Usage Control Policy Enforcement in SDN-based Clouds: A Dynamic Availability Service Use Case

***Abstract*—With the growing interest in Software Defined Networking (SDN) and thanks to the programmability provided by SDN protocols like OpenFlow, network application developers have started implementing solutions to fit corporate needs, like firewalls, load balancers and security services. In this paper, we present a novel solution to answer those needs with usage control policies. We design a policy based management framework offering SDN network security policies. This approach is used to enforce performance requirements (e.g., to ensure a certain level of network connectivity). A top-down approach is proposed, in order to refine the policies into the appropriate network rules, via the OpenFlow protocol. Finally, we implement the solution with an availability service use case and we provide a set of experiments to evaluate its efficiency.**

***Keywords*-Usage Control; SDN; OpenFlow; Network Policies; Service Availability**

I. Introduction

Everything-as-a-Service is emerging as a business transformation approach, far exceeding the usual cloud service provider stack (IaaS, PaaS, SaaS) and attracting many players to the market. This often implies a relaxed security in an environment where the concepts of external threats and internal assets is now blurred [6]. In such clouds, ensuring the availability of this wealth of diverse services may require as much flexibility as it is needed to orchestrate them. In this regard, the Software Defined Network (SDN) paradigm is a realistic contender to enhance the security of a cloud computing environment thanks to its network-wide visibility and programmability features. It introduces an additional abstraction layer between the control plane and data, which are now decoupled. Decision process is now centralized in a single component, namely the SDN controller, which offers a global view of the network to the operators. Thus, they are able to anticipate network resources misuse and plan network bandwidth allocation more efficiently for the hosted services.

Recent research works have already highlighted the promises of SDN by proposing new security services [15], [16], [18], [13]. In particular, countering massive threats such as distributed denial of service (DDoS) attacks has been extensively studied [18] as SDN key features are expected to minimize the delay from detection to mitigation. Other aspects of the SDN paradigm have also been leveraged: FRESCO [15] provides a modular framework to develop security applications over SDN while SIMPLE [16] builds upon the decoupling of the control plane from the data plane to ease the process from policy expression to middlebox traffic steering. Chaining services is made possible not only for physically-hosted services, but also to compose network virtual functions as in Slick [17]. [13] proposes a new energy efficient routing algorithm with SDN for data center networks. This approach can schedule flows to the queues of each link to minimize the energy for the given volume of data center traffic.

Although the importance of these solutions, it still has a critical need on a whole framework that provides orchestration and automatic management of network policies [7]. Indeed, we need a flexible and generic mechanism that permits to respond automatically to the detected alerts and problems. Moreover, the solution should provide a high abstract description of the security services to be used with different cloud providers without dependence from their used technology.

However, SDN and the OpenFlow protocol, a de-facto standard for the southbound interface, do not allow to trivially acquire services requirements and translate them into forwarding policies. Indeed, among the previous examples, writing applications that could be directly enforced in the network usually relies on a domain-specific language, slowing its adoption. On the other hand, when high-level languages are provided to express policies, they can only be translated to the extent of a service chain, which is not directly deployable to the forwarding plane. Therefore, there is a need for a policy-based management framework able to collect high-level inputs from operators and provide appropriate reaction scripts that can configure SDN equipments. Additionally, the programmibility of the network is rarely exploited to introduce autonomicity: a policy management approach should be context-aware, so that policies are dynamically enforced depending on the network and security states.

In this paper, we propose a policy-based management approach that collects and processes the output of monitoring tools (e.g., incident alerts and network metrics) and provides the necessary OpenFlow rules to reconfigure the data plane. Our framework offers a set of formal security policies and SDN templates which will be used to automate the network configuration and reactions by managing and deploying rules, as well as, detecting potential conflicts. For this purpose, we use a high level formalism, the OrBAC [2] model, which main benefits are: a) the introduction of a distinctive entity, the *organization*, permitting the representation of global entities in which a policy is applied, e.g., an SDN infrastructure, b) the usability of administration, c) the diversiy of access types (permission, prohibition and obligation), and d) the definition of a high abstraction level which simplifies the management of policy rules and the detection of conflicts.

Moreover, our framework includes a new policy instantiation engine to map OrBAC policies and monitoring notifications for context awareness, and an SDN policy deployment engine to translate the inferred rules into OpenFlow configurations. This last feature allows an administrator to write rules for an SDN infrastructure without deep knowledge of the OpenFlow standard. Different security services may be implemented easily with our framework such as availability, attack mitigation, etc. In this paper, we will demonstrate the availability service as a case study.

The rest of the paper is structured as follows. In Section II related work is discussed. In Sections III, IV and V, the proposed framework, architecture, components and workflow, are detailed respectively. In Section IV, the experimentation is presented and discussed. Finally, in section V, the conclusion with some lines of future work are presented.

II. RELATED WORK

One of the salient features of the SDN paradigm is that it makes programming the network possible, but not necessarily easy. As a matter of fact, many controller platforms require that high-level policies be managed at the individual switch level. The community generally agrees that SDN applications should specify high-level policies in two distinct parts: an endpoint policy, that specifies actions to packets at the controller domain level; a routing policy, that specifies paths taken by the traffic according to traffic engineering considerations [14]. However, SDN comes in with some challenges. Among them, dynamic policy enforcement and conflict detection are quoted. Kreutz et al.. [4] have outlined them in their SDN survey. Indeed, leveraging SDN control languages to program networks according to some policy is not straightforward, according to Foster et al. [9]. Their early state-of-the-art analysis of the OpenFlow controller languages has exposed some substantial difficulties: (1) network functions written in these languages do not compose; (2) low-level hardware-dependent semantics complexify how rules are written and distributed; (3) network programs' two-tiered architecture (spanning both the controller and the switch) incurs concurrency issues. To address these issues, they proposed Frenetic [9], a high-level network programming language, and in particular, a network policy management library that includes a set of source-level operators for constructing and manipulating streams of network traffic. They later extend their work in the Pyretic language [10] by introducing an abstract packet model, a sequential composition operator and a topology abstraction using network objects. Concurrently, Voellmy et al. have proposed Procera [11], a functional reactive programming language like Frenetic, with which it shares similar constructs. But unlike Frenetic which applies only to packets, Procera handles any event stream allowing for both sequential and parallel composition, and uses function values to represent high-level policy. While these languages were all declarative, Voellmy et al. took another direction when proposing the Maple language [12] which introduces *algorithmic policies*, high-level policies that decide the behaviors of an entire network. Kang et al. have considered an abstraction similar to Maple, i.e., the whole network is *one big switch* [14], but are more concerned than other proposals on how to satisfy user-defined policies and adhere to the underlying hardware resources constraints at the same time. In contrast, FRESCO offers a development environment that is specialized to serve the needs of security applications [15]. It facilitates the rapid design, and modular composition of OF-enabled detection and mitigation modules.

In this paper, we are leveraging a common usage control model language to specify the endpoint policy of the network. As OrBAC is an all-purpose policy model language, it is expected to be more powerful in dealing with contexts, conflicts and usages, which can be wider than the scope of the existing DSLs. The particular use case we present is a proof that programming the network can be addressed as designing an organization policy. For the moment, the results are similar to legacy mechanisms used to provide resilience to cloud services (replication, load-balancing, etc.). However, leveraging the SDN paradigm for network virtualization allows for increased flexibility in managing resources, which can be costly in replication strategies.

III. SDN Policy Enforcement and Management Framework

In this paper, we design a new framework that will offer a high description of network security policies. It will permit to react and re-configure dynamically an SDN network against incidents and attacks. Different components are defined:

- **Context and Monitoring Module - CMM:** A monitoring service must be defined in order to provide reports on the status of VMs, the network and the detected attacks or malicious behaviors. These status will define conditions or alerts that monitor the changes in the system behavior. Several intrusion detection systems and network monitoring tools may be used in a multi-cloud environment. In this paper, we do not focus on the choice or the update of a monitoring solution. However, we will work on the management of the received notifications in order to define contexts that will be used to activate or deactivate the adequate reaction policies.

- **Security Policy Module - SPM:** SPM is composed of two sub components 1), Security Policy Modeling and 2), Security Policy Enforcement Engine. The SPM specifies and models the security requirements to be taken into account within the applications. The security policy to be applied is a set of rules defining the objectives that need to be respected by the system. It includes different rule types: permissions, prohibitions and obligations. Section IV presents how to specify and model the SDN security policy using the Or-BAC model and it gives different elements and components used to build such a policy. Whereas, policy decision engine is the architecture engine that will receive the requests and the notifications. Based on them, it will activate and deactivate the adequate policies and rules. This engine has to manipulate a set of pre-defined

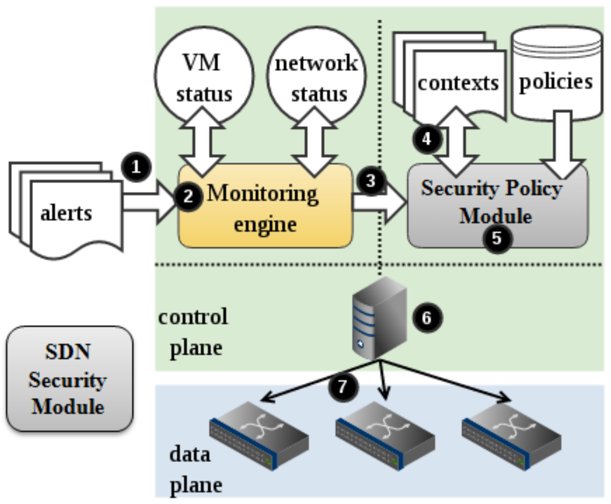

Figure 1. Workflow of the policy enforcement approach

network security policies based on the activated context. More details about this module is presented in Section IV-B.

- **SDN Security Module - SSM:** SSM is responsible for the implementation of the current policy in the Policy Enforcement Points (PEPs). PEPs comprise the OpenFlow switches and the SDN controller. The PEP will enforce automatically the selected policy by the SPM using the OpenFlow scripts. More details about this module could be found in Section V.

Fig. 1 illustrates the workflow of our approach. A monitoring engine continuously supervises the data plane (2) to detect any failures, that may be system- or network-based. This component updates accordingly the virtual machines' status, as well as the network status based on alerts received from local or external sources (1). When resources are critically damaged (failure, error, etc.), the OrBAC-PDP (Policy Decision Point) is notified (3). This policy engine activates contexts based on the status of the networks or virtual machines (4). Appropriate remediation policies are then selected and pushed into the data plane in order to preserve the availability of the targeted services (5, 6, 7). The deployment of these policies is ensured by the SSM module based on predefined SDN templates.

A. *Availability service use case*

With our framework, several network security policies may be proposed as the availability and attack mitigation services. The first one aims to enhance the application performance against system and network failures. It will offer a technique to distribute traffic between different servers when it is needed. It will apply some techniques as the load balancing and the replication strategies based on the SDN technologies and only when some conditions are detected. Indeed, our solution may adapt the dynamic changes of the cloud (adding and deleting VMs, modification of the allocated resources, migration of a VM, etc). The second service is responsible to react against the detected attacks. It offers the cloud providers to complement their practical defense systems. In the rest of this paper, we will focus more about the availability service that will be used as a case study of our proposal. All the presented examples will be related to this service.

In this section, we present two possible reactions that are managed by our availability service.

- The first one is based on the use of a replica server with degraded Quality of Service (QoS). This scenario presents the case where the provider pledges to preserve the availability of the service without necessarily maintaining the quality of the network. The replica is deployed to receive live copies of user requests, but does not honor them as long as the main server is running. However, if connectivity to the main server is lost or get degraded, the replica takes over the main server and see its QoS increased to act as the main service:
  During the whole lifecycle of the service, two servers are kept alive, one, the main server, with the requested QoS and the second, the backup or replica, with degraded QoS. The role of the availability policy is to manage the performance of the networks and to redirect the flows to the replica with the modification of its QoS in case of a connection problem with the main server. In other words, even if the main server could not provide the required QoS, we provide new paths with the needed performances for the replica to keep the requested QoS.
- The second service proposes a load balancing on demand. At the beginning, only one server provides a service to the customers. Next, based on the network status, a second server is provided in order to achieve the required QoS. This policy provides network resilience for the customer's service by categorizing the network status into three different states. The *NORMAL* state, where the service can answer requests properly and does not suffer from bandwidth nor computation power shortage. The *MIDDLE* state corresponds to a service with several simultaneous connections or less than 50% of its available bandwidth remaining. Finally, the *CRITICAL* state characterizes an offline system. In this scenario, the second machine is solicited depending on the state of the main machine.

IV. Security Policy Module - SPM

A. *Modeling Security Policies*

Based on the different requirements and challenges highlighted in the Section I, we need to select a security policy model which is capable of modeling various types of security requirements (access control, usage control, and information flow, etc.), provide a way to enforce security requirement dynamically, and most importantly security policy model should be capable of handling conflicts in security policies. In this section, we will present our solution to model SDN network which is adequate for handling most of these requirements.
Traditionally security policies define a set of rules which are intended to control the access to resources and their usage during this access. RBAC [1] is the most well

known model defining security policies as a set of permissions. Permissions are rules granting the right to access a specific resource. The approach used by RBAC model to specify security policies is static. Later, the OrBAC model [2] proposed to introduce another type of rules named prohibitions which are denying rights to access specific resources. Obligations are the third type of rules specified to enforce the usage control. They are actions that the system or users are required to take. These actions are essential for the expression and enforcement of a large number of requirements. In SDN network, we consider these three types of security rules. We model these rules using the OrBAC model. Instead of modelling the policy by using the concrete implementation-related concepts of subject, action and object, the OrBAC model suggests reasoning with the roles that subjects, actions or objects are assigned within an organization. The role of a subject is simply called a role as in the RBAC model. The role of an action is called activity and the role of an object is called view. The security rules specify that some roles are permitted or prohibited or obliged to carry out some activities on some specific views under certain conditions. The main advantage of OrBAC model in the SDN network is that the security rules do not apply statically but their activation may depend on contextual conditions [2]. For this purpose, the concept of context is explicitly included in OrBAC. Many types of contexts have been defined within the OrBAC model in order to take into account different types of conditions. The activation of some contexts activates the rules defined under that specific context.

1) Specification of Access and Usage Control Rules: Using the concepts introduced by the OrBAC model and based on a first order logic formalism, the specification of the abstract permissions or prohibitions is defined using a 5-places predicate as follows:

Definition 1: an OrBAC security rule is defined as: `Security_Rule (Org, Role, Activity, View, Context)` where `Security_Rule` $\in$ {`Permission, Prohibition`}.

The obligations differ from permissions/prohibitions by controlling the behavior of the system based on specific events that may occur. For the definition of obligations, we consider two different contexts: the obligation's activation context and the obligation's violation context.

- *Obligation Activation:* An obligation has an activation event after which it becomes effective. This event may be a temporal or an action-based event [3].
- *Obligation Violation:* An obligation has a violation event which specifies when it is violated. This event may be an action-based, temporal or a relative temporal deadline [3].

From its specification until its fulfillment, an obligation can have different status, apart from *Obligation Activation* and *Obligation Violation*:

- *Obligation Deactivation:* An active obligation is deactivated when its context ceases to hold. The deactivation of an obligation depends on the type of the obligation. Some obligations may remain required forever after their activation until they are fullfilled [3].
- *Obligation Fulfillment:* An obligation is fullfilled when its action is taken. Thus, the obligation ceases to be required [3].

Thus, the specification of the obligations can be expressed as follows:
Definition 2: an OrBAC obligation is defined as:
`Obligation (Org, Role, Activity, View, Activation_Context, Violation_Context).`

Considering our OrBAC security policy model, we have designed a set of entities and components that will permit to define an SDN security policy. In the following we present some examples of privileges and conditions for our SDN network availability use case (cf. section III):

- P_1: A subscribed system administrator in Cloud Service provide (CSP) facility is authorized to access various SDN controller services in a default context:
  - P_1: `Permission(CSP, system-admin, access, sdn-controller-services, default-context)`
- P_2: The SDN controller is allowed to activate loadBalancing-on-demand which is running on the main server by using the best path in a normal context:
  - P_2: `Permission(CSP, SDN-Controller, activate, loadBalancing-on-demand, normal-context)`
- O_1: The SDN controller is obliged to self-activate loadBalancing-on-demand which is running on the main-server, when the network status is partially critical:
  - O_1: `Obligation(CSP, SDN-Controller, self-activate, loadBalancing-on-demand, partial_critical_network_status, system-timeout)`
- O_2: The SDN controller is obliged to redirect loadBalancing-on-demand to the second-server, when the network status is critical:
  - O_2: `Obligation(CSP, SDN-Controller, redirect, loadBalancing-on-demand, critical_network_status, system-timeout)`

Based on the OrBAC derivation mechanism [2], we first define the set of concrete subject [admin-user-1, controller-1], action [access-cmd, activate-cmd, self-activate-cmd, redirect-cmd] and objects [sdn-service-pack, s1]. In the next step, we start by empowering, considering, and using these subjects, actions, and objects into roles, activities and views, respectively. For instance, we empower admin-user-1 subject into system-admin Role, access-cmd action is considered for Access activity, and we use sdn-service-pack object into sdn-controller-services view. Thus, the concrete rules look like as follows:

- $P_{1'}$: `Is_Permitted(admin-user-1, access-cmd, sdn-service-pack)`
- $P_{2'}$: `Is_Permitted(controller-1, activate-cmd, s1)`
- $O_{1'}$: `Is_Obliged(controller-1, self-activate-cmd, s1)`
- $O_{2'}$: `Is_Obliged(controller-1, redirect-cmd, s1)`

As mentioned before, the approach that we are considering is dynamic since it takes into account the changes that may happen during the application execution. These changes are modeled in the security policies (as shown in security rules: P_1, P_2, O_1, O_2) as a specific context. Thus, in order to invoke the permissions or obligation, the activation context should. More specifically for obligations, when *Activation Context* is evaluated as true, the obligation is considered active and it should be fulfilled before the *Violation Context* holds. Whereas, if *Violation Context* holds, then the obligation is considered violated. In section IV-B, we will explain in a detail how the context is managed and evaluated dynamically to enforce security policies.

2) Conflict Management: In SDN networks, security rules can be hierarchically structured so that they are inherited in the organization, role, activity and view hierarchies. Since a security policy can be inconsistent because of conflicting security rules (for example a permission can be in conflict with a prohibition). Our solution should be able of managing the conflicts at abstract levels. In our approach, we provide two possible reactions regarding this problem:

- **Separation Constraints:** When assigning a concrete entity to an abstract one, the separation constraints are verified to prevent the user from violating one and potentially generate concrete conflicts. The separation constraints are not exclusively used to prevent concrete conflicts but can be used to specify separation of duty.
- **Rule Priorities:** Define a priority order between the two conflicting rules. We associate the authorization rules with priorities in order to evaluate their significance in conflicting situations. Priorities between access control rules may be sometimes derived from the rules syntactical format.

B. OrBAC-PDP:Security Policy Enforcement Engine

In the previous sections, we introduced our concepts of SDN security policies. In this section, we present how the policy engine module analyzes and interprets these security policies and dynamically enforce them. This module acts as a Policy Decision Point (OrBAC-PDP) with a public SDN API in order to manage security policies. It allows to take a decision (allow/deny) according to specific actions to be applied within the application and also trigger obligations for example in the case of usage control rule enforcement or in the case of any security flaw detection. More specifically, once security policies are loaded in the OrBAC-PDP, the next step is to dynamically control the enforcement of these security policies based on the changed occurring in the environment. Thus, the set of active/inactive policies needs to be synchronized with the set of active/inactive rules which are already enforced by the SDN controller. For this purpose, we have defined an enforcement approach based on the concept of push and pull modes, depending on the type of the security rules that we are considereing.

- **Pull Enforcement:** *is a type of mode in which SDN controller invokes OrBAC-PDP to retrieve all the active rules related to permissions and prohibitions*. The SDN controller calls, using the variables <subject, action, object>, the API method Is_permitted in order to evaluate the access as shown in code snippet in the Listing 1.

```
resultPermission = policy.is_Permitted(
     _subject, _action, _object);
resultProhabition = policy.is_Prohibitted(
     _subject, _action, _object);
   if ((resultPermission == true) && (
       resultProhabition ==false)){
      notificationList.prepareActiveRuleList(
           policy);
     } else{
      notifyNegativeEvalutationResult(_subject
            + "is not allowed to perform the
           specific" +_action + "on the" +
           _object);}
```

Listing 1. Security policy pull enforcement

While evaluating the security policies, the OrBAC-PDP consults the context and monitoring module in order to retrieve an actual status of the context as shown in Figure 1. Based on the context status and the elements in the query, the OrBAC-PDP checks its concrete rules (see rules $P_{1'}, P_{2'}, O_{1'}, O_{2'}$) to decide about the access. In our approach as a response the OrBAC-PDP send a list of active and inactive rules to the SDN controller. Based on this list, as shown in Listing 3, the SDN controller activate or deactivate different network configurations.

- **Push Enforcement:** *is a type of mode in which, instead of SDN controller, the OrBAC-PDP pushes the changes, that occur in the security policy, to the SDN controller in the form of notifications to activate or deactivate different network configurations.* Our core purpose to define push mode is to control obligation enforcement during the application execution. As discussed in the previous section IV-A, these obligations can have different states (Activation, Deactivation, Violation, Fulfillment), during the whole lifecycle of the application execution. In our approach, we enforce these different changes of obligation through considering and evaluating different states of the context. For this purpose, the context and monitoring modules are actively monitoring the changes happening in the environment and regularly notify the status of context to the OrBAC-PDP. Once the OrBAC-PDP has been notified that a context is in a new state, an analysis phase takes place in the OrBAC-PDP to check if there is any obligation that is activate, deactivate, fulfilled or obligation is violated.

```
while(_context.getContextState())
  {
  resultObligation= policy.Is_Obliged(_subject
     , _object, _action);
    if (resultobligation == true)
    {
      for (Iterator<CConcreteObligation> it =
            policy.GetConcreteObligations().
            iterator(); it.hasNext();) {
                 CConcreteObligation
                      _obligation = it.next();
                 if (_obligation.IsActive()) {
                     notificationList.
                          prepareActiveRuleList
                          (policy);
                     ... } } } }
```

Listing 2. Security policy push enforcement

Thus, based on this analysis, the OrBAC-PDP retrieves all the active obligations, see code snippet in Listing 2, and push these active obligations to the SDN controller to dynamically enforce security policies. In our approach, we use the same message format defined in Listing 3 to notify active obligations. The SDN controller receives notification about a new security policy to be deployed, or about changes on the current security policies deployed and generates a new security adaptation plan (a list of actions – activate/deactivate/reconfigure – to be performed in the network) conformed to the security policy.

```
<xml version ="1.0" encoding="UTF-8"standalone
     ="Yes"?>
  <Notification>
    <RuleStatus>
      <Rules>
        <Rule Id="P1", Type ="Permission",
             Value ="active">
          <Status>"active"</Status>
          <Subject>"admin-user-1"</Subject>
          <Action>"access-cmd"</Action>
          <Object>"sdn-service-pack"</Object>
        </Rule>
        <Rule Id="P2", Type ="Permission",
             Value ="inactive">
          <Status>"active"</Status>
          <Subject>"controller-1"</Subject>
          <Action>"activate-cmd"</Action>
          <Object>"s1"</Object>
        </Rule>
        <Rule Id="O1", Type ="obligation",
             Value ="active">
          <Status>"active"</Status>
          <Subject>"controller-1"</Subject>
          <Action>"self-activate-cmd"</Action>
          <Object>"s1"</Object>
        </Rule>
        <Rule> ... </Rule>
      </Rules>
    </RuleStatus>
  </Notification>
```

Listing 3. Security policy: Active/Inactive rules

V. SDN Security Module - SSM

This module aims to deploy the selected rules by the SPM into the SDN switches. Indeed, any OrBAC security rule should be translated to a set of three possible actions: adding, deleting and/or modifying a rule from the flow table of a SDN switch. In this module, we have designed three SDN templates for each action. The deployment of an OrBAC security is based on the Algorithm 1. It will take two inputs: the security policy to deploy and the network topology that will permit to configure OpenFlow scripts. After this, for each rule, it will translate it to a set of actions with the needed parameters extracted from the network topology (as the server IP address, the switch Identity, the ports, etc). At the end of this loop, it will deploy the needed strategy in the controlled switches based on the rest API of the controller.

Algorithm 1 SSM: SDN policy deployment

Input: Security Policy SP; Network Topology NT
Output: OpenFlow scripts deployed in the SDN switches

```
flow To_add[],To_delete[],To_update[];
      ▷ These arrays will contain respectively the list of
           ▷ OpenFlow rule to add,to delete and to update
for $R_i$ in SP do
    $R_i$.extract_Add_Rule(To_add);
    $R_i$.extract_Delete_Rule(To_delete);
    $R_i$.extract_update_Rule(To_update);
end for
Concertize(To_add,To_delete,To_update,NT);
Deploy_Policy();
```

VI. Implementation and Evaluation

In this section, we evaluate the our framework with a use case on network availability.

A. *Implementation*

In order to implement the use case, we have set up a testbed (see Figure 2) featuring several hardware servers and OpenFlow (OF) switches:

- 4 Alcatel OS6860 switches, OF-capable with 24 ports running at 1 Gbps.
- Two web servers running Apache v2.4.7 providing a set of video files. (Named main server and second server in the following)
- Two clients streaming video from the server using Tapas [5], an adaptive video streaming client. It simulate a video player without adding the computational overhead of the decoding process.
- An SDN controller managing the switches. In these experiments, we have installed Floodlight[1] on a virtual machine. The OrBAC PDP, the security policies and the SDN security module are deployed in the same virtual machine for eased communication. This machine will be responsible for the reaction strategies and the automatic deployment.

[1]Floodlight is an open source controller, available at http://www.projectfloodlight.org/floodlight/

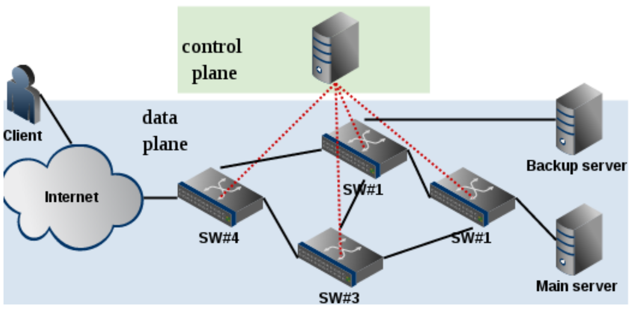

Figure 2. Exprimental testbed.

The use case represents a client watching videos over the internet. In our setup, the clients will be virtual machines and video will be played using Tapas. The testbed will serve as the provider's infrastructure. In this paper, we do not cover the network monitoring and notification processes.

B. Evaluation

Network Failure Scenario: In this experiment, we want to prove the feasibility of our solution when several non-simultaneous network failures occur. The situation is presented in the second availability service (load balancing on demand).

Initial situation: During the initial deployment, the context is set as normal-context. Based on this context the OrBAC-PDP will activate a set of rule. Among them, we find the following rule:

- P_2: `Permission(CSP, SDN-Controller, activate, loadBalancing-on-demand, normal-context)`

These latter will be mapped to openflow scripts that will deploy the best path for the client. As shown in Figure 5, flow rules are pushed toward the different switches during the first period. During this phase, only one client is using the web service. Figure 3 shows an average latency of around 500 μs between client 1 and the server. As shown in Figure 4, the client's requests increases the packet-per-second rate received by the main server.

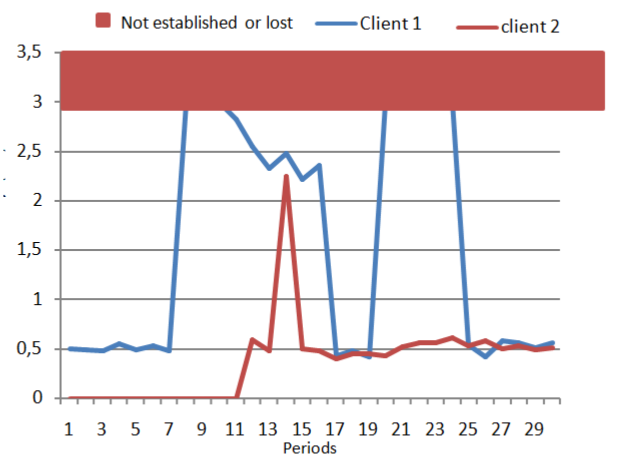

Figure 3. Network failure experiment: The latency of the client 1 and 2.

Incident: A partial network failure occurs, which means a network problem is detected in the best path. We simulate this by unplugging a cable between switch 1 and switch 3. We observe the loss of connectivity at the 9th period in Figure 4 (no data received by the Apache server).

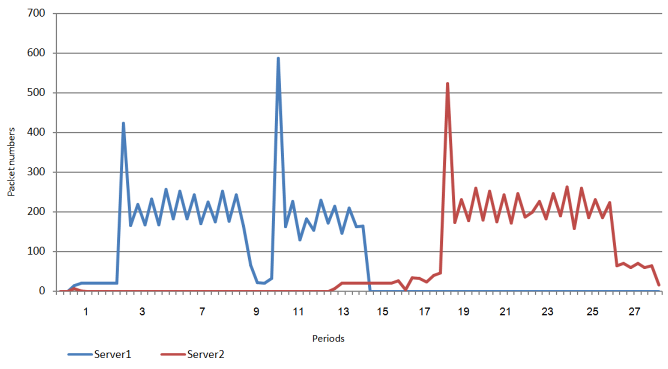

Figure 4. Network failure experiment: Number of packets received by the deployed servers.

Response: Once the failure is detected, the context partial_critical_network_status is activated.

This context triggers the following new rule:

- O_1: `Obligation(CSP, SDN-Controller, self-activate, loadBalancing-on-demand, partial_critical_network_status, system-timeout)`

.

Let's analyse the reaction of our system after the deployment of this rule: Figure 5 from period 11 to 12 shows the OrBAC-PDP deploying the new path, allowing the traffic to reach server 1 with a lower QoS. In Figure 5, between the periods 9 and 12, we notice that flow rules are pushed on the switches. Once new flow rules are effective, the connectivity is re-established for client 1, with a much higher latency though (> 2 ms). In the meantime, a new client attempts to access the service, but is redirected to the backup server in accordance to the availability policy. The Qos for client 2 is the same as for client 1 in a normal situation. Latency in periods [1-7] for client 1 is similar as the client 2 latency during periods [16-29] (see Figure 3).

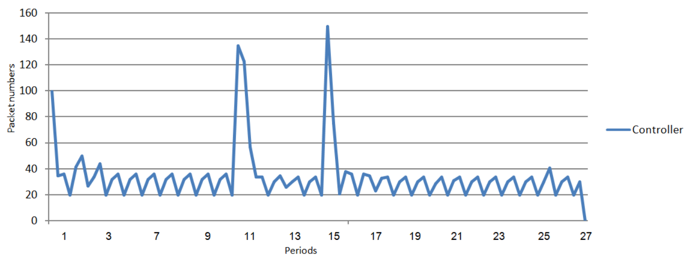

Figure 5. Network failure experiment: Number of packets sent by the controller.

Incident: Now a complete failure occurs in the network, implying that there is no network path leading to server 1. In Figure 3 from the 19th period to the 25th, the server 1 is not reachable anymore. Therefore, the communication between client 1 and server 1 must be redirected toward the second server.

Response: The context is now set as critical_network_status. The OrBAC engine activates and deactivates different rules. Let's take the example of the following rule:

- O_2: `Obligation(CSP, SDN-Controller, redirect, loadBalancing-on-demand, critical_network_status, system-timeout)`

The OrBAC-PDP will ask the controller to deploy new openflow rules to the switches, thus enabling a new path for client 1 with a new server S2.

We observe, in Figure 4, the second server taking over the first one in processing the requests from client 1. The latency of the two clients are now similar, at around 500 μs.

VII. CONCLUSION

In this paper, we have designed a flexible and generic framework that will manage and define dynamically the network security policies and push/pull them to the SDN equipment. This solution permits to respond automatically to the detected alerts and problems. It offers also to an administrator or a developer to specify modify rules or to configure a service without a deep knowledge of the openflow standard.

We illustrated our work with a real SDN testbed and ran different experiments with an availability security policy as a case study. The results assure the feasibility and the importance of our solution that may enhance the performance of the application.

Future work aims at completing the implementation with the appropriate monitoring tools to assist network administrators in tasks such as alert data extraction and verification of the VM status. We also plan to evaluate our solution in more complex network scenarios, including different network operators.

ACKNOWLEDGMENT

This research has been supported in part by the European Union's Horizon 2020 Research and Innovation Programme under grant agreement No. 643964 (SUPERCLOUD), and by the French national Fund for Digital Society (FSN). The opinions expressed in this paper are those of the authors and do not necessarily reflect the views of the Ministry of Internal Affairs and Communications, Japan, or of the European Commission.

REFERENCES

[1] Sandhu, R. S., Coyne, E. J., Feinstein, H. L., Youman, C. E. (1996). Role-based access control models. Computer, (2), 38-47.

[2] Kalam, A. A. E., Baida, R. E., Balbiani, P., Benferhat, S., Cuppens, F., Deswarte, Y., Trouessin, G. (2003, June). Organization based access control. In the International Workshop on Policies for Distributed Systems and Networks (POLICY 2003), IEEE .

[3] Elrakaiby, Y., Cuppens, F., Cuppens-Boulahia, N. (2012). Formal enforcement and management of obligation policies. Data and Knowledge Engineering, 71(1), 127-147.

[4] Kreutz, D., Ramos, F. M., Esteves Verissimo, P., Esteve Rothenberg, C., Azodolmolky, S., Uhlig, S. (2015). Software-defined networking: A comprehensive survey. Proceedings of the IEEE, 103(1), 14-76.

[5] De Cicco, L., Caldaralo, V., Palmisano, V., Mascolo, S. (2014, December). TAPAS: a Tool for rApid Prototyping of Adaptive Streaming algorithms. In Proceedings of the 2014 Workshop on Design, Quality and Deployment of Adaptive Video Streaming (pp. 1-6). ACM.

[6] Cropper, J., Ullrich, J., Fruhwirt, P., Weippl, E. (2015, August). The Role and Security of Firewalls in IaaS Cloud Computing. In the International Conference on Availability, Reliability and Security (ARES), IEEE.

[7] Richard, S. (2015) Effective Network Orchestration Starts by Automating Provisioning. Gartner's report.

[8] Mann, V., Kannan, K., Vishnoi, A., Iyer, A. S. (2013, May). Ncp: Service replication in data centers through software defined networking. In the International Symposium on Integrated Network Management (IM 2013). IEEE.

[9] Foster, N., Harrison, R., Freedman, M.J., Monsanto, C., Rexford, J., Story, A., Walker, D. (2011). Frenetic: A network programming language. In ACM SIGPLAN Notices, 46(9), 279–291, ACM.

[10] Monsanto, C., Reich, J., Foster, N., Rexford, J., Walker, D. (2013). Composing software defined networks. In Proceedings of the tenth USENIX Symposium on Networked Systems Design and Implementation (NSDI 13) (pp. 1-13). USENIX.

[11] Voellmy, A., Kim, H., Feamster, N. (2012, August). Procera: a language for high-level reactive network control. In Proceedings of the first workshop on Hot topics in software defined networks (pp. 43-48). ACM.

[12] Voellmy, A., Wang, J., Yang, Y. R., Ford, B., Hudak, P. (2013, August). Maple: simplifying SDN programming using algorithmic policies. In ACM SIGCOMM Computer Communication Review (Vol. 43, No. 4, pp. 87-98). ACM.

[13] Xu, G., Dai, B., Huang, B., Yang, J. (2015, August). Bandwidth-Aware Energy Efficient Routing with SDN in Data Center Networks. In 17th International Conference High Performance Computing and Communications (HPCC), 2015 IEEE .

[14] Kang, N., Liu, Z., Rexford, J., Walker, D. (2013, December). Optimizing the one big switch abstraction in software-defined networks. In Proceedings of the ninth ACM conference on Emerging networking experiments and technologies (pp. 13-24). ACM.

[15] Shin, S., Porras, P. A., Yegneswaran, V., Fong, M. W., Gu, G., Tyson, M. (2013, February). FRESCO: Modular Composable Security Services for Software-Defined Networks. In Proceedigns of the ISOC Network and Distributed System Security Symposium (NDSS). ISOC.

[16] Qazi, Z. A., Tu, C. C., Chiang, L., Miao, R., Sekar, V., Yu, M. (2013, August). SIMPLE-fying middlebox policy enforcement using SDN. In ACM SIGCOMM computer communication review (Vol. 43, No. 4, pp. 27-38). ACM.

[17] Anwer, B., Benson, T., Feamster, N., Levin, D., Rexford, J. (2013, August). A slick control plane for network middleboxes. In Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking (pp. 147-148). ACM.

[18] Yan Q., Yu F. R., Gong Q. and Li J., ”Software-Defined Networking (SDN) and Distributed Denial of Service (DDoS) Attacks in Cloud Computing Environments: A Survey, Some Research Issues, and Challenges,” in IEEE Communications Surveys and Tutorials, vol. 18, no. 1, pp. 602-622 2016.