# Automated Network Programmability Using OpenConfig YANG Models and NETCONF Protocol

Abderrahim Amlou*, Amar Abane*, Mheni Merzouki*,
Lydia Ait Oucheggou*, Zineb Maasaoui*, and Abdella Battou*
*National Institute of Standards and Technology, 100 Bureau Dr, Gaithersburg, MD 20899, USA

***Abstract*—This paper introduces a microservice-based architecture designed to enable automation of network programmability and management. Amid the complexity of today's networks and the diversity of equipment, achieving efficient and reliable network programmability poses a significant challenge. Our architecture leverages the OpenConfig YANG models and the NETCONF protocol to simplify network configuration, automate tasks, and avoid errors. We demonstrate how our solution streamlines the collection and configuration workflows, enabling network operators to efficiently manage complex networks. The paper further presents a performance evaluation of the proposed design, which confirms its efficacy in handling complex configurations and its fault tolerance capabilities.**

***Index Terms*—Network management; Network automation; Network configuration; OpenConfig; YANG model; NETCONF**

## I. Introduction

Network programmability is essential to provide operators with flexibility in configuring and managing the network. However, the complexity of modern networks and diversity of equipment makes network management challenging, requiring standardized models and protocols to achieve efficient and reliable network programmability. The use of standardized models and protocols such as the OpenConfig YANG (Yet Another Next Generation) models [1] and the NETCONF protocol [2] simplifies network configuration and enables operators to automate tasks while reducing errors.

OpenConfig YANG models provide a standardized, vendor-neutral, and understandable data descriptions for device configuration, while NETCONF protocol carries YANG data in efficient and secure manner between the devices and the management system [2].

In this paper, we present a microservice-based architecture that efficiently automates programmability of configuration-based networks for management purposes. The architecture provides network state collection, device configuration and real-time notifications using NETCONF protocol and the OpenConfig YANG models. Our approach facilitates the development of automated network management functions by simplifying the collection and configuration workflows. It also allows network operators to achieve efficient and reliable network programmability in complex networks.

The proposed design is evaluated on a testbed network to assess its efficiency in handling complex configurations and providing fault tolerance.

This paper is organized as follows. Section II describes network programmability concepts and appraoches, and reports on recent related work. In Section III, we introduce the design and implementation of the microservice architecture for efficient network programmability. In Section IV-B, a performance evaluation of the architecture is presented. Section V concludes with a discussion on implications of the proposed architecture and outlines research perspectives.

## II. Related Work

OpenConfig YANG models [1] and NETCONF protocol [2] are two key standards that have emerged in recent years to address network programmability challenges.

NETCONF protocol is an industry-standard network management protocol widely supported by equipment. It provides a programmatic interface for configuring and managing network devices. The protocol allows network administrators to automate network management tasks such as device discovery, configuration, and monitoring, using standardized data models. The protocol uses the YANG data modeling language to describe the data exchanged between network devices and the management system, allowing for a standard representation of network configuration, state, and notification [3].

OpenConfig YANG models are a set of vendor-neutral data models developed by network operators. These models provide a standardized representation of network devices and services, enabling uniform management access across different vendors. OpenConfig YANG models are designed to be modular, extensible, and easily understandable.

Leveraging OpenConfig YANG models and the NETCONF protocol for automated network programmability offers notable advantages. These standards streamline network management tasks by offering a uniform framework for network configuration, minimizing complexity and enhancing automation. The risk of errors is reduced due to standardized data models, which promote consistency and accuracy. The NETCONF protocol's transactional operations, validation, and rollback features ensure uniform device updates. Furthermore, the combination of OpenConfig YANG models and NETCONF improves network reliability and scalability by reducing human errors and facilitating remote management for large-scale networks.

Network programmability provided by YANG models and NETCONF can be automated in different ways. For example, Ansible [6] can be used in conjunction with NETCONF for automated network management. Ansible is a popular open-source automation tool for IT infrastructure management. It allows users to automate the deployment, configuration, and management of applications and systems, as well as orchestrate complex workflows. The tool is widely used in DevOps and IT operations to automate repetitive tasks and streamline operations. Ansible, with its simple and robust framework, is a powerful tool for network configuration and management. It allows IT administrators to automate routine tasks, promote efficiency, and reduce human error. While Ansible does offer some level of verification via its idempotent nature and check mode, these features may not provide a comprehensive solution for complex network management tasks. However, for extensive validation, performance optimization, or policy compliance checks, network engineers need to employ additional tools or manual checks. This could add to the complexity and time-consuming nature of network management.

Automated network programmability has been considered mainly in Software Defined Networking (SDN), especially next-generation SDN controllers which natively support device configuration in addition to flow control (e.g., OpenFlow). For example, ONOS (Open Network Operating System) has been updated to a modern cloud-native ecosystem called micro-ONOS, also referred to as $\mu$ONOS [4]. $\mu$ONOS aims to provide an industrial-grade platform that can drive the transformation of network infrastructures by meeting the key requirements of modern networked systems, including network control, configuration, packet-level network telemetry, runtime verification, diagnostics, and support for 5G functions at the network edge. Teraflow [5] is another cloud-native SDN controller for flow control and integration with optical and microwave network equipment. TeraFlow also incorporates security using Machine Learning (ML) and distributed ledgers. These projects demonstrate the growing interest and importance of network programmability and automation in modern network management. TeraFlow's focus on performance testing and measurement highlights the need for automated and scalable solutions in network management. The micro-ONOS/ONOS project emphasizes the importance of an open-source platform for network programmability and automation, providing a flexible and customizable solution for network operators.

In the realm of optical networks, an insightful study titled "Software-Defined Networks for Optical Networks using Flexible Orchestration" [7] addresses the challenges brought about by the complexities of modern optical networks. The paper reviews existing solutions that leverage software-defined networks (SDNs) and network service orchestrators to tackle issues related to network flexibility, reliability, and quality of service. Moreover, it identifies gaps within these solutions and delves into the improvements introduced by custom-tailored SDN implementations. The proposed orchestration architecture, which incorporates the ONOS SDN controller, dockerisation, and Kubernetes clusterisation, demonstrates enhanced flexibility, reliability, customizability, and quality of service compared to existing approaches. This contribution serves as an advancement in the ongoing efforts to optimize optical network performance and functionality.

## III. Proposed Approach

### A. Overview

The proposed approach is based on a microservice architecture to enable automated network management and configuration verification while maximizing software agility and scalability.

The microservice approach provides isolation and decoupling of management functions during design, development, testing, and deployment stages. This improves scalability as containerized microservices can independently and automatically scale up and down as the network size and the management loads change. It also procudes more resilient systems; a failure in one microservice doesn't necessarily impact the whole system. Moreover, since microservices are independent entities, different development teams can work on different services at the same time, potentially reducing time to market.

The decoupled microservice architecture is further improved by adopting an event-driven interaction among microservices. This allows the architecture to react instantly to events, facilitating real-time responses. Improved performance is achieved through asynchronous event processing, enabling the system to handle high loads without hindering user interactions. The architecture also promotes real-time data synchronization across services, minimizing data inconsistency, and parallel development by different teams becomes feasible due to the independence of microservices.

However, this architecture also presents challenges, such as managing data consistency and complex inter-service communication, necessitating careful consideration before implementation.

Instead of the path-value representation derived from YANG models, our design utilizes an internal and intuitive model of the network that reflects multiple levels of network abstractions, including network elements, physical and logical topology, configurations, and data plane information. This allows for easy implementation of configuration verification and network state analysis without the necessity to translate path-value information into a more expressive model. Hence, configurations can be verified before applying then to the devices, thereby avoiding potential errors and reducing the risk of network downtime.

When a network administrator issues a command using the NETCONF protocol, the command is encapsulated in an XML message and sent to the network device. The device then processes the command and sends a response message back to the controller, also in XML format. The controller can then process the response message and perform further operations if needed. The OpenConfig YANG models are used to define the data models that are used in our solution. These models define the structure and semantics of the data that

is exchanged between the controller and network devices. By using standardized data models, our solution simplifies network management and reduces the risk of errors.

### *B. Microservice Design*

The proposed design is composed of several microservices (see Figure 1) responsible for different tasks and all interacting with the inventory microservice.

The inventory microservice provides access to the network data repository via an encapsulation of the network model. The network data is an SQL database that persists the network state and pending configurations. The inventory microservice exposes a RESTful API to external clients such as the Graphical User Interface (GUI), and streams events to the other microservices when changes are made to the network repository.

The collection microservice gather status and configuration data from network devices, and pushes it into the inventory service. This data includes device status, interface configurations, routing tables, etc.

The notification microservice receives real-time notifications from network devices via NETCONF for port/link status and alarms. The service then updates the network state accordingly via the inventory service and may notify operations via the GUI.

The configuration microservice receives pending configuration changes inserted via the inventory service and tries to apply them to the appropriate device. It can rely on specialized YANG-adapter microservices to translate the internal events into the appropriate XML format according to the supported YANG model, before applying the configuration to the device's NETCONF server.

Intelligent services such as VLAN and routing configuration can be deployed to automatically configure the network by inserting desired changes via the inventory service. Similarly, analysis services such as configuration verification and fault management may access network state via the inventory service and provide feedback on the network behaviour and troubleshooting.

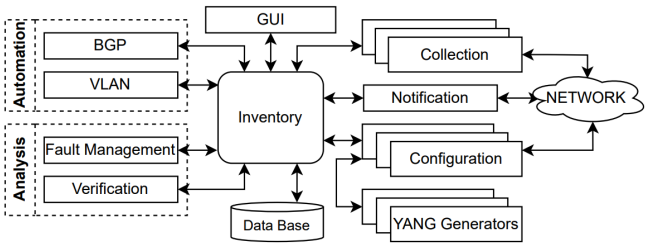


Fig. 1: Microservice architecture design

Figure 2 shows the communication between the proposed architecture and the network devices.

Figure 3 summarizes the network concepts currently supported by the system.

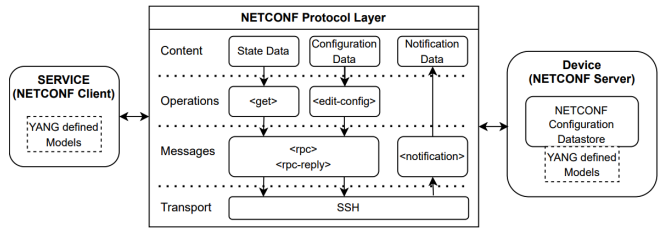


Fig. 2: The NETCONF protocol layer

| Service design | | |
|---|---|---|
| **Content** | Microservice | SUPPORTED |
| **State Data** | Collection | • **Physical interfaces, Ip interfaces, VLAN, Routing, LLDP** |
| **Notification Data** | Notification | • **PORT STATUS, LINK STSTUS, ALARM (FREQ CHANGED VALUES)** |
| **Configuration data** | Interfaces | • **Physical interface configuration : (speed, MTU, adminStatus, description)**<br>• **IP interface parameters : ( IP address, subnet mask)** |
| | Static Routes | • **Routing configuration: (Static routes)** |
| | Routing Protocols | • **Routing tables configuration: (Routing tables, BGP)** |
| | Switching | • **VLAN configuration: (VLAN, VlAN IDs, names)**<br>• **VLAN Membership configuration**<br>• **SVI configuration**<br>• **Bridge Group configuration: (Bridge Group ID, Bridge Group name)** |
| | Security | • **Access Control Lists (ACLs)** |
| | Tunnel | • **Virtual Private Network (VPN) ( L2, L3)** |
| | OPTICAL | • **Cross connects**<br>• **Optical path** |

Fig. 3: The service design with supported operations

### *C. Implementation*

The proposed architecture has been implemented with Java Vert.x and Python for the backend services, and Vue3 for the GUI. The microservices and the GUI are deployed as Docker containers and managed via Docker Compose.

The configuration process begins with the events generated by the inventory based on the changes inserted from the GUI or from an automated configuration microservice such as BGP (Border Gateway Protocol). After receiving the event, the configuration microservice calls the corresponding YANG-adapter to translate the configuration into the appropriate XML message format. An example of such translation is shown in Figure 4. The XML command is then sent to the device to be applied.


Fig. 4: Example of the XML message created based on the event

## IV. Evaluation and Results

### A. Testbed Description

In order to examine the primary functionalities of the system, we established a testbed illustrated in Figure 5 which comprises of numerous switches and routers that are compatible with the Netconf protocol. the switches and routers are interconnected and configured to achieve complete connectivity. The target routers and switches are reachable to the different system microservices through an out-of-band management switch.

Each microservice interacts with the routers to either collect up-to-date information about the network or apply a new configuration and then update the database when needed, these updates are then reflected on the web interface which end users utilize to interact with the system.

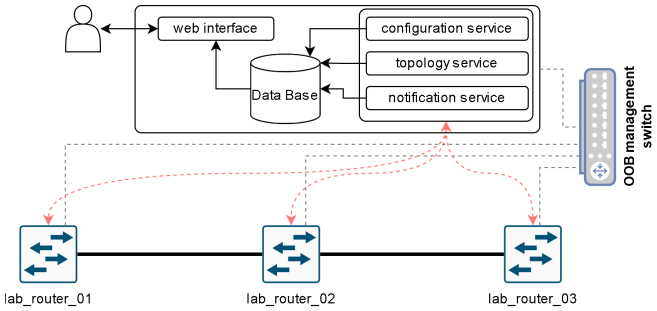


Fig. 5: The testbed design

### B. Evaluation

To assess the performance of our solution, we conducted a series of experiments on a testbed network. We focused on three key metrics: throughput in messages per second,delays of configurations per events and the size of the NETCONF messages in bytes.

To evaluate the performance of our solution, we measured the maximum throughput in messages per second for different types of configurations. Figure 6 illustrates the achieved throughput for each configuration type.This metric encapsulates the combined impact of configuration processing and NETCONF communication delays. Subsequently, we juxtaposed this throughput data against the configuration processing throughput, as showcased in Figure 7, thereby facilitating an insight into the individual effects of these components.

We employed a comparable methodology to assess latency. Figure 8 offers an overview of the total process-induced latency, encompassing all relevant factors. Conversely, Figure 9 provides a focused depiction of the latency attributed specifically to configuration processing. These visualizations also clearly illuminate the distinct impact of each element.

To gain insights into the size of NETCONF messages generated by our solution, we measured the size of the messages in bytes. Figure 10 provides a visualization of the message size distribution. The graph shows that the majority of NETCONF messages are within a certain size range, indicating the efficiency of our solution in generating compact and concise messages.

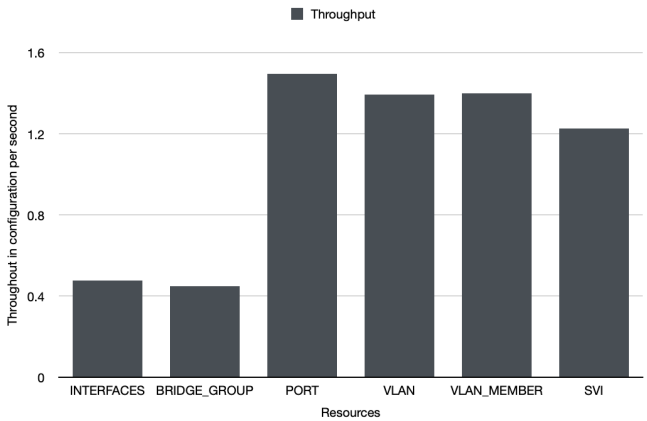


Fig. 6: Configuration throughput in messages per second for different resources

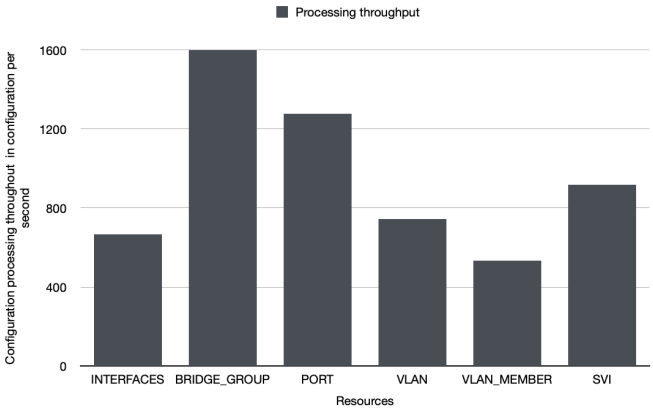


Fig. 7: Configuration processing throughput in messages per second for different resources

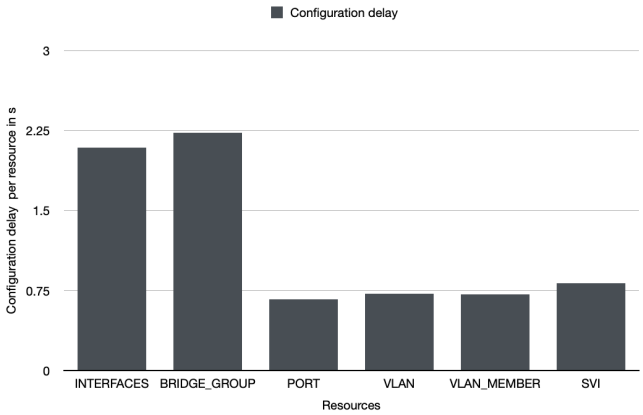


Fig. 8: Configuration delay per resource in seconds

Our comprehensive evaluation underscores the robustness and efficacy of our proposed solution in addressing the challenges of network programmability and management. Our

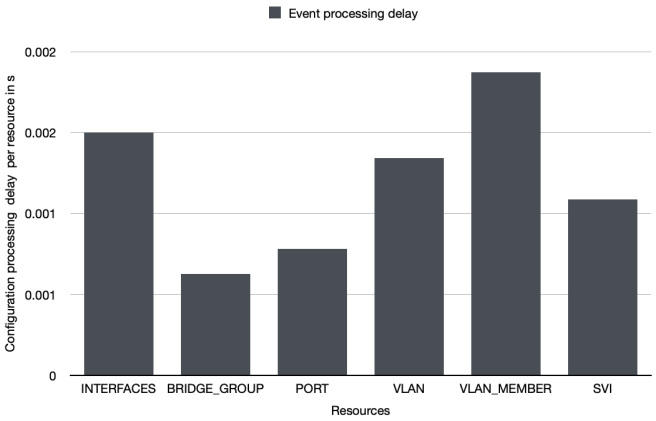


Fig. 9: Configuration processing delay per resource in seconds

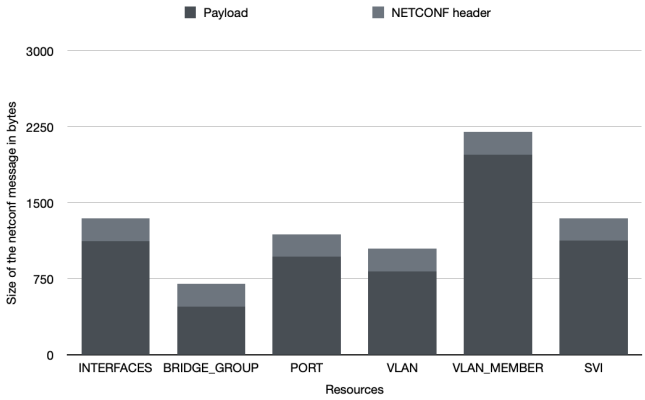


Fig. 10: Size of NETCONF messages in bytes

detailed examination of the metrics demonstrates that our solution performs well across different aspects. and also highlights the notable efficiency in generating NETCONF messages, signifying the practical viability of our approach. For instance, the achieved processing configurations throughput, averaging around 1000 messages per second, aptly meets contemporary demands. Furthermore, the compact message size ensures an economical utilization of network resources, exemplifying our solution's ability to streamline network operations and minimize resource consumption.

## V. Conclusion and Future Work

In this study, we developed and showcased a microservice-oriented architecture that effectively automates network programmability, while also enabling configuration validation and network analysis. Our design utilizes the NETCONF protocol and OpenConfig YANG models to achieve vendor-agnosticism. The efficacy and efficiency of our approach, particularly in managing heavy configuration operations and ensuring fault tolerance, are confirmed by our performance evaluation.

Despite encouraging outcomes, there is room for enhancement and further exploration. We aim to broaden our evaluation to include larger and more intricate network topologies to assess the scalability of our solution. In addition, we plan to study the incorporation of other programmability protocols, such as RESTCONF and gNMI (gRPC Network Management Interface), to offer greater adaptability and compatibility across varying network environments. We further plan to enrich our system with automation services for routing and VLAN configuration and explore the application of machine learning methods for network state analysis services.

## VI. Disclaimer

Any mention of commercial products or reference to commercial organizations is for information only; it does not imply recommendation or endorsement by NIST, nor does it imply that the products mentioned are necessarily the best available for the purpose.

## References

[1] OpenConfig Working Group. (n.d.). OpenConfig. Retrieved March 30, 2023, from https://www.openconfig.net/

[2] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, https://www.rfc-editor.org/info/rfc6241.

[3] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, https://www.rfc-editor.org/info/rfc7950.

[4] Open Networking Foundation (ONF), "Micro ONOS (uONOS)," 2020. [Online]. Available: https://opennetworking.org/news-and- events/blog/

[5] R. Vilalta et al., "TeraFlow: Secured Autonomic Traffic Management for a Tera of SDN flows," 2021 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit), Porto, Portugal, 2021, pp. 377-382, doi: 10.1109/EuCNC/6GSummit51104.2021.9482469.

[6] Ansible. (n.d.). Retrieved May 11, 2023, from https://www.ansible.com/overview/how-ansible-works

[7] Nsafoa-Yeboah, K.; Tchao, E.; Yeboah-Akowuah, B.; Kommey, B.; Agbemenu, A.S.; Keelson, E.; Monirujjaman Khan, M. Software-Defined Networks for Optical Networks using Flexible Orchestration : Advances, Challenges and Opportunities. Preprints 2022, 2022070281. https://doi.org/10.20944/preprints202207.0281.v1