A Programmable Controller for Unified Management of Virtualized Network Infrastructures

Mohamad Yassin, Karine Guillouard, Meryem Ouzzif, Roland Picard, Denis Aluze

A Programmable Controller for Unified Management of Virtualized Network Infrastructures

Mohamad Yassin, Karine Guillouard, Meryem Ouzzif, Roland Picard, Denis Aluze
Orange Labs, 4 Rue du Clos Courtel, 35510 Cesson Sévigné, France
{name.surname}@orange.com

***Abstract*—We are currently witnessing an evolution towards network virtualization and Software Defined Networking (SDN) paradigm. SDN permits the decoupling of the control and data planes of traditional networks, and offers flexible and programmable networks. Besides, the need for dynamic provisioning and composition of networking services leads to the deployment of software components on virtualized network infrastructures. In this paper, we propose a unified management of networking and Information Technology resources and the exposure of management operations to the operator and to third-party service providers. For this sake, we introduce a programmable *Controller* that offers runtime reconfiguration of components. It exposes different management views according to operator or third-party applications' needs, and simplifies the automation of virtualized networking services. A networking use case is presented to evaluate the advantages of the proposed *Controller*, where our approach is compared to the control of both Linux Bonding and Click modules. Our *Controller* overcomes the limitations of virtual network functions management and provisioning approaches at the expense of a slight increase in execution time.**

***Index terms*—** Network virtualization, component management, runtime reconfiguration, resource controller, 5G.

I. Introduction

With the proliferation of networking services demands and the generalization of computing infrastructures, network functions are implemented as software components on virtualized network infrastructures. These functions include among others, packet processing, caching, load balancing, network address translation, firewalling, and security functions. Instead of relying on hardware infrastructures and proprietary appliances, Network Function Virtualization (NFV) decouples networking services processing and applications from their underlying hardware appliances [1, 2], and offers flexible provisioning of these virtualized functions on general purpose computing infrastructures.

Each networking service is decomposed into a set of Virtual Network Functions (VNFs) that may be deployed as software components on Virtual Machines (VMs) running on physical servers. Moreover, Software Defined Networking (SDN) [3–5] separates the network's control plane from the underlying data plane. It promotes network programmability and flexibility in management operations. In this context, the 5G network architecture [6] is expected to support on-demand composition of network functions and network capabilities, where VNFs are units of networking assembled according to operator and clients' needs. Particularly, 5G introduces various management and control views such as a logical and functional view, a system management view, a resources view, an infrastructure control view, and an application view. Therefore, programmability enablers or controllers are required to make use of these views. These enablers are essential to achieve the objectives of flexible 5G networking-as-a-service, which targets on-demand composition, configurability, exposure, and autonomy of network functions.

Our proposal is to unify as much as possible the management of heterogeneous resources such as networking, storage, memory, virtual machines, and software functions. In fact, this proposal relies on component-based modelling that offers a unified representation of heterogeneous resources, including service, management, and infrastructure elements. This is particularly interesting for 5G architecture management that will involve the slicing of diverse resources. In this paper, we propose a novel *Controller* that offers the means to flexibly manage different types of resources, and that exposes several management views depending on administration needs. For instance, when a client requests a networking service, its Service Level Agreement (SLA) is negotiated with the network operator. The required components, including networking and Information Technology (IT) resources, are initialized, assembled, and installed to provide the requested service. Hence, our *Controller* ensures service establishment and lifecycle management by providing the means and the mechanisms to automate component administration operations, as long as the networking service is used. The modification of components assembly should be done on-the-fly and without service disruption. Moreover, our *Controller* may be programmable to offer management views on a selected set of homogeneous or heterogeneous resources, such as the previously mentioned 5G management and control views, depending on the running service. This *Controller* exposes component management to the network operator, to third-party service providers, and to the client itself.

The remainder of this paper is structured as follows. In section II, we describe state-of-the-art approaches related to networking functions abstraction, deployment, and management. Our proposed *Controller* is explained in section III, where the specifications of its novel management functions are given. Section IV provides a networking use case that illustrates the advantages of our proposed *Controller*. A qualitative comparison with the control of the Linux Bonding module [7] and the Click module [8] is also given. After this comparison,

we evaluate the performance of the described approaches in terms of complexity and response time. Evaluation results are given in section V. Section VI concludes the paper, and summarizes our contributions.

II. Related Work and Requirements

In 5G networks [6], applications and business services of different clients and service providers will be allowed to manage and control the underlying infrastructure resources, including logical and physical networking resources. There is a need to fill the gap between Telco business needs and operational management systems via customizable orchestrators and infrastructure programmability enablers. For instance, authors of [9] address the changing requirements of users, services, and operators while taking networking and IT resources constraints into account. An embedded softwarization layer that includes programmability enablers is introduced, and it takes energy-efficiency constraints into account. In [10], authors introduce an SDN controller with service orchestration capabilities; however, the unified modeling of heterogeneous resources is not considered.

We identify the requirements of a programmable controller as follows:

- Unifying the management operations by standardizing heterogeneous resources such as networking infrastructure and IT resources. For instance, the Grid Component Model (GCM) is a hierarchical component model that represents software modules as components with a standardized description of what they need and provide [11]. Although it defines a standardized membrane with connections to the control modules, the dynamic components reassembly is tedious due to the tightness of hierarchical bindings between components. In the Scylla framework [12], different types of programmable network fabrics, such as forwarding nodes, packet processing nodes, and radio nodes, are addressed. However, the implementation of third-party network functions is not supported.
- Exposing customized controllability and programmability enablers to third-party management entities. The NFV architecture framework proposed by ETSI includes a functional block called Element Management (EM) responsible for configuration, fault management, accounting, performance measurement, and security management of a VNF [13]. These functionalities are exposed by the EM to the VNF Manager (VNFM) that performs lifecycle management of VNF instances. For example, the Juju provisioning tool provides EMs called Juju Charms [14, 15] that can only be managed by the Juju Generic VNFM. These Juju Charms do not expose controllability and programmability functions to third-party management entities and applications.
- Providing different management views over the involved resources depending on the management functions' needs. For instance, the policy manager component proposed in [16] automatically identifies and configures security functions. However, it only integrates network and security policy management into the NFV framework. Note that the 5G architecture identifies applications, management, infrastructure, and physical resources views.
- Controlling components assembly and reconfiguring components structure at runtime [17], while avoiding service disruption. Monitoring and configuration data can be modeled using a modular language, such as YANG [18], and sent over a network configuration protocol.
- Implementing the Manage Analyze Plan Execute (MAPE) loop [19, 20], that enables autonomic resource capabilities such as fault resolution, monitoring, and self-protection. For example, a multi-layer diagnosis framework is proposed in [21]. It offers a fine diagnosis granularity by constructing network and service dependency graphs. The proposed approach promotes on-the-fly root-cause analysis, but it does not provide the means for service and resource recovery management.

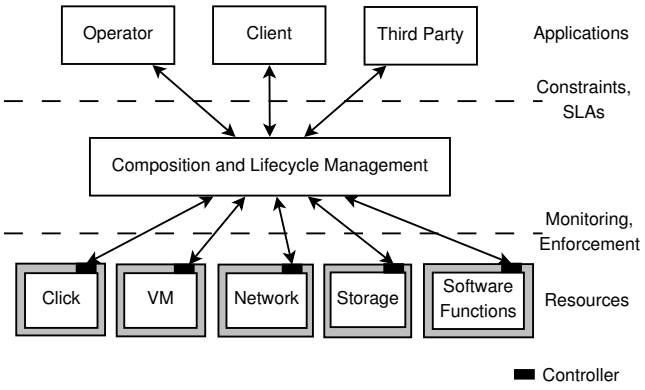


Fig. 1: Managing heterogeneous resources

Therefore, there is a need for flexible, customizable, and programmable controllers depending on the management tasks and the underlying resources. To address this need, we introduce an adaptive management approach illustrated by the programmable *Controller*. It is a software component for dynamic reconfiguration, component assembly configuration, and component management exposure. These operations are made available by the *Controller* regardless of the type of the managed resources, such as network, storage, memory, or software functions, as shown in Fig. 1. The *Controller* offers management views tailored to the administrator's needs, and different levels of granularity for control operations.

In our work, we choose the Click framework [8] to concretely illustrate our programmable *Controller*. Click is a highly programmable resource that allows the construction of flexible and configurable packet processing nodes such as routers, firewalls, Network Address Translators (NAT), etc. It is also used to virtualize broadband remote access servers [22] due to its modular capabilities. A Click router is an assembly of atomic packet processing software components, which may be dynamically modified. Each component performs basic and simple router functions such as counting, dropping, schedul-

ing, classification, and interfacing with network devices. Components are linked together using a declarative programming language called *Click* [23, 24]. Click offers component-specific and generic control functions called *handlers*, that are `C++` functions performing read or write operations. However, it is not possible to have different management views, such as manipulating a selected set of components, or to expose unified management functions to third-party service providers. Our proposed *Controller* is described in the following section.

III. Programmable Controller

A. Description

We are mainly focusing on three objectives. Firstly, the unification of management operations for different types of resources, such as Click, VMs, network, storage, and software functions. Secondly, the exposure of these operations to the network operator, to the client, and to third-party applications. Thirdly, the provisioning of different views over a set of components depending on management needs. Further objectives include runtime reconfiguration and automation of control operations.

To achieve these goals, an enhanced version of the EM functional block proposed by ETSI in their NFV MANagement and Orchestration (MANO) architectural framework [25] is needed. We refer to our solution as *Controller*, and we use the Click module [24], a highly programmable open source environment for components assembly and reconfiguration, to provide a demonstration of performance gains. Note that the *Controller* can also manage other types of resources such as VMs, network resources, storage resources, and software functions.

B. Concept

As mentioned in the previous section, our programmable *Controller* offers different views for control operations, as well as runtime component reconfiguration. Control operations are exposed to the network operator or to third-party service providers to monitor the running networking service and guarantee the negotiated SLA.

The Click module [8, 24] is used to provide a concrete implementation of our concept. We provide an implementation of the *Controller* as a `C++` class: the files `Controller.hh` (header file) and `Controller.cpp` are created. It is a component with *zero input* and *zero output* ports, and it is not meant for packet processing. Our proposed implementation of the *Controller* manages Click components and configurations. For this sake, it provides control functions called *handlers* in Click terminology.

C. Capabilities

Several levels of management are provided by our programmable *Controller*: it allows managing single Click components, a set of active components within a running configuration, or the entire Click configuration. Therefore it supports fine-grained management, coarse-grained management, and the compromise between fine and coarse-grained management

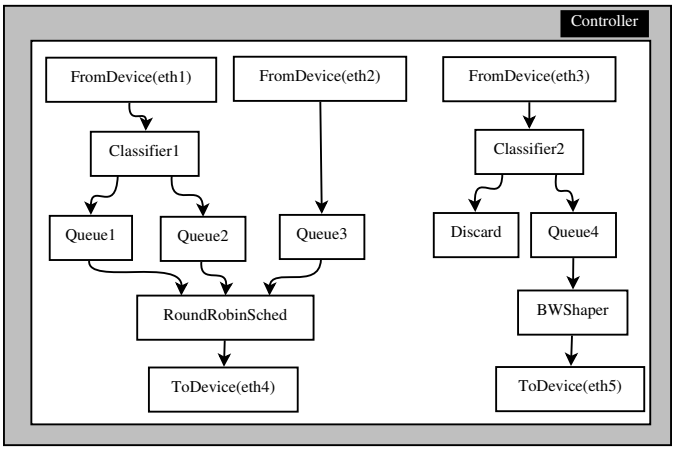


Fig. 2: Click components assembly

(managing a set of active components). A Click configuration example with *Controller* is given in Fig. 2. It is an assembly of components that perform packet processing tasks such as classification, queuing, discarding, and scheduling.

Our proposed *Controller* offers read/write handlers that allow accessing the existing handlers of any component of the running Click configuration in read or write modes. In read mode, it allows getting the values of component attributes such as queue size or capacity, counter value, and packet generation rate of a source. In write mode, it is used to set the values of component attributes, like modifying the rate of a packet generator, resetting a counter, increasing the capacity of a queue, and so on.

Moreover, the *Controller* offers means to easily manipulate a selected set of components. These components are identified either by their names, or by their class. In the following, we define our proposed handlers for managing a set of components:

- `read_class_attribute(Class,att[,rule])` It is a *read* handler that returns a vector filled with the value(s) of the attribute given as parameter for all the components belonging to the class `Class` in the running configuration. Using the keyword `rule` $\in$ {`min, max, mean`} when applicable, *e.g.*, when attributes are integers, we get more precise information about the aforesaid attributes such as the minimum, maximum, or mean value among the selected attributes. These feedback values are used to monitor the running configuration, and serve as indicators to verify whether the established SLA is satisfied or not. Hence, this handler is a useful tool not only for monitoring a set of components, but also for supervising the provided networking service.
- `write_class_attribute(Class,att[,value])` Similarly to `read_class_attribute`, the `write_class_attribute` handler acts on all the components of `Class` in the running Click configuration. However, it is a *write* handler that modifies the aforesaid attribute(s). For instance, if we

want to reset the `count` attribute of all the counters in the configuration without restarting the configuration itself, this operation is now feasible using a single handler call. We can also adjust the `value` of the attribute(s) by providing the new one as parameter when calling the handler. Hence, it is possible to manage the same attribute of several components via a single handler call.

- `manage_component_set(logical_operation)` This handler allows managing a set of heterogeneous components *i.e.*, components belonging to different classes. The logical operation given as parameter manipulates the values of components' attributes in order to provide the requested feedback. Let us consider the following example:
  `manage_component_set("(counter.count > 100 && queue.length > 50) ? true : false")`
  where `counter` belongs to the class *Counter* and `queue` belongs to the *Queue* class. The handler returns `true` if the `counter`'s *count* attribute is greater than 100 and the *length* of the `queue` component is greater than 50; otherwise, it returns `false`. More sophisticated logical operations involving different components could be used to get feedback about the monitored set of components.

We also provide other handlers for managing the entire Click configuration and modifying components assembly. With these handlers, we are able to add, remove, or duplicate components of the Click chain without service disruption. Besides these novel functionalities, the main advantage of our proposed *Controller* is its ability to offer different management views based on administration needs. Before performing any control operation, the scope of any intervention is specified by selecting the related components. For instance, three different management views are illustrated in Fig. 3. The first one selects the components receiving packets over *eth1*, *eth2*, and *eth3* interfaces. In the second management view, different components are concerned *i.e.*, components performing packet classification or scheduling. These views are available thanks to the *Controller*.

Our *Controller* exposes control interfaces to perform monitoring tasks or to execute management decisions at runtime. Component management operations are now available not only to the network operator, but also to third-party service providers, and even to the clients. Therefore, component management becomes flexible and dynamic. It can be performed via remote handlers calls by the operator itself or delegated to its clients. Note that our proposed *Controller* may be manipulated by an administrator, by another *Controller*, or by any other control or management entity. Moreover, it operates independently of the service integration platform.

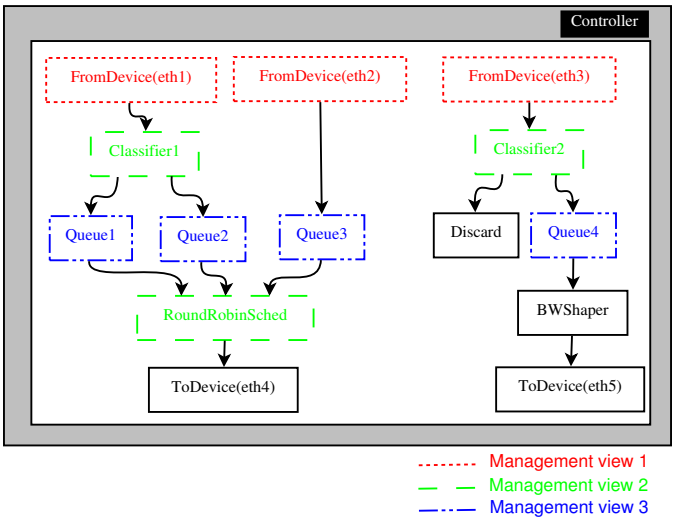


Fig. 3: *Controller*'s management views

Fig. 4: Use-case network architecture

IV. Networking Use Case

A. Use Case Description

In this section, we present and describe a networking use case that illustrates the advantages of our component-based approach. We focus on the capabilities of our *Controller* to supervise a running networking service, to manage components assembly, and to expose components management to third-party service providers.

Let us consider a connectivity service *i.e.*, providing access to Internet Protocol (IP) services, such as Internet browsing, video streaming, etc. A network-edge equipment called *box* is in charge of guaranteeing connectivity between the operator's network and users, as well as connectivity between the operator's network and Internet, as shown in Fig. 4.

Software functions providing authentication, connectivity, routing, and other packet processing tasks run on the *box*. In our use-case, the *box* connecting the operator's network to Internet implements Gateway (GW) functions *e.g.*, Authentication, Authorization, and Accounting (AAA) services, router, firewall server, and proxy server functions. It also provides sessions control and signaling functionalities. Users requesting connectivity and other IP services, such as Internet browsing, video streaming, data transfer, etc., are connected to the *box* via their private network. Once the user is authenticated, its access to the requested service is granted according to the negotiated SLA.

Our scenario is illustrated in Fig. 5, where user A connected to *box* A has initially a Web browsing session. User B is connected to *box* B in charge of providing an IP service

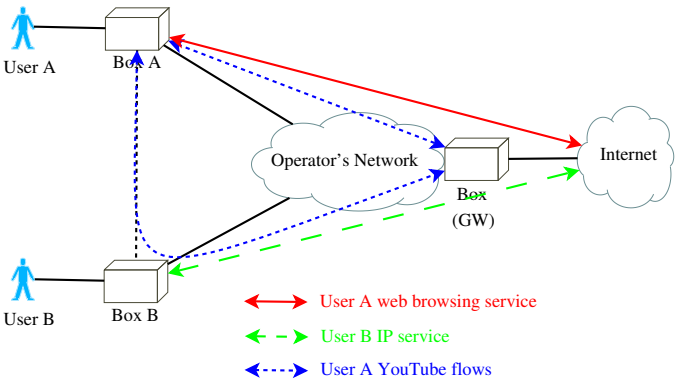


Fig. 5: Connectivity hybridization scenario

required by this user. At a given moment, user A requires a YouTube session in addition to the original Web browsing service. *Boxes* are able to collaborate in order to satisfy user needs in terms of connectivity, caching, and other service requirements. This collaboration, information exchange, and software functions modification are transparent to the user. The provided IP services are the composition of elementary software components deployed on each *box*.

B. Advantages of the Controller

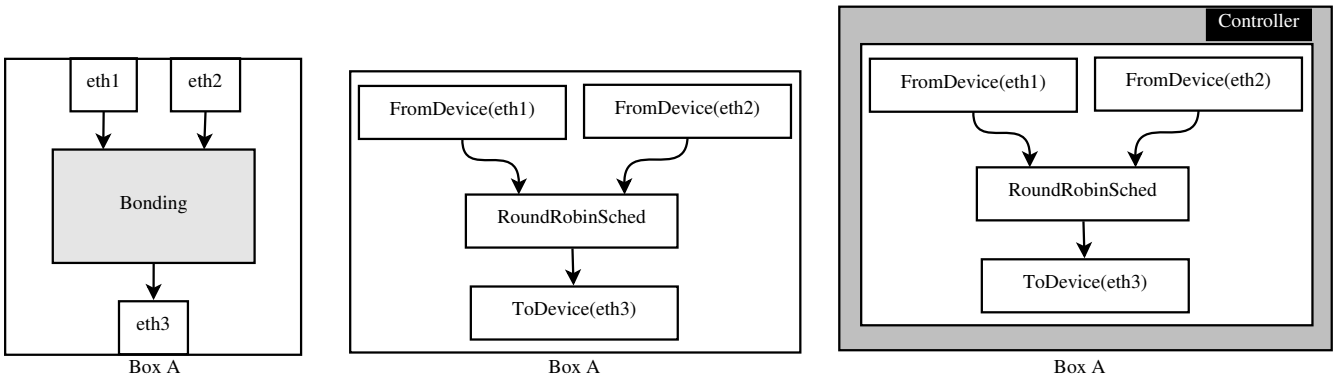
We focus on the networking functions on *box* A, where a Click configuration is in charge of aggregating the flows destined to user A. *Box* A is connected to the operator network and to *box* B via two network interfaces named *eth1* and *eth2*, respectively. Another network interface, called *eth3*, is used to forward the received traffic to user A. Initially, *box* A forwards the traffic of the web browsing session received on *eth1* to user A over *eth3* interface. After initiating the YouTube session, the related packet flows are received on both interfaces *eth1* and *eth2*. The configuration of *box* A is adjusted accordingly.

The aggregation of packet flows is a networking service required at *box* A to fulfill QoS requirements of user A. The establishment of this service could be carried out through different solutions such as those described in the following scenarios. In the first scenario we use the Linux Bonding interface, which is a virtual interface resulting of the aggregation of several network interfaces. It is shown in Fig. 6a where the bonding module merges interfaces *eth1* and *eth2* into a virtual interface. The aggregated traffic is therefore forwarded through *eth3* interface. The second scenario is illustrated in Fig. 6b where a Click configuration is created by concatenating a set of atomic software functions. In the third scenario, we add our *Controller* to the same Click configuration of the second scenario. The new configuration is illustrated in Fig. 6c. The *Controller* allows monitoring, supervising, and modifying the running configuration. It also allows component management exposure to external control entities and third-party service providers.

For the three defined scenarios, the aggregation of traffic flows received on *eth1* and *eth2* interfaces is performed by software modules, and the aggregated traffic is forwarded to user A over *eth3* interface. We are also able to check service statistics whether we are using the Linux Bonding module or Click components. Another management purpose is the ability to modify a running networking service on-the-fly *i.e.*, without service disruption. In our use-case, on-the-fly modification is possible for the Linux Bonding module and for the Click configuration with and without *Controller*. However, we note that accessing service statistics and modifying the running configuration are done differently for each scenario. In fact, Linux Bonding requires root access to get statistics related to the networking service, or to modify the running function *e.g.*, add or remove network interfaces to the Linux Bonding interface. This constraint brings on serious security breaches since the supervision entity, which could be the network operator or a third-party service provider, has the root access guaranteed on *box* A when the Linux Bonding module is used. It is authorized to perform any operation, thus jeopardizing the performance and efficiency of the virtual functions running on *box* A, as well as the availability of *box* A itself. On the contrary, root access is not required for monitoring or modifying running Click configurations. There is no room for security threats like in the Linux Bonding scenario.

Another advantage of the Click-based scenarios is that they make use of the component-based approach's ability to offer flexible, modular and dynamically compoundable software functions, according to user demands, service needs, and operator's strategy. For instance, we are able to add other packet processing functionalities, such as packet queuing, filtering, switching, etc., besides the original aggregation function. The added services are dynamically customizable in response to service requirements, contrarily to static approaches where software modules installed in response to a previously identified need are not expandable to support additional networking services. Moreover, Click offers unified procedures for network functions monitoring, for the dynamic adjustment of components' parameters, and for on-the-fly modification of running networking functions. In fact, each of these operations is performed in a specific manner for Linux Bonding, which induces a lack in procedure unification and generic control.

The insertion of our proposed *Controller* allows us to make use of the advantages of the Click module along with additional improvements. The *Controller*'s handlers allow the management of elementary components, compound components, sets of heterogeneous components, and the running configuration as a whole. Thus, it offers multiple management views for performing service supervision and monitoring. Another advantage of the *Controller* is service exposure to the network operator, to other control entities, and to third-party service providers. The control functions (handlers) for the different levels of management are unified, and the process is secure and transparent. Moreover, these control functions are exposed at runtime to any user such as an external controller or a third-party service provider. They are now able to monitor the running service, to collect statistics about the processed packets, and to adjust the running components or a given set



(a) Interface bonding (b) Click configuration (c) Click configuration with *Controller*

Fig. 6: Software functions running on *box* A

of components on-the-fly.

The *Controller* renders supervision automation easier by offering, in the first place, the means to monitor and analyze the status of one or several components. In the second place, it provides the required functions to execute the decisions made after the analysis and planning phases. Therefore, it is a component-based enabler of the autonomic MAPE loop. The described characteristics, advantages, and limitations of the three scenarios are summarized in Table I.

TABLE I: Scenarios comparison

Feature \ Approach	Bonding	Click	Ctrl
Packet processing	✓	✓	✓
Service statistics	✓	✓	✓
On-the-fly modification	✓	✓	✓
Security	✗	✓	✓
Modular functionalities	✗	✓	✓
Unified procedures	✗	✓	✓
Control views	✗	✗	✓
Component exposure	✗	✗	✓
Supervision automation	✗	✗	✓
MAPE loop	✗	✗	✓

V. Controller Evaluation

A. Number of Instructions

We compare the number of instructions required to perform different component management tasks for the three scenarios: Linux Bonding module, Click-only module, and Click with our proposed *Controller*. The number of required instructions for each scenario is shown in Table II.

As explained previously, the Linux Bonding module allows on-the-fly modification of the running configuration, but it only permits adding or removing a network interface. Other networking functionalities, such as packet classification, queuing, and firewall functions, cannot be added on-the-fly to the Bonding module. Adding or removing a network interface and managing one component (read and write operations) have the same cost in terms of number of instructions for the three scenarios.

TABLE II: Complexity comparison

Feature \ Approach	Number of instructions: Bonding	Click	Ctrl
Add/Remove interface	1	1	1
Manage one component	1	1	1
Add functionalities	✗	1	1
Manage k components	N/A	k	1
Get aggregate feedback	✗	✗	1

Contrarily to the Linux Bonding scenario, the component-based approach illustrated by Click allows on-the-fly modification of a running VNF so that additional functionalities can be added without service disruption. This is achieved using the `hotconfig` generic handler. Hence, only one single instruction is required to modify a running configuration on-the-fly. Besides the exposure of component management operations to the operator, the client, and to third-party service providers, our *Controller* simplifies these management operations. For instance, it is possible to perform read or write operations on several components, such as instances of the same class or different classes, via one single handler call. This feature is not available under the Linux Bonding module, since it only manages network interfaces. Using the *Controller*, the number of instructions required to manage one or several components does not depend on their number, type, or composition.

Finally, getting an aggregate feedback from a heterogeneous set of components is mandatory to enable automated supervision. This feature is now available and exposed at runtime to the operator, the clients, and to third-party service providers, thus enabling the implementation of an autonomic MAPE loop. Like the previously described management tasks, getting an aggregate feedback costs only one instruction. Note that the *Manage* and *Execute* operations are implemented and exposed by our *Controller* to a third-party user, that can be a similar *Controller*, a network administrator, or a different

management component. *Analyze* and *Plan* operations depend on the negotiated SLA. They can be performed locally by each *Controller*, or by a centralized control entity that makes its decisions based on *Controllers*' feedback.

B. Response Time

We compare the time required to modify a running Click configuration on-the-fly using the generic handler `hotconfig` provided by Click, and via our *Controller*. The Click module runs on an Ubuntu virtual machine having 1 MB of random access memory. The mean response time along with the 95% confidence interval for these two scenarios are illustrated in Fig. 7.

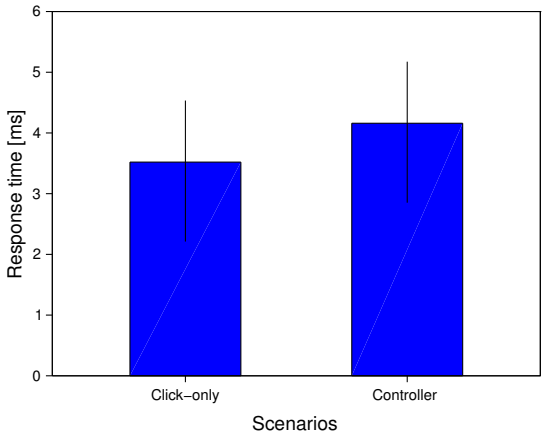


Fig. 7: On-the-fly configuration response time

According to these results, the time required to modify a running configuration is slightly increased when using our proposed *Controller*. Although it allows flexible and generic component management operations, the usage of the *Controller* creates an additional layer for these operations. We make use of the advantages and the novel features introduced by the *Controller* at the expense of a management overhead. However, the cost of this overhead in terms of response time is negligible. It is less than 1 ms when performing `hotconfig` operations, as shown in Fig. 7.

C. Response Time Versus Number of Components

We also evaluate the time required to execute read and write operations using Click-only handlers and via our proposed *Controller*. The response time for read/write operations is measured for several Click configurations, where the number of components per configuration is increased. Response time measurements are illustrated in Fig. 8.

Read operations are used to perform a supervision task in order to request components' feedback, while write operations are used to execute the decision made after analyzing the received feedback. In Fig. 8, *Read* and *Write* curves correspond to the operations of the Click-only module, and the remaining curves are related to different implementations of the *Controller*. *Read Ctrl* and *Write Ctrl* correspond to a Click-specific

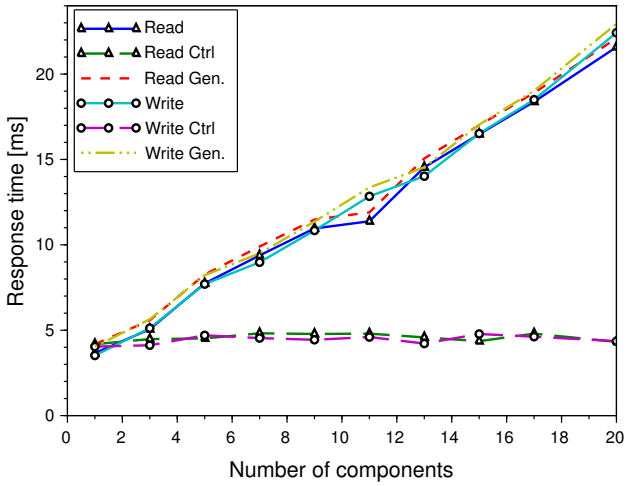


Fig. 8: Response time versus number of components

implementation of the *Controller*, where components built-in handlers are not reused. Nevertheless, *Read Gen.* and *Write Gen.* represent a generic implementation of the *Controller*, where the existing Click handlers are reused, and the *Controller* only offers the means to execute components' handlers.

We notice that the time required to execute read or write operations using Click-only handlers increases with the number of controlled components. In fact, the components are managed via a sequential call of their handlers. Similarly, when the *Controller* reuses the existing handlers (generic implementation), the same behavior is found with a slight increase due to the management overhead. Note that this time overhead is the same as for the `hotconfig` generic handler. Nevertheless, response time for read and write operations does not increase with the number of managed components when the Click-specific implementation of the *Controller* is used, since it does not perform a sequential call of the managed components' handlers.

VI. Conclusion

The fast evolution of network virtualization technologies leads to the deployment of networking functions over virtualized infrastructures. Our aim is to unify the management of heterogeneous resources including networking resources, such as routers, firewalls, and points-of-presence, and IT resources such as processors and storage. This unification promotes the slicing of networking and IT resources and enables flexible 5G architecture management. A networking service is constructed by assembling several components according to the negotiated SLA. However, the need for full network virtualization, flexible control functions, and components management exposure to third-party service providers requires dynamic, customizable, and programmable controllers.

In this paper, we introduced a novel programmable *Controller* that meets the demand for flexible and efficient com-

ponent management. We define the *Controller*'s specifications, and we present a concrete implementation using the Click module. Our *Controller* offers various features for reconfiguration and exposure of components and VNFs at runtime. It permits unified components management exposure, and provides various management views, *i.e.*, fine or coarse-grained, depending on operator, client, or third-party applications' needs. The proposed component-based approach overcomes the limitations of state-of-the-art approaches, such as Linux Bonding and Click modules used in our networking use case, at the expense of a slight increase in response time. In a future work, the *Controller* will be extended to deal with other types of resources.

ACKNOWLEDGMENTS

Our programmable *Controller* is proposed in the context of Orange Labs Global Operating System (GlobalOS) project. GlobalOS is a Software platform and infrastructure that operates networking and IT resources. It offers flexible and dynamic services, ensures resource abstraction and exposure, and provides management and control functions. The authors would like to express their gratitude to Arnaud Monteilhet and Pierrick Seite for their contribution to the accomplishment of this work.

REFERENCES

[1] Y. Li and M. Chen, "Software-Defined Network Function Virtualization: A Survey," *IEEE Access*, vol. 3, pp. 2542–2553, December 2015.

[2] R. Jain and S. Paul, "Network Virtualization and Software Defined Networking for Cloud Computing: A Survey," *IEEE Commun. Mag.*, vol. 51, no. 11, pp. 24–31, November 2013.

[3] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, January 2015.

[4] H. Kim and N. Feamster, "Improving Network Management with Software Defined Networking," *IEEE Commun. Mag.*, vol. 51, no. 2, pp. 114–119, February 2013.

[5] C. Trois, M. D. D. D. Fabro, L. C. E. de Bona, and M. Martinello, "A Survey on SDN Programming Languages: Towards a Taxonomy," *IEEE Commun. Surveys and Tutorials*, vol. PP, no. 99, April 2016.

[6] 5GPPP, "View on 5G Architecture," White Paper, July 2016.

[7] S. Aust, J.-O. Kim, P. Davis, A. Yamaguchi, and S. Obana, "Evaluation of Linux Bonding Features," in *Int. Conf. Commun. Technology*, Guilin, November 2006.

[8] E. Kohler, "The Click Modular Router," Ph.D. dissertation, Massachusetts Institute of Technology, 2001.

[9] S. Clayman, L. Mamatas, and A. Galis, "Energy-Efficiency Enablers and Operations in Software-Defined Environments," in *IEEE/IFIP Network Operations and Management Symp.*, Istanbul, April 2016.

[10] A. A. Mohammed, M. Gharbaoui, B. Martini, F. Paganelli, and P. Castoldi, "SDN Controller for Network-Aware Adaptive Orchestration in Dynamic Service Chaining," in *IEEE NetSoft Conf. and Workshops*, Seoul, June 2016, pp. 126–130.

[11] CoreGRID, "Basic Features of the Grid Component Model," Deliverable D.PM.04, March 2007.

[12] R. Riggio, I. G. BenYahia, S. Latre, and T. Rasheed, "Scylla: A Language for Virtual Network Functions Orchestration in Enterprise WLANs," in *IEEE/IFIP Network Operations and Management Symp.*, Istanbul, April 2016, pp. 401–409.

[13] ETSI GS NFV-IFA 009, "Network Functions Virtualisation; Management and Orchestration; Report on Architectural Options," ETSI, Deliverable V1.1.1, July 2016.

[14] Canonical, "Juju." [Online]. Available: https://www.ubuntu.com/cloud/juju

[15] ——, "Juju Charms." [Online]. Available: https://jujucharms.com/

[16] C. Basile, A. Lioy, C. Pitscheider, F. Valenza, and M. Vallini, "A Novel Approach for Integrating Security Policy Enforcement with Dynamic Network Virtualization," in *1^{st} IEEE Conf. Network Softwarization*, London, April 2015, pp. 1–5.

[17] R. Mijumbi, J. Serrat, J. L. Gorricho, N. Bouten, F. D. Turck, and R. Boutaba, "Network Function Virtualization: State-of-the-Art and Research Challenges," *IEEE Commun. Surveys and Tutorials*, vol. 18, no. 1, pp. 236–262, January 2016.

[18] IETF, "YANG - A Data Modeling Language for the Network Configuration Protocol," Request for Comments: 6020, October 2010.

[19] J. O. Kephart and D. M. Chess, "The Vision of Autonomic Computing," *IEEE Computer*, vol. 36, no. 1, pp. 41–50, January 2003.

[20] P. Arcaini, E. Riccobene, and P. Scandurra, "Modeling and Analyzing MAPE-K Feedback Loops for Self-Adaptation," in *IEEE/ACM 10^{th} Int. Symp. Software Engineering for Adaptive and Self-Managing Systems*, Firenze, May 2015, pp. 13–23.

[21] J. M. Sanchez, I. G. BenYahia, and N. Crespi, "Self-Modeling Based Diagnosis of Services over Programmable Networks," in *2^{nd} IEEE NetSoft Conf. and Workshops*, Seoul, June 2016, pp. 277–285.

[22] T. Dietz, R. Bifulco, F. Manco, J. Martins, H.-J. Kolbe, and F. Huici, "Enhancing the BRAS Through Virtualization," in *1^{st} IEEE Conf. Network Softwarization*, London, April 2015.

[23] R. Morris, E. Kohler, J. Jannotti, and M. F. Kaashoek, "The Click Modular Router," *ACM SIGOPS Operating Systems Review*, vol. 33, no. 5, pp. 217–231, December 1999.

[24] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click Modular Router," *ACM Trans. Computer Systems*, vol. 18, no. 3, pp. 263–297, August 2000.

[25] ETSI GS NFV-MAN 001, "Network Functions Virtualisation; Management and Orchestration," ETSI, Deliverable V1.1.1, December 2014.