# AN SDN APPROACH TO MITIGATING NETWORK MANAGEMENT CHALLENGES IN TRADITIONAL NETWORKS

*Blessing Bwalya*, Aaron Zimba*

Computer Science and Technology Dept., Mulungushi University, Kabwe Zambia

* Corresponding Author, e-mail: blessingbwalya.bb@gmail.com

**Abstract:** Network management consists of monitoring and controlling activities that ensure the network devices are delivering value. In traditional networks, network management is primarily performed manually via the command-line interface using SSH. This approach has several challenges, configuring large networks via the CLI is inefficient, it leads to configuration drift, and it's error-prone. This paper proposes an SDN approach that implements Ansible and Git in centralized network architecture to address the aforementioned problem. The paper shows how network management using an Ansible playbook that dynamically configures multiple devices and push's backup files to Git for version control can mitigate the highlighted challenges.

**Key words:** Ansible, Git, Continuous Deployment, SDN, Network Management.

## 1. INTRODUCTION

Network management aims to offer a service to end-users that is desirable within the confines of the network. The architecture for network management consists of a centralized controller and a series of end devices called managed devices that consist of computer systems and other network devices. From the central network management station, an engineer can survey end devices to verify the values of variables relevant to each management feature [1]. A survey conducted by [2], shows that the Command Line Interface (CLI) on individual devices is the most common primary method for making changes to network devices. According to Cisco [3] 95% of network changes are still being done manually, which means that operational expenses are 2 to 3 times greater than network costs.

Traditional networks have limitations that bring about challenges as the number of network devices increases. They are often made up of managed switches, routers and other components that are statically arranged. Such a design is very inflexible and inadequate to meet the demands of today's applications for complex traffic and programmable bandwidth [4]. The paper addresses network management challenges relating to network configurations via CLI.

Network management using the per device manual CLI model works well for small networks. Another common approach is copying pasting configurations from one device to another. Some devices have Graphical User Interface (GUI's) which can be used for configuration, however, these tools still configure one device at a time and are prone to being slow and efficient. This approach doesn't scale well in a large network and causes problems, its time consuming configuring one device at a time making the whole process inefficient [5].

Configuration drift occurs when changes are made at ad hoc and not communicated or recorded in effect causing the configuration to drift from the intended configuration over time. Traditional network management is prone to configuration drift because the system does not monitor the history of the changes. It's so easy for one engineer to alter the configuration of a device, thereafter another engineer will alter the configuration on the same device without leaving any documentation [6].

The rigid design of traditional networks adds a layer of complexity when it comes to agility and scalability. Network admin's will use Secure Socket Shell (SSH) to configure devices remotely, however, this approach has a lot of overhead and there are situations where configurations are not input correctly making the process tedious [7].

Engineers will typically prefer the CLI and will access it via SSH. Engineers typically copy and paste configurations and switch from one device to another. However this approach is prone to error as it increases the likelihood of making mistakes or typos [5].

The findings in this paper aims to contribute to network management techniques and the results will benefit computer network professionals in the industry. The paper aims to improve network management in traditional networks by proposing the use of Ansible with Git.

## 2. RELATED LITERATURE AND CONCEPTS

### 2.1. SDN Architecture

The emergence of Software Defined Network (SDN) as a new networking paradigm has been looked at as a new way to implement networks that provide centralized control. SDN is a networking paradigm that decouples the data and control functions from the networking device through the use of Application Programming Interface (API's) and moving them to a logically centralized controller. The network devices, therefore, function as general-purpose forwarding devices [18]. The SDN architecture consists of three layers, the application layer, control layer, and infrastructure or data layer. This architecture is designed to centralize the control, intelligence and management of the network. This has led to improvements in network programmability and the development of new features and services. With a centralized design, SDN makes provisioning easier while improving policy management efficiency and granularity [19].

### 2.2. Network Management using Configuration Management Tools

Configuration management is a means of dealing with changes in a system in a systematic fashion so that the system's integrity is maintained throughout time. Every change to a system is recorded in a log, together with information about who made the change, when it was done, and why it was made. This enables us to determine the precise status of a system at any given time [8].

Configuration management is a method of keeping computers, servers, and software in a consistent, desired condition. It's a method of ensuring that a system continues to work as intended while modifications are introduced over time. Administering IT system configurations entails determining the desired state of a system, such as server configuration, and then constructing and maintaining those systems [9]. RedHat developed Ansible, which is a provisioning tool, configuration management, and application deployment tool.

### 2.3. Ansible

RedHat created Ansible, which is a provisioning tool, configuration management, and application deployment tool. Ansible does not require any additional software to be installed on the target machines because it is agentless. Ansible is developed in Python and uses YAML-based Playbooks. It also relies on SSH to connect to the managed servers, as well as a host configuration file in which the managed hosts are stored [10].

Ansible uses an agentless architecture to manage network devices. A network device that does not require any programming is referred to as "agentless". Ansible uses a push model and requires SSH or NETCONF to connect to devices in order to make changes and extract data. Ansible uses a series of text files called Playbooks, which contain instructions and logic for what Ansible should do. The inventory stores device hostnames and information about each device, such as device roles. Ansible uses this information to run operations on subsets of the inventory [11]. Template files consist of device configurations with variables. Variables are a list of YAML variables that Ansible will use to replace placeholders in templates [12].

### 2.4. Git

Git is a distributed revision control system that is accessible as free software on all major development platforms. Software revision is a problem for developers, and git helps by providing each user with a complete private copy of the software repository and a variety of techniques to handle changes inside it. The ability to link a local repository to several remote ones enables developers and their managers to create a wide range of distributed workflows, the majority of which would be difficult to execute on a typical centralized version control system. The local repository also makes Git more responsive, simple to set up, and independent of the Internet [13].

### 2.5. Continuous Deployment

Continuous Deployment (CD) delivers the program to production or client environments automatically and constantly. CD ensures that a developer's changes are being automatically released from the repository to production, where they may be used by consumers [14].

### 2.6. Related Works

Apostolidis, 2015 [21] presents the use of Open Flow protocol in software defined networks. The open flow protocol is the primary protocol used in the SDN architecture to decouple the data plane from the control plane. An OF network consists of one or more OF Switches that communicate with SDN controller. The OF messages specify the behavior of the switch which responses to the commands sent by the controller. The controller maintains the desired flow table entries in the switch.

Asadollahi, et al. [20] presents the use of the Open Daylight (ODL) SDN controller (SDNC) used in OF networks. ODL is an open-source Java-based SDN controller supported by the Linux Foundation and sponsored by several organizations including high profile technology companies like IBM, Cisco, Juniper, VMWare and several other major networking vendors.

Luchian, et al. [15] present the use of Puppet as a configuration and provisioning tool Pupppet is an open source client-server configuration management tool, in which clients poll the server for desired states on a regular basis and report back to the server with the results (master) [15]. Puppet is model driven and uses its own declarative language for describing system settings. For network device support, Puppet generally employs an agent-based architecture. An on-device agent on some network devices enables Puppet to interface with the device [5].

Luchian, et al. [15] also presents Chef as an alternative to Puppet. Chef is a tool for automating the installation of programs and software on bare metal, virtual machine, and container-based clouds. Organizations, environments, cookbooks, recipes, and resources are all used in the configuration, which is defined in Ruby DSL and driven by given or derived characteristics [16]. Chef has a similar architecture to Puppet, each managed device (also known as a Chef node) runs an agent for network devices. The agent monitors configuration by pulling recipes and resources from the Chef server and then adjusting its configuration to keep up with the specifics in those recipes and runlists [5].

Masek, et al. [17] presents the use of Configuration Management tools for automating deployments in DevOps (Development, Operations) environments. DevOps strives to enhance communication, cooperation, and integration between software developers (Dev) and IT operations experts (Ops). DevOps adopts the Continuous Integration/Continuous Delivery or Continuous Deployment model that employees the use of software, automation, agile principles to reduce the time and efforts between the software development and operations [17]. The table below compares the effectiveness of proposed approach to network management using SDN with OF, and network management with Python and REST API's.

*Table 2- 2: Comparison of network management approaches*

| | Adaptation to network requirements | Not susceptible to configuration drift | Less prone to Syntax errors | Efficient configuration deployment | Scalabil ity |
|---|---|---|---|---|---|
| CLI | × | × | × | × | × |
| SDN (OF) | ✓ | × | ✓ | ✓ | ✓ |
| Python and REST API's | × | × | × | ✓ | × |
| Chef | ✓ | × | ✓ | ✓ | × |
| Puppet | ✓ | × | ✓ | ✓ | × |
| Chef and Git | ✓ | ✓ | ✓ | ✓ | × |
| Puppet and Git | ✓ | ✓ | ✓ | ✓ | × |
| Proposed Solution (Ansible and Git) | ✓ | ✓ | ✓ | ✓ | ✓ |

## 3. METHODOLOGY

Two topologies were created in GNS3. For the first topology (Fig. 1) the network automation container and two CiscoIOSVL2 switches were used to limit the resource consumption of GNS3. The second topology (Fig. 2) utilized Ansible installation on the Ubuntu virtual machine (VM) and Git was also installed on this instance. The second topology focused on implementing the pipe line with the use of two routers.

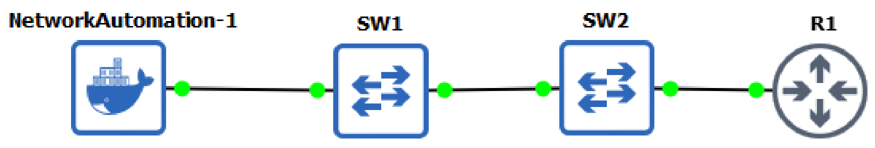


*Figure 1. First Topology*

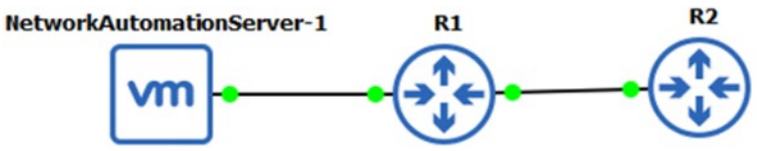


*Figure 2. Second Topology*

Ansible is installed on the network management server installed in VMWare Work Station. The network automation server is running Ubuntu 18.06 LTS, with Python 3 installed. The full list of requirements for the controller are listed below: Python 2.6 or higher, paramiko, PyYAML Jinja2, httplib2, Unix-based OS. The work automation container in GNS3 was used to reduce the resource consumption of running VMWare and GNS3 at the same time. Git is installed in Ubuntu VMWare Work Station, and it's connected to GNS3. Ansible supports various connects from major networking vendors. For the paper Cisco swiches and routers were used. The switches were running the Cisco IOSvL2 switch image and the routers were running the Cisco 3600 router image. SSH has to be configured and running on the network devices for Ansible to establish a connection. The crypoto key rsa command was used to create an SSH key on all the Cisco devices.

## 3.1. Logical Architectural Design

From a logical view the solutions uses a centralized network architecture as shown below (Fig. 3). The network automation server which consists of Ansible and Git seats a centre of the network, all network configurations are performed from a central location, and the server is able to push configurations to all managed network devices as well as pull system state information for backups.

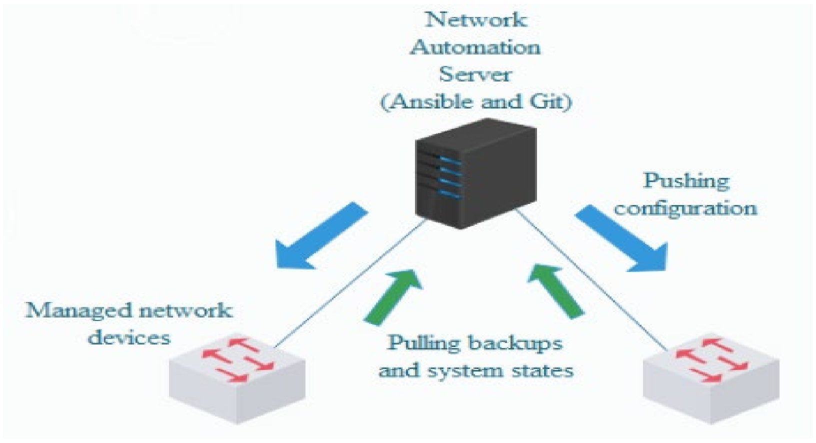


*Figure 3. Logical architecture design*

### 3.2. Framework Design

The framework proposed illustrates how configurations will flow from the network automation server to the devices. The framework diagram (Fig. 4) illustrates how configurations will flow from the network automation server to devices.

1. The engineer builds the playbook file.
2. The engineer tests the playbook on the network by running the playbook with the Ansible check condition. The playbook runs on the network without making any changes.
3. The engineer builds the playbook file.
4. The engineer tests the playbook on the network by running the playbook with the Ansible check condition. The playbook runs on the network without making any changes.
5. If the playbook produces an error, the engineer is shown an error message, the playbook needs to be modified.
6. If the playbook does not have any issues, the engineer runs the playbook without the check condition. The playbook is then pushed to the network.
7. The last task at the end of the playbook is a show running config command which saves the output to a file which is then pushed to Git

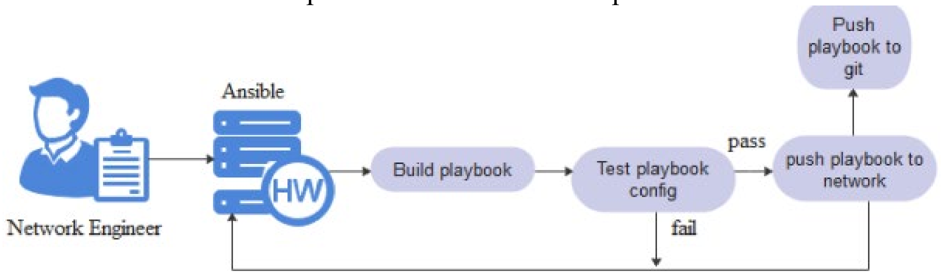


*Figure 4. Framework design*

The architecture diagram in Fig. 5 illustrates how configurations will flow from the network automation server to devices. An explanation of the various steps is highlighted below.

1. The network engineer creates the various configurations that need to be deployed in YAML files called playbooks. The playbook will consist of the devices that need to be configured, name of each task and the various tasks Ansible needs to execute and in what order.

2. The engineer runs the playbook file, Ansible uses the device information specified an access the device using the credential variables given to it.

3. Ansible runs the playbook on each device specified and in the order the engineer intended. A show running config is executed at the end of each play.

4. The output from the playbook is sent back to Ansible and displayed in the terminal. The output from the show run is saved in a file a pushed into the git repository and committed

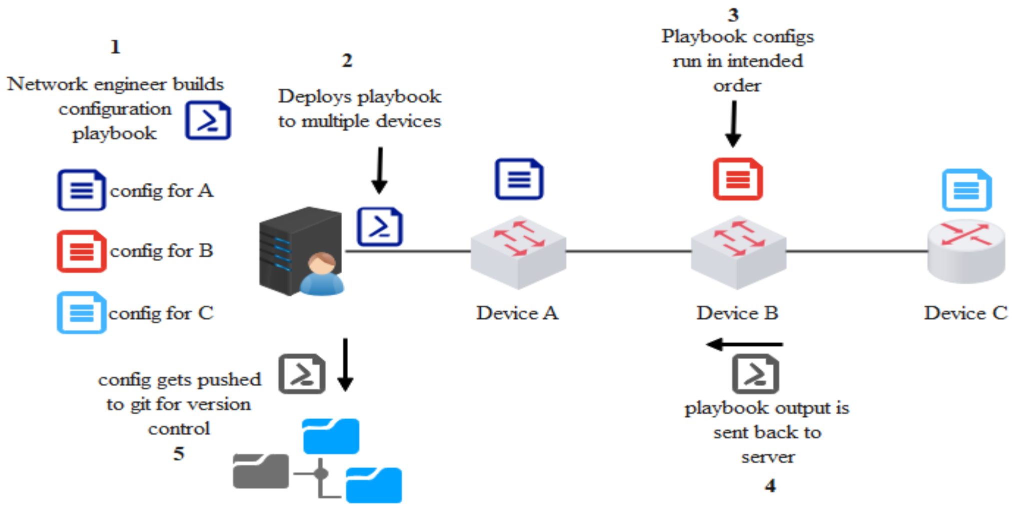

1
Network engineer builds configuration playbook
config for A
config for B
config for C
2
Deploys playbook to multiple devices
3
Playbook configs run in intended order
Device A
Device B
Device C
config gets pushed to git for version control
5
playbook output is sent back to server
4

*Figure 5. Ansible and Git architecture*

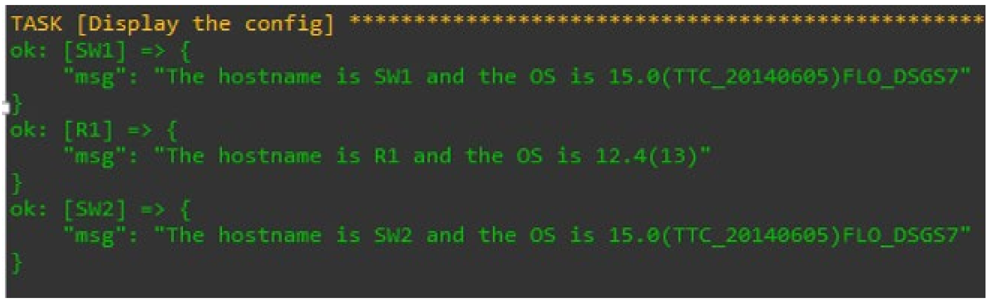
## 4. ILLUSTRATIVE RESULTS AND ANALYSIS

### 4.1. Deploying Configurations with Ansible

The first playbook created was used to get device information from all the devices on the network (Fig. 6). The code is written in YAML and is read by Ansible from top to bottom. The playbook is run by running the ansible-playbook command. The playbook logic is as follows:

- Connect to the devices defined in the all variable using the network_cli.
- Get the IOS hostname and version details from all the devices
- Display the output in the format specified.

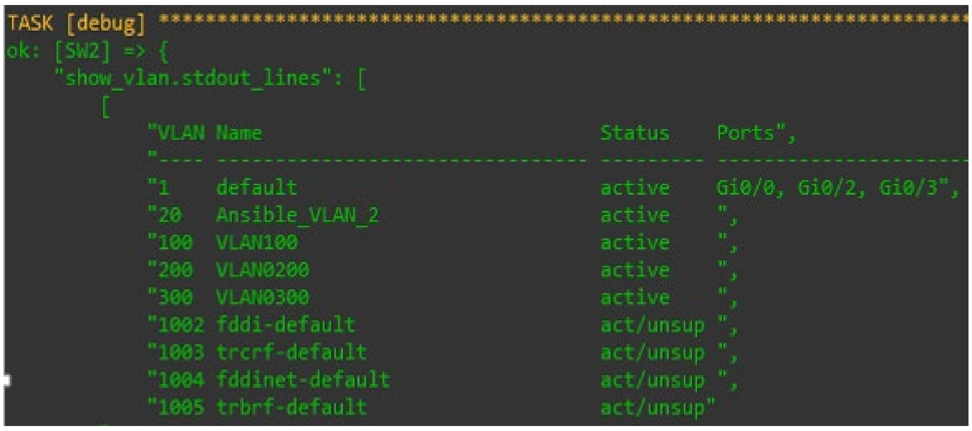
The output is displayed below showing the IOS information of SW1, SW2 and R1.



```
TASK [Display the config] ************************************************
ok: [SW1] => {
    "msg": "The hostname is SW1 and the OS is 15.0(TTC_20140605)FLO_DSGS7"
}
ok: [R1] => {
    "msg": "The hostname is R1 and the OS is 12.4(13)"
}
ok: [SW2] => {
    "msg": "The hostname is SW2 and the OS is 15.0(TTC_20140605)FLO_DSGS7"
}
```

*Figure 6. Playbook output*

The next playbook created was used to create VLANs with Ansible. The Ansible playbook will create VLANs on switches declared in the variable switches. The playbook will create VLAN 20,100,200, and 300. The playbook will name the VLAN's Ansible_ VLAN_2, VLAN100, VLAN0200 and VLAN0300 respectivily. Lastly the playbook will run the The show vlan brief command is run at the end and out as part of the Ansible output (Fig. 7).



```
TASK [debug] **********************************************************************
ok: [SW2] => {
    "show_vlan.stdout_lines": [
        [
            "VLAN Name                             Status    Ports",
            "---- -------------------------------- --------- ---------------------
            "1    default                          active    Gi0/0, Gi0/2, Gi0/3",
            "20   Ansible_VLAN_2                   active    ",
            "100  VLAN100                          active    ",
            "200  VLAN0200                         active    ",
            "300  VLAN0300                         active    ",
            "1002 fddi-default                     act/unsup ",
            "1003 trcrf-default                    act/unsup ",
            "1004 fddinet-default                  act/unsup ",
            "1005 trbrf-default                    act/unsup"
```

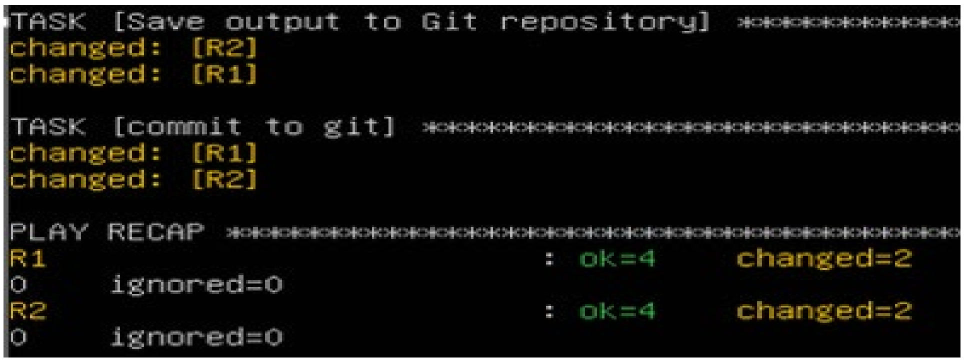
*Figure 7. Ansible vlan output*

The final playbook uses Ansible with Git for version control. This playbook uses the Ubuntu VM to run the playbook and is based on the second topology. This playbook builds of the first backup playbook and adds steps that push the backup files to Git. The logic of the playbook will get configuration information from all the hosts, the logic is as follows: the show run command is run on all the routers, the results of the command will be used for backup, the configuration backups are saved in the Git repository.

This playbook uses the Ubuntu VM to run the playbook and is based on the second topology. This playbook builds of the first backup playbook and adds steps that push the backup files to Git. The logic of the playbook will get configuration information from all the hosts, the logic is as follows

- The show run command is run on all the routers
- The configuration backups are saved in the Git repository
- Ansible does not have a built in Git add or commit module, the shell is used
- The shell changes the dir to the ansible git repository
- The Git add and commit are run from the shell.

Playbook is run on all the routers, the Git repository has highlighted a change. The output is shown in Fig. 8. The playbook was executed correctly.



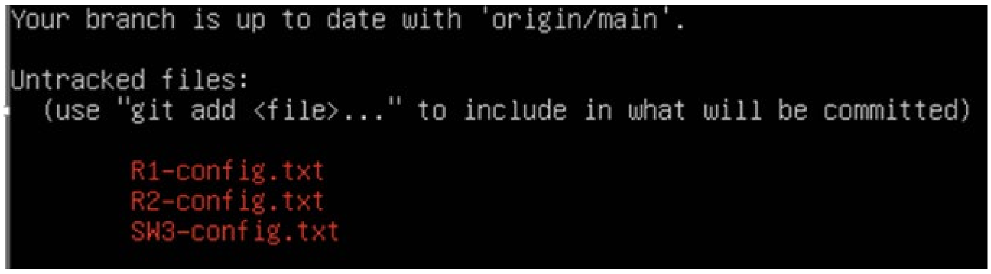
```
TASK [Save output to Git repository] *************
changed: [R2]
changed: [R1]

TASK [commit to git] *****************************
changed: [R1]
changed: [R2]

PLAY RECAP ***************************************
R1                          : ok=4    changed=2
0     ignored=0
R2                          : ok=4    changed=2
0     ignored=0
```

*Figure 8. Output from running the playbook*

The Git repository is populated with the configuration files from the devices specified in the playbook. The ability to produce to place configuration output files in a specific directory allows Ansible to be used as a quick way to collect running configurations from devices, backing up configurations and a debugging tool, this is shown in the scenario where Ansible was used to produce a text file of the running configurations of all the devices and placing them in a Git repository (Fig. 9).



```
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        R1-config.txt
        R2-config.txt
        SW3-config.txt
```

*Figure 9. Repository after playbook*

Performance analysis in table 1 shows how long it takes to run the commands using Ansible and cli. Deploying configurations using Ansible is on average much faster than using the cli. The time taken to configure the devices was recorded using an external watch. Three runs were conducted for each configuration and the average was calculated at the end. The output is recorded in the table below.

*Table 1. Performance analysis*

| | Ansible | | | | CLI | | | |
|---|---|---|---|---|---|---|---|---|
| | **1st** | **2nd** | **3rd** | **Average** | **1st** | **2nd** | **3rd** | **Average** |
| **IOS info** | 7s | 3s | 4s | 4.6 | 10s | 12s | 11s | 11 |
| **config vlan** | 5s | 3s | 4s | 4 | 36s | 35s | 33s | 34.6 |
| **backing up** | 10s | 9s | 11s | 10 | 45s | 48s | 43s | 45.3 |

## 5. CONCLUSION

Ansible for network automation, which incorporates YAML for data formats needed with automation, and Git for version control is detailed in this paper. Ansible is a Red Hat open source technology with a low entry barrier for network automation and a large user community. Git is a distributed version control system that suited to keep track of various file versions as they change. The paper utilized GNS3 to create a virtual topology using Cisco switches and routers. By using Ansible for network management as Git version control, the paper was able to implement a workflow pipeline that automates and tracks version changes of the configurations. This pipeline embraces a continuous deployment approach to network management to reduce CLI interaction, provide a means to track the configuration changes and transparency. With this approach the network management challenges of traditional networks are mitigated

## REFERENCES

[1] Rao, U. H. Challenges of Implementing Network Management Solution. *International Journal of Distributed and Parallel Systems*, Vol.2, No.5, 2011, p.67.

[2] Gartner. *Look-beyond-network-vendors-for-network-innovation0*. Retrieved 06 13, 2021, from https://www.gartner.com/en/documents/3847469/look-beyond-network-vendors-for-network-innovation0

[3] Cisco. *Intent-based-networking.html*. Retrieved 06 14, 2021, from https://www.cisco.com/c/en_uk/solutions/enterprise-networks/intent-based-networking.html

[4] Deshpande, H. A. Software Defined Networks: Challenges Opportunities and Trends. *International Journal of Science and Research (IJSR)*, 328, 2013.

[5] Odem, W. *CCNA 200-301 Official Cert Guide Volume 2*. Cisco Press, 2020.

[6] Lammle, T. *CCNA Certification Study Guide Exam 200-301*. John Wiley and Sons, 2020.

[7] Tank, G., Dixit, A., Vellanki, A., & D, D. A. Software Defined Networks: The New Norm for. *International Journal of Science and Research (IJSR)*, 33, 2014.

[8] Heap, M. *Ansible From Beginner to Pro*. Reading, Berkshire: Apress, 2016.

[9] Redhat. *What-is-configuration-management*. Retrieved 06 08, 2021, from https://www.redhat.com/en/topics/automation/what-is-configuration-managemen

[10] Islami, M. F., Musa, P., & Lamsani, M. *Jurnal Ilmiah KOMPUTASI*, Vol. 19, No.2, 2020, p. 129.

[11] Shah, J., Dubaria, D., & Widhalm, P. J. *A Survey of DevOps tools for Networking*, 2018.

[12] John, A. *31 Days Before Your CCNA Exam 200-301Certification Exam.* Hoboken: Ciscopress, 2020.

[13] Spinellis, D. Git. *IEEE Software*, Vol. 29, No. 3, 2021.

[14] Weber, I., Nepal, S. & L. Zhu. Developing Dependable and Secure Cloud Applications,. *IEEE Internet Computing*, Vol. 20, No. 3, 2016, pp. 74-79.

[15] Luchian, E., Filip, C., Rus, A. B., Ivanciu, I.-A., & Dobrota, V. *Automation of the Infrastructure and Services for an OpenStack Deployment Using Chef Tool*. Technical University of Cluj-Napoca., 2016.

[16] Chef. *Docs.chef.io*. Retrieved July 22, 2016, from https://docs.chef.io/

[17] Masek, P. et al. Unleashing Full Potential of Ansible Framework: University Labs Administration. s.l., *The 22nd Conference of Fruct Association*, 2018.

[18] Pradeep Kumar Sharma, S. S. T. Improving Security through Software Defined Networking (SDN): AN SDN based Model. *International Journal of Recent Technology and Engineering (IJRTE)*, Vol. 8, No. 4, 2019.

[19] Taha, A. *Software-Defined Networking and its Security*. Aalto university school of electrical engineering, 11, 2014.

[20] Asadollahi, S., Goswami, D. B. & Gonsai, D. A. M. Implementation of SDN using OpenDayLight Controller. *International Journal of Innovative Research in Computer and Communication Engineering,* Vol. 5, No. 2, 2017, p. 220.

[21] Apostolidis, Panagiotis. *NetworkMA, Network management aspects in SDN*, 2016

## *Information about the authors:*

**Blessing Bwalya** – is a lecturer at Northrise University and currently holds a Bachelor of Science in Computer Networking from Northrise University and currently pursuing his Masters of Science in Information Technology from Mulungushi University. He is also a member of the ICT Association of Zambia.

**Aaron Zimba** – is a lecturer at Mulungushi University and obtained his PhD in Network and Information Security at the University of Science and Technology Beijing in the Department of Computer Science and Technology. He received his Master and Bachelor of Science degrees from the St. Petersburg Electrotechnical University in St. Petersburg in 2009 and 2007 respectively. He is also a member of the IEEE.

**Manuscript received on 29 September 2021**