf:capability:rollback-on-error:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:notification:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:interleave:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:with-defaults:1.0?basic-mode=explicit&amp;also-supported=report-all-tagged</capability>
```

Figure 6. Variables to get the modules of RESTCONF

OSPF (needed in the second scenario) was enabled on all routers of the topology. When we tried to retrieve OSPF-related information, we compared the output of the SSH Client with the output of Postman (Figures 7 and 8). Note that the path to make a request was: https://192.168.100.39/restconf/data/Cisco-IOS-XE-native:native/router/ospf.


Give the config a REST (haha) / Native and OSPF
GET
https://192.168.100.39/restconf/data/Cisco-IOS-XE-native:native/router/ospf
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Type
Basic Auth
Heads up! These parameters hold sensitive c
recommend using variables. Learn more abo
The authorization header will be automatically
Body
Cookies
Headers (7)
Test Results
Pretty
Raw
Preview
Visualize
XML

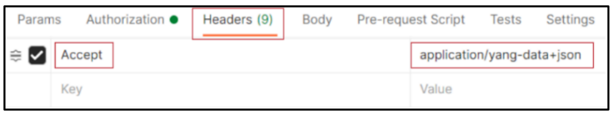
```
<ospf xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-ospf"  xmlns:ios-ospf="
cisco.com/ns/yang/Cisco-IOS-XE-native">
    <id>1</id>
    <network>
        <ip>10.1.1.0</ip>
        <mask>0.0.0.255</mask>
        <area>0</area>
    </network>
    <network>
        <ip>20.1.1.0</ip>
        <mask>0.0.0.255</mask>
        <area>0</area>
    </network>
    <network>
        <ip>192.168.100.0</ip>
        <mask>0.0.0.255</mask>
        <area>0</area>
    </network>
</ospf>
```

Figure 7. OSPF networks enabled on R2 in Postman

```
R2#show ip ospf interface brief
Intf  PID  Area  IP Addr/Mask     Cost  St  Nbrs
F/C
Gi2   1    0   192.168.100.39/24  1    DR    0/0
Gi3   1    0   20.1.1.254/24      1    DR    0/0
Gi1   1    0   10.1.1.254/24      1    DR    0/0
```

Figure 8. OSPF interfaces on R2 in SSH Client

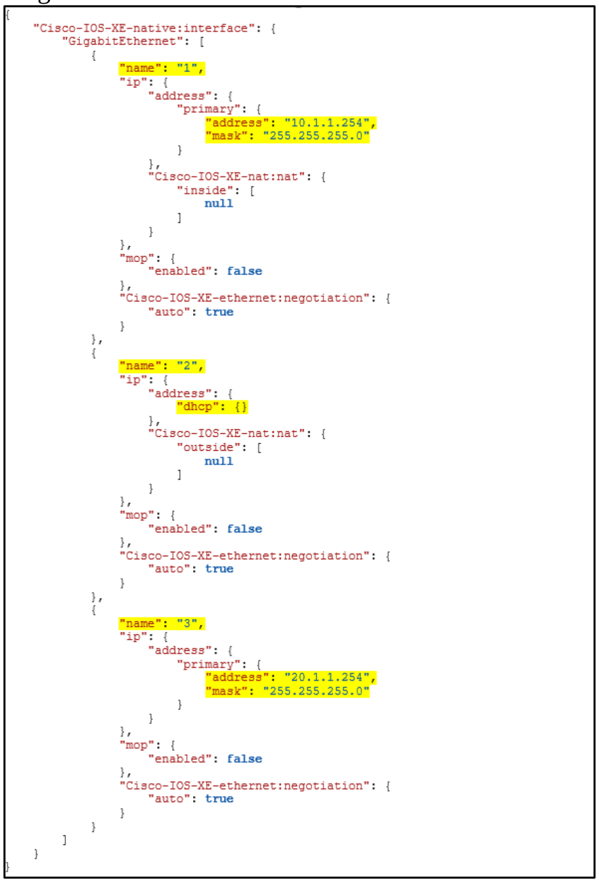
The returned answer, which was initially delivered back in XML format, was easily converted to JSON encoding using Postman. In the Headers tab, the value `application/yang-data+json` should have the Accept Key (Figure 9).



```
Accept          application/yang-data+json
```

Figure 9. JSON encoding Postman settings

Next, information about the network interfaces available on the router had to be retrieved (see Figure 10). The response was then compared to the console output, as in Figure 11.



```
{
    "Cisco-IOS-XE-native:interface": {
        "GigabitEthernet": [
            {
                "name": "1",
                "ip": {
                    "address": {
                        "primary": {
                            "address": "10.1.1.254",
                            "mask": "255.255.255.0"
                        }
                    },
                    "Cisco-IOS-XE-nat:nat": {
                        "inside": [
                            null
                        ]
                    }
                },
                "mop": {
                    "enabled": false
                },
                "Cisco-IOS-XE-ethernet:negotiation": {
                    "auto": true
                }
            },
            {
                "name": "2",
                "ip": {
                    "address": {
                        "dhcp": {}
                    },
                    "Cisco-IOS-XE-nat:nat": {
                        "outside": [
                            null
                        ]
                    }
                },
                "mop": {
                    "enabled": false
                },
                "Cisco-IOS-XE-ethernet:negotiation": {
                    "auto": true
                }
            },
            {
                "name": "3",
                "ip": {
                    "address": {
                        "primary": {
                            "address": "20.1.1.254",
                            "mask": "255.255.255.0"
                        }
                    }
                },
                "mop": {
                    "enabled": false
                },
                "Cisco-IOS-XE-ethernet:negotiation": {
                    "auto": true
                }
            }
        ]
    }
}
```

Figure 10. Network interfaces available on the router via Postman

The path for the call in Figure 10 was `https://192.168.100.39/restconf/data/Cisco-IOS-XE-native:native/interface`.

```
R2#show ip interface brief
Intf IP-Address     OK? Method Status  Prot.
Gi1  10.1.1.254     YES NVRAM  up      up
Gi2  192.168.100.39 YES DHCP   up      up
Gi3  20.1.1.254     YES NVRAM  up      up
```

Figure 11. Network interfaces available on the router via SSH

The interface GigabitEthernet4 (Gi4) was also present on the router but it did not have an IP address allocated. Using a PATCH request, this time via Python, a new IP address was set to 50.1.1.1, with mask 255.255.255.0 (Figure 13). The other interfaces could have been set much easier using this method. The variable `url` stored the path to the network interface which was configured. `auth` stored the credentials of the users, whilst `headers` was used to display the result in a JSON format and to use the data in the same format as the information patched directly on the router. Variable `verify` was set to false to ignore the SSL (Secure Sockets Layer) verification. After the requests library was imported, the request was done (Figure 12). Note that `InsecureRequestWarning` was used to disable warnings in requests' vendored urllib3. This meant that there were no Python warnings on the output console.

```
# RESTCONF Interface Configuration
import requests
import json
from urllib3.exceptions import
InsecureRequestWarning
requests.packages.urllib3.disable_warnings(ca
tegory=InsecureRequestWarning)
response = requests.patch(
   url =
'https://192.168.100.39/restconf/data/Cisco-
IOS-XE-
native:native/interface/GigabitEthernet=4',
   auth = ('admin', 'cisco'),
   headers = {
      'Accept': 'application/yang-data+json',
      'Content-Type': 'application/yang-
data+json'
   },
   data = json.dumps({
      'Cisco-IOS-XE-native:GigabitEthernet':
{
         'ip': {
            'address': {
               'primary': {
                  'address': '50.1.1.1',
                  'mask': '255.255.255.0'
               }
            }
         }
      }
   }),
   verify = False)
# Printing and verifying the HTML response
code
print('Response Code: ' +
str(response.status_code))
if (int(response.status_code / 100) == 2):
   print ("Success!")
else:
   print ("Failed!")
```

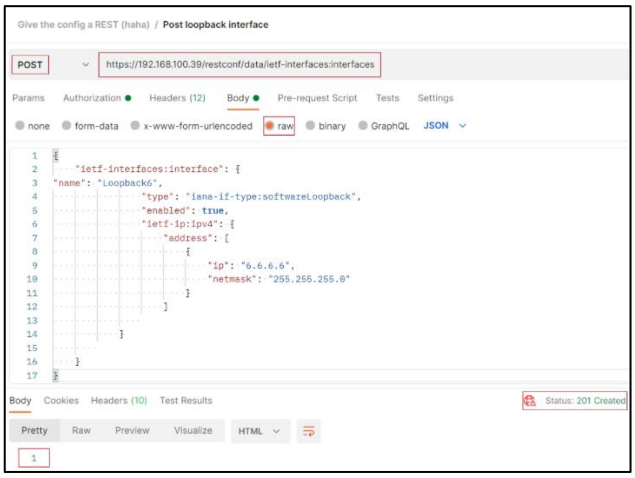
Figure 12. Python PATCH request

After the request was done, information about the new interface settings was displayed in the SSH terminal (see Figure 13). In the PyCharm console a code was printed. The status code differed from one type of request to another and the acknowledgement that the operation was successfully done was given by a 3-digit number starting with the digit 2. If the operation failed, a 3-digit number starting with the digit 4 was displayed.

```
Jun 6 15:55:50.352: %DMI-5-CONFIG_I:  F0:
nesd:  Configured from NETCONF/RESTCONF by
admin, transaction-id 2913
R2#show ip interface brief
Intf IP-Address      OK? Method Status   Prot.
Gi1  10.1.1.254      YES NVRAM  up       up
Gi2  192.168.100.39 YES DHCP   up       up
Gi3  20.1.1.254      YES NVRAM  up       up
Gi4  50.1.1.1        YES other  adm.down down
```

Figure 13. IP allocation through Python request checking

Then we used Postman to add a loopback virtual interface. Being unique, the machine communicates with itself via this logical device [14]. This is mostly used for diagnostics and troubleshooting, as well as for connecting to local servers. After the GET request at the path `https://192.168.100.39/restconf/data/ietf-interfaces:interfaces`, a JSON object of the returned interfaces was simply copied and modified in order to fulfill the task. In the body of a POST request with the same path, the object was pasted and the request was made (Figure 14). This interface had the IP address 6.6.6.6 with mask 255.255.255.0.



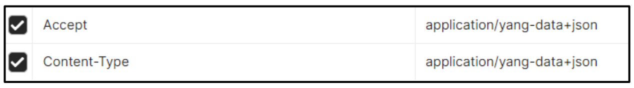
```
{
    "ietf-interfaces:interface": {
"name": "Loopback6",
                "type": "iana-if-type:softwareLoopback",
                "enabled": true,
                "ietf-ip:ipv4": {
                    "address": [
                        {
                            "ip": "6.6.6.6",
                            "netmask": "255.255.255.0"
                        }
                    ]

            }

    }
}
```

Status: 201 Created

```
1
```

Figure 14. Creating a Loopback interface using POST request via Postman

In order to be able to display the result in a JSON format and post the data in the same format, the following Headers were included (Figure 15) just as in the Python script from Figure 16.



✓	Accept	application/yang-data+json
✓	Content-Type	application/yang-data+json

Figure 15. POST request headers

Figure 16 shows only a single 1 in the console's body which was the response of the request. In the right side, the Status 201 Created acknowledged that the request was successfully made. A notice was also shown on the router SSH terminal. To ensure that the interface was up and functioning, a quick check was performed immediately in the console (Figure 17).

```
Jun  6 10:40:17.459: %LINEPROTO-5-UPDOWN:
Line protocol on Interface Loopback6, changed
state to up
Jun  6 10:40:17.485: %DMI-5-CONFIG_I:  F0:
nesd:  Configured from NETCONF/RESTCONF by
admin, transaction-id 1545
R2#show ip interface brief
Intf IP-Address      OK? Method Status   Prot.
Gi1  10.1.1.254      YES NVRAM  up       up
Gi2  192.168.100.39 YES DHCP   up       up
Gi3  20.1.1.254      YES NVRAM  up       up
Gi4  unassigned      YES NVRAM  adm.down down
Lo6  6.6.6.6         YES other  up       up
```

Figure 16. The interfaces available on the router

A ping was used to test connection to a second device at the Network Layer. It did this by sending and receiving Internet Control Message Protocol (ICMP) Echo Request messages. In Windows, the ping command sent four queries by default unless it has been modified [14]. The Loopback interface's functionality was tested using this method (Figure 18).

```
R2#ping 6.6.6.6
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 6.6.6.6,
timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip
min/avg/max = 1/2/4 ms
```

Figure 17. Ping response

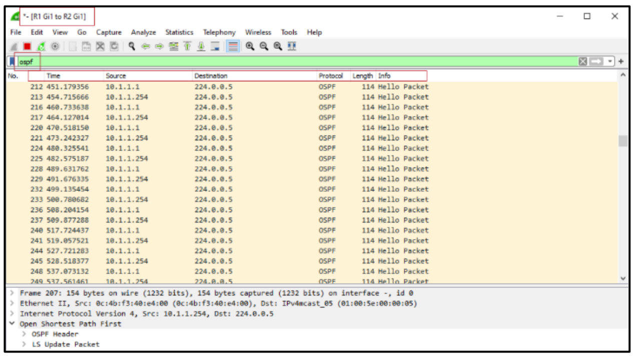
The Python script ran again, this time with changes to the response body to ensure that the DELETE method worked too. After a successful run, a message was printed in the SSH console (Figure 18) and the interface was now removed.

```
Jun  6 11:14:53.924: %DMI-5-CONFIG_I:  F0:
nesd:  Configured from NETCONF/RESTCONF by
admin, transaction-id 3366
Jun  6 11:14:53.925: %DMI-5-SYNC_NEEDED:  F0:
syncfd:  Configuration change requiring
running configuration sync detected - 'no
interface Loopback6 '. The running
configuration will be synchronized to the
NETCONF running data store.
Jun  6 11:14:55.875: %LINEPROTO-5-UPDOWN: Line
protocol on Interface Loopback6, changed state
to down
```

Figure 18. Loopback interface deletion

B. Experiments with OSPF

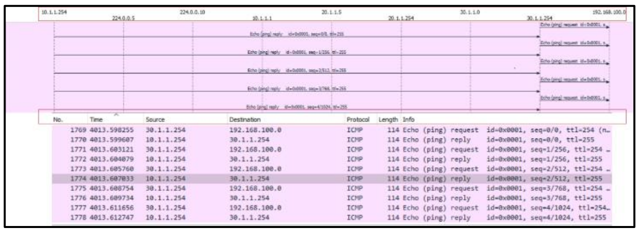
The accessible and active network interfaces on the router R2 were set through Postman or Python using the RESTCONF protocol, as previously shown. OSPF was enabled on all three routers and after the RESTCONF-manner configuration of router R2, OSPF functionality was verified. This routing protocol allowed the delivery of packets to the neighbors in order to build and to maintain adjacencies, to transmit and to receive requests, to assure reliable delivery of Link-state Advertisements (LSAs) to neighbors, and to define link-state databases. All of the LSAs that an area router sends and receives were used to create link-state databases. The Shortest Path First (SPF) technique was then used to build the shortest-path spanning tree involving the link-state database [17]. A Hello packet was a specific packet delivered by a router to create and validate network adjacency connections on a regular basis. These packets were captured in the R1-R2 network and displayed using Wireshark (Figure 19).


*- [R1 Gi1 to R2 Gi1]
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
ospf

```
No.   Time        Source       Destination   Protocol Length Info
  212 451.179356  10.1.1.1     224.0.0.5     OSPF     114 Hello Packet
  213 454.715666  10.1.1.254   224.0.0.5     OSPF     114 Hello Packet
  216 460.733638  10.1.1.1     224.0.0.5     OSPF     114 Hello Packet
  217 464.127014  10.1.1.254   224.0.0.5     OSPF     114 Hello Packet
  220 470.518150  10.1.1.1     224.0.0.5     OSPF     114 Hello Packet
  221 473.242327  10.1.1.254   224.0.0.5     OSPF     114 Hello Packet
  224 480.325541  10.1.1.1     224.0.0.5     OSPF     114 Hello Packet
  225 482.575187  10.1.1.254   224.0.0.5     OSPF     114 Hello Packet
  228 489.631762  10.1.1.1     224.0.0.5     OSPF     114 Hello Packet
  229 491.676335  10.1.1.254   224.0.0.5     OSPF     114 Hello Packet
  232 499.135454  10.1.1.1     224.0.0.5     OSPF     114 Hello Packet
  233 500.780682  10.1.1.254   224.0.0.5     OSPF     114 Hello Packet
  236 508.204154  10.1.1.1     224.0.0.5     OSPF     114 Hello Packet
  237 509.877288  10.1.1.254   224.0.0.5     OSPF     114 Hello Packet
  240 517.724437  10.1.1.1     224.0.0.5     OSPF     114 Hello Packet
  241 519.057521  10.1.1.254   224.0.0.5     OSPF     114 Hello Packet
  244 527.721283  10.1.1.1     224.0.0.5     OSPF     114 Hello Packet
  245 528.518377  10.1.1.254   224.0.0.5     OSPF     114 Hello Packet
  248 537.073132  10.1.1.1     224.0.0.5     OSPF     114 Hello Packet
  249 537.561461  10.1.1.254   224.0.0.5     OSPF     114 Hello Packet

> Frame 207: 154 bytes on wire (1232 bits), 154 bytes captured (1232 bits) on interface -, id 0
> Ethernet II, Src: 0c:4b:f3:40:e4:00 (0c:4b:f3:40:e4:00), Dst: IPv4mcast_05 (01:00:5e:00:00:05)
> Internet Protocol Version 4, Src: 10.1.1.254, Dst: 224.0.0.5
v Open Shortest Path First
  > OSPF Header
  > LS Update Packet
```

Figure 19. OSPF Hello packets exchanged between router R1 and router R2

Through OSPF, the path to the neighboring network interfaces was now known by all routers. The ping command was used to exemplify the accessibility of devices (Figure 20).



```
No.   Time         Source       Destination     Protocol Length Info
 1769 4013.598255  30.1.1.254   192.168.100.0   ICMP     114 Echo (ping) request  id=0x0001, seq=0/0, ttl=254 (n…
 1770 4013.599607  10.1.1.254   30.1.1.254      ICMP     114 Echo (ping) reply    id=0x0001, seq=0/0, ttl=255
 1771 4013.603121  30.1.1.254   192.168.100.0   ICMP     114 Echo (ping) request  id=0x0001, seq=1/256, ttl=254 …
 1772 4013.604079  10.1.1.254   30.1.1.254      ICMP     114 Echo (ping) reply    id=0x0001, seq=1/256, ttl=255
 1773 4013.605760  30.1.1.254   192.168.100.0   ICMP     114 Echo (ping) request  id=0x0001, seq=2/512, ttl=254 …
 1774 4013.607033  10.1.1.254   30.1.1.254      ICMP     114 Echo (ping) reply    id=0x0001, seq=2/512, ttl=255
 1775 4013.608754  30.1.1.254   192.168.100.0   ICMP     114 Echo (ping) request  id=0x0001, seq=3/768, ttl=254 …
 1776 4013.609734  10.1.1.254   30.1.1.254      ICMP     114 Echo (ping) reply    id=0x0001, seq=3/768, ttl=255
 1777 4013.611656  30.1.1.254   192.168.100.0   ICMP     114 Echo (ping) request  id=0x0001, seq=4/1024, ttl=254…
 1778 4013.612747  10.1.1.254   30.1.1.254      ICMP     114 Echo (ping) reply    id=0x0001, seq=4/1024, ttl=255
```

Figure 20. Ping command flow chart between router R3 and router R1

Each of the routers had a routing table available (see Figure 21-23). The router configured via RESTCONF functioned identically to the routers configured manually. The OSPF routes could also be seen in the previously mentioned figures.

```
Gateway of last resort is 10.1.1.254 to
network 0.0.0.0
S* 0.0.0.0/0 [1/0] via 10.1.1.254
   10.0.0.0/8 is variably subnetted, 2
subnets, 2 masks
C  10.1.1.0/24 is directly connected,
GigabitEthernet1
L  10.1.1.1/32 is directly connected,
GigabitEthernet1
   20.0.0.0/24 is subnetted, 1 subnets
O  20.1.1.0 [110/2] via 10.1.1.254, 00:22:20,
GigabitEthernet1
   30.0.0.0/8 is variably subnetted, 2
subnets, 2 masks
C  30.1.1.0/24 is directly connected,
GigabitEthernet2
L  30.1.1.1/32 is directly connected,
GigabitEthernet2
O  192.168.100.0/24 [110/2] via 10.1.1.254,
00:32:53, GigabitEthernet1
```

Figure 21. Routing table for router R1

```
Codes: L - local, C - connected, S - static,
R - RIP, M - mobile, B - BGP, D - EIGRP, EX -
EIGRP external, O - OSPF, IA - OSPF inter
area, N1 - OSPF NSSA external type 1, N2 -
OSPF NSSA external type 2, E1 - OSPF external
type 1, E2 - OSPF external type 2, i - IS-IS,
su - IS-IS summary, L1 - IS-IS level-1, L2 -
IS-IS level-2, ia - IS-IS inter area, * -
candidate default, U - per-user static route,
o - ODR, P - periodic downloaded static
route, H - NHRP, l - LISP, a - application
route, + - replicated route, % - next hop
override, p - overrides from PfR

Gateway of last resort is 192.168.100.1 to
network 0.0.0.0

S* 0.0.0.0/0 [1/0] via 192.168.100.1
   10.0.0.0/8 is variably subnetted, 2
subnets, 2 masks
C  10.1.1.0/24 is directly connected,
GigabitEthernet1
L  10.1.1.254/32 is directly connected,
GigabitEthernet1
   20.0.0.0/8 is variably subnetted, 2
subnets, 2 masks
C  20.1.1.0/24 is directly connected,
GigabitEthernet3
L  20.1.1.254/32 is directly connected,
GigabitEthernet3
   30.0.0.0/24 is subnetted, 1 subnets
O  30.1.1.0 [110/2] via 10.1.1.1, 00:06:03,
GigabitEthernet1
   192.168.100.0/24 is variably subnetted, 2
subnets, 2 masks
C  192.168.100.0/24 is directly connected,
GigabitEthernet2
L  192.168.100.39/32 is directly connected,
GigabitEthernet2
```

Figure 22. Routing table for router R2

```
Gateway of last resort is 20.1.1.254 to
network 0.0.0.0

S* 0.0.0.0/0 [1/0] via 20.1.1.254
   10.0.0.0/24 is subnetted, 1 subnets
O  10.1.1.0 [110/2] via 30.1.1.1, 00:08:39,
GigabitEthernet2
   20.0.0.0/8 is variably subnetted, 2
subnets, 2 masks
C  20.1.1.0/24 is directly connected,
GigabitEthernet3
L  20.1.1.5/32 is directly connected,
GigabitEthernet3
   30.0.0.0/8 is variably subnetted, 2
subnets, 2 masks
C  30.1.1.0/24 is directly connected,
GigabitEthernet2
L  30.1.1.254/32 is directly connected,
GigabitEthernet2
O  192.168.100.0/24 [110/3] via 30.1.1.1,
00:08:39, GigabitEthernet2
```

Figure 23. Routing table for router R3

VI. CONCLUSIONS AND FUTURE WORK

Being a REST-like protocol and employing the well-known HTTP methods, RESTCONF is an optimal alternative to NETCONF for automating the networks. Following the implementation steps available in the Cisco Programmability Configuration Guide, the functionality of the protocol has been verified through Command Line Interface instructions. The Postman software simplified the way requests were made by providing a Graphical User Interface. Thus, we managed to rapidly execute router configuration tasks. A first scenario was to enable RESTCONF, involving just a single Cisco CSR 1000V virtual router connected directly from GNS3 to the physical network. Once different types of requests have been double-checked and the router has been fully configured, another two Cisco CSR1000V virtual routers were added in the network topology's infrastructure. To evaluate the behavior of the RESTCONF configured router we selected the OSPF protocol to assist the discovery of all nearby interfaces. Obviously we wanted to demonstrate with these experiments that RESTCONF has an excellent potential to automate the network devices configuration in more complex infrastructures, being deployed on a larger scale.

A major outcome of this project was that we were able to get long lists of capabilities at `https://{{host}}/restconf/data/netconf-state/capabilities`, that can be split in individual items and reused later for automatic configuration.

ACKNOWLEDGMENT

A preliminary work was presented at Student Symposium on Electronics and Telecommunications SSET 2021 - Cluj-Napoca 28 May 2021. An expanded version was presented by V.A. Monoranu as B.Sc. thesis in Telecommunications Technologies and Systems in 16 July 2021.

REFERENCES

[1] S.M. Albercht, "RESTCONF Tutorial - Everything you need to know about RESTCONF in 2020", Ultraconfig.com 2020, [Online], Available: https://ultraconfig.com.au/blog/restconf-tutorial-everything-you-need-to-know-about-restconf-in-2020/
[2] "RESTCONF protocol", IETF, 2017, Available: https://tools.ietf.org/html/rfc8040.
[3] "Network automation", VMware, 2021, [Online], Available: https://www.vmware.com/topics/glossary/content/network-automation
[4] "What Is Network Automation?", Cisco, 2021, [Online] https://www.cisco.com/c/en/us/solutions/automation/network-automation.html
[5] S. Sinche et al., "A Survey of IoT Management Protocols and Frameworks," in IEEE Communications Surveys & Tutorials, vol. 22, no. 2, pp. 1168-1190, Secondquarter 2020, doi: 10.1109/COMST.2019.2943087.
[6] A. G. Prieto, A. Leung and K. Rockwell, "Automating the testing of RESTCONF agents," 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), 2015, pp. 984-989, doi: 10.1109/INM.2015.7140422.
[7] A. Ferrari et al., "A Two-Layer Network Solution for Reliable and Efficient Host-to-Host Transfer of Big Data," 2018 20th International Conference on Transparent Optical Networks (ICTON), 2018, pp. 1-4, doi: 10.1109/ICTON.2018.8473597.
[8] O. Sanghak, E. Kim, J. (Paul) Jeong, H. Ko, and K. Hyoungshick. 2017. A flexible architecture for orchestrating network security functions to support high-level security policies. In Proceedings of the 11th International Conference on Ubiquitous Information Management and Communication (IMCOM '17). Association for Computing Machinery, New York, NY, USA, Article 44, 1–5. DOI:https://doi.org/10.1145/3022227.3022270
[9] A. Mayoral et al., "First demonstration of YANG push notifications in Open Terminals," 2020 International Conference on Optical Network Design and Modeling (ONDM), 2020, pp. 1-3, doi: 10.23919/ONDM48393.2020.9133037.
[10] R. Vilalta et al., "Control and Management of a Connected Car Using SDN/NFV, Fog Computing and YANG data models," 2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft), 2018, pp. 378-383, doi: 10.1109/NETSOFT.2018.8460131.
[11] M. Granderath and J. Schönwälder, "A Resource Efficient Implementation of the RESTCONF Protocol for OpenWrt Systems," NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium, 2020, pp. 1-6, doi: 10.1109/NOMS47738.2020.9110458.
[12] "Programmability Configuration Guide, Cisco IOS XE Fuji 16.9.x," Cisco, First Published: 2018-07-18, pp. 120-130
[13] "What is Postman and how to use postman to test api?", Testrigtechnologies 2020, [Online], Available: https://www.testrigtechnologies.com/what-is-postman-and-how-to-use-postman-to-test-api/
[14] J. Edwards, "IT Basics: The Ping Utility Explained ",whatsupgold.com 2020, [Online], Available: https://www.whatsupgold.com/blog/it-basics-the-ping-utility-explained
[15] V.A. Monoranu. V. Dobrota., "Device Level Programmability Using RESTCONF", Student Symposium on Electronics and Telecommunications SSET 2021, Cluj-Napoca 28 May 2021.
[16] V.A. Monoranu, "Device Level Programmability Using RESTCONF", B.Sc. Thesis, Technical University of Cluj-Napoca, 16 July 2021.
[17] "Packet types for OSPF ", IBM, 2021, [Online], Available: https://www.ibm.com/docs/en/i/7.1?topic=concepts-packet-types-ospf
[18] "Cisco Cloud Services Router 1000v Data Sheet", Cisco, 2018, [Online], Available: https://www.cisco.com/c/en/us/products/collateral/routers/cloud-services-router-1000v-series/data_sheet-c78-733443.html
[19] B. Claise, J. Clarke, J. Lindblad, "Network Programmability with YANG", Addison-Wesley Professional, 2019
[20] "GNS3 Windows Install", GNS3, 2021, [Online], Available:]https://docs.gns3.com/docs/getting-started/installation/windows/
[21] "PyCharm", Wikipedia, 2021 [Online], Available: https://en.wikipedia.org/wiki/PyCharm.