iMPROVE: Enhancing the Introduction of Services on Programmable Virtual Networks

Ricardo Luis dos Santos, Muriel Figueredo Franco, Eder John Scheid,
Ricardo José Pfitscher, Lisandro Zambenedetti Granville, Liane M. Rockenbach Tarouco
Institute of Informatics – Federal University of Rio Grande do Sul, Brazil
{rlsantos, mffranco, ejscheid, rjpfitscher, granville}@inf.ufrgs.br, liane@penta.ufrgs.br

***Abstract*—Programmable Virtual Networks (PVNs) make the network more flexible and allow the fast introduction of new services. However, several shortcomings hamper their wider adoption, including: (*i*) the extensive knowledge required to configure and manage the NetApps; (*ii*) the lack of descriptors to detail all nuances of the NetApps; and (*iii*) there is no solution that enables to distribute and configure NetApps over distinct technologies. Therefore, we propose `iMPROVE` to simplify the introduction of services in PVNs, enhancing the distribution of NetApps. We also extend the ETSI network service descriptor to support distinct technologies as well as to represent conflict issues. We demonstrate evidence of `iMPROVE`'s feasibility in a case study and compare it with the main solutions for distributing and deploying applications over multiple platforms.**

***Index Terms*—Network Programmability; Network Virtualization; Programmable Virtual Networks; Future Internet.**

I. Introduction

Computer networks are built up using dedicated devices based on purpose-specific hardware [1]. These devices (a.k.a. middleboxes) generally run closed source and proprietary software that carry out a wide range of network services, such as traffic filtering, intrusion detection, and routing. Both middleboxes and combined software have undergone years of development and testing to achieve stability and a reliable performance. As a result, there has been an increase in CAPital EXpenditure (CAPEX) and OPerational EXpenditure (OPEX) which restricts the networking market to large-scale companies. Besides, even straightforward and necessary changes (*e.g.*, IPv6 and IPSec) depend on the interests of these companies and on the replacement of middleboxes in the whole network, which set up a huge barrier to innovation [2] [3].

Network virtualization has allowed the migration of network services from middleboxes to virtual machines hosted on Commercial Off-The-Shelf (COTS) servers. Also, programmability technologies such as ETSI's NFV [2], Juniper's Junos SDK [4], Cisco's ONE [5], OpenFlow [6], and IETF's ForCES [7] enable administrators to develop and deploy their programs into the network. The introduction of both programmability and virtualization leads to Programmable Virtual Networks (PVNs). PVNs make the network more flexible and allow rapidly to introduce new services by opening up the networking market to small companies and third-party developers while increasing network innovation [8]. Also, PVNs enhance the management of computational resources, reducing the CAPEX and OPEX by using COTS servers to host network services [9].

PVNs can replace middleboxes and closed software by Virtual Network Appliances (VNAs). VNAs are virtual machines containing an Execution Environment (EE), which is able to run network applications (a.k.a. NetApps), programmed by administrators or third-party developers. A single VNA hosts from just one NetApp (*e.g.*, Firewall or Load-Balancer) to a whole framework containing several NetApps (*e.g.*, a security framework with: Firewall, Encryption, Intrusion Protection System, and Deep Packet Inspection). However, a network built up solely with VNAs is unfeasible. The performance limitations of COTS servers and the overhead introduced by the virtualization have prevented high-speed network processing because they decrease throughput and increase latency [10]. Some NetApps (*e.g.*, switching and error control) must stay hosted on middleboxes to fulfill their performance constraints. Thus, VNAs and middleboxes can coexist in the same network, providing performance and flexibility [11].

Even with the benefits of PVNs, several shortcomings hamper their wider adoption. First, integrating VNAs with real-world production networks is a hard task, mainly because of the lack of documentation and the extensive knowledge required to deploy NetApps [8]. Second, the absence of repositories and the unavailability of NetApps for download restrict the distribution of developed services to just a few PVNs owners [12]. Third, to add new services, administrators must now address dependency and conflict issues among VNAs and also among NetApps (inside VNAs) [13]. Finally, to the best of our knowledge, there is no descriptor to detail VNAs or NetApps of distinct programmability-related technologies, and thus preventing the automation of some tasks (*e.g.*, distribution, configuration, and conflict detection).

In this paper, we propose the `iMPROVE` to simplify the introduction of services in PVNs, enhancing the distribution of NetApps and VNAs. We designed a framework for repositories that allows to store, publish, and distribute network services. We also introduce a marketplace for distinct technologies, such as NFV, OpenFlow, and Cisco ONE. This marketplace enables PVNs owners to easily select, download, and deploy new network services. In addition, we extended the ETSI Network Service Descriptor (NSD) [13] to support NetApps and VNAs of several programmability-related technologies, as well as to describe conflict issues. Finally, we conduct a case study to provide the evidence of `iMPROVE`'s feasibility.

The remainder of this paper is organized as follows. In

1550-445X/17 $31.00 © 2017 IEEE
DOI 10.1109/AINA.2017.113

Section II, we detail the iMPROVE conceptual architecture and describe our extension for the ETSI NSD. In Section III, we conduct a case study to prove the concept of the iMPROVE. Next, in Section IV, we discuss and compare iMPROVE with the main research efforts to simplify the introduction of services and to distribute applications. Finally, in Section V, we present final remarks and future work.

II. iMPROVE Architecture

iMPROVE simplifies the introduction of new services in PVNs based on different technologies. It involves setting up a platform that allows developers to send their programs to an online repository. This platform also enables PVN owners to deploy compatible programs. For this, iMPROVE carries out the necessary actions to select the correct packages or images as well as to check dependency and conflict issues. Besides, iMPROVE triggers commands to install and configure the NetApp package or instantiate the VNA image via a Generic Deployment Platform (GDP) (*e.g.*, App2net [8] or OpenStack [14]).

In summary, iMPROVE performs several actions in the services lifecycle, which include: (*i*) to publish and store NetApps packages, VNAs images, and the NetApp Descriptor (NAD) file; (*ii*) to enable PVN owners to select and download NetApps and VNAs; (*iii*) to verify conflict issues; (*iv*) to check dependencies; and (*v*) to send commands to the GDP. iMPROVE retrieves information about NetApps from the iMPROVE NetApp Catalog, so that, it can determine the compatibility of NetApps. It also matches the NetApps requirements with those of the PVN (*e.g.*, supported programmability technologies and used EEs), by finding out and displaying the compatible NetApps there are available in the Marketplace. Finally, the PVN owner can select which NetApps will be installed.

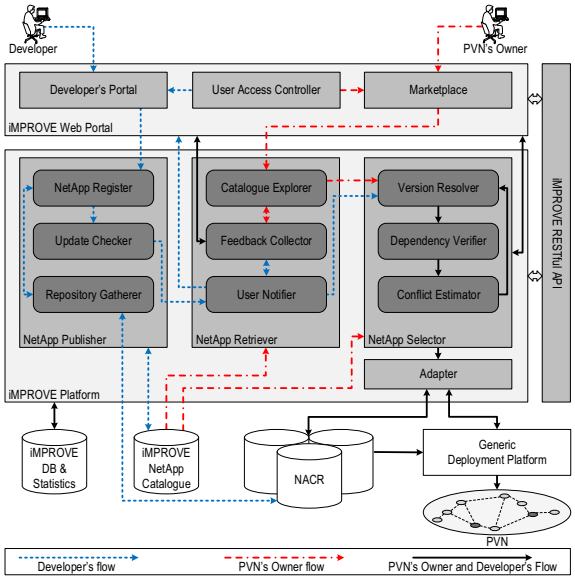

Fig. 1: The conceptual architecture of iMPROVE.

Two of the main actors interact with iMPROVE: developers and PVN owners. Developers can make NetApps and their initial functional settings available, as well as the VNAs. Whereas, PVN owners can select, download, and deploy a NetApp or VNA, which provide a network service (*e.g.*, a Deep Packet Inspection and video encoding/transcoding). In Figure 1 are depicted the main components of iMPROVE: NetApp Publisher, NetApp Retriever, and NetApp Selector.

A. The NetApp Publisher

Let us suppose that a developer wants to release a new network service. He/She must first send the NetApp package, VNA image, and the NetApp Descriptor (NAD) file through the iMPROVE Web Portal (see Subsection II-D). Later, the NetApp Publisher registers it and makes the network service available to other users in the Marketplace, as a means of allowing PVN owners to deploy it in their networks. The NetApp Publisher is composed of three modules to ensure the modularity of the platform: NetApp Register, Update Checker, and Repository Gatherer.

The NetApp Register module imports data from the NAD file. After this, this module checks whether the network service has been registered, (*i.e.*, if there is a record containing the vendor and identification). If there is no record, the Register stores the initial data about the network service lifecycle in the DB & Statistics, such as categories, responsible company or developers, last update, and descriptive features.

When developers update an existing network service, the Update Checker module checks which PVNs are affected. This module obtains the preferences of the owners from iMPROVE DB & Statistics, and checks which NetApps or VNAs need to be updated. Some NetApps/VNAs cannot be automatically updated because of critical issues such as conflicts and unsuitable virtual hardware requirements. In these cases, the Checker sends a notification about these problems to the User Notifier for human intervention.

Developers or companies may wish to store NetApps and VNAs in their own repositories. In this case, a developer registers its repository rather than NetApps or VNAs, and provides data such as IP address, access credentials, and asymmetric keys. Thus, the Repository Gatherer imports all the available NAD files from the External Online Repository, and sends these files to the NetApp Register for registration. In this case, the NetApp packages or VNA images must remain in the external repository. In addition, the Repository Gatherer must periodically check that NAD files including or updating their records; for this, the developer can configure the time interval to perform this verification process.

B. The NetApp Retriever

Both developers and PVN owners interact with the NetApp Retriever component, which carries out a number of actions at various stages of the process. The main function of the NetApp Retriever is to assist the users in finding the required services provided by NetApps or VNAs. Furthermore, the Retriever manages the communication between the actors and the platform as well as collects the feedback from the users. This component is formed of three modules: Catalog Explorer, Feedback Collector, and User Notifier.

When choosing a network service, the PVN owner interacts with the Catalog Explorer via the Marketplace. First, the Explorer retrieves the data about the PVN such as supported technologies, installed NetApps, VNA instances, and virtual hardware specifications. The iMPROVE Catalog provides data about the available network services and then, the Explorer conducts a matching operation with this data, which only shows the services that are compatible. The owner can specify the categories and keywords to refine the obtained results, and thus enhance the search process. After a network service has been chosen, this module triggers the NetApp Selector to choose the correct NetApp package or VNA image and checks if there are any possible conflicts and dependencies.

The Feedback Collector obtains the users' evaluations for each network service and stores them in DB & Statistics. These evaluations contain a textual review and a rating score. The Collector module enables users to report bugs, errors, and unknown conflicts. Also, this module can notify the responsible developers about any problems that must be addressed. Likewise, when a developer submits a patch to correct a bug, the Collector informs the interested PVN owners (who then report it) about the update that has been received.

Essentially, the User Notifier sends messages to both actors: developers and PVN owners. When an update has been registered, the Notifier sends a message containing the data about the update, which can occur automatically. However, if an error occurs during the update process or if the PVN owner wishes to authorize it, the Notifier requests the PVN owner to make an intervention. Note that if an error occurs and the owner does not address the problem, this module will periodically require an intervention, depending on the owner's preferences. If there is no problem, the process continues as normal. Moreover, the Notifier informs the developers when a PVN owner reports a new bug or if there is a conflict in their network services.

C. The NetApp Selector

This component is responsible for selecting the correct NetApp package or VNA image, by taking into account the version, virtual hardware requirements, supported technologies, and used network services. It should be noted that the Selector only determines which version of the NetApp package will be used, based on issues of dependency and conflict. We have divided the Selector into three modules: Version Resolver, Dependency Verifier, and Conflict Estimator.

Both developers' and owners' flows arrive at the Version Resolver (Figure 1). This module requests NetApp Catalog information about all the versions of the desired network service. On the basis of the current PVN data, the Resolver identifies the last compatible version; this means, the last version of the NetApp package or VNA instance that is supported by the PVN in accordance with both technologies and virtual hardware requirements. Although the Version Resolver selects a compatible version, conflicts or dependencies may occur with installed NetApps and, in this case, the Resolver selects another NetApp version.

Once a NetApp or VNA version has been selected, the Dependency Verifier will check the dependency tree. The Verifier performs this by obtaining the NetApp dependencies, which are informed in the NAD file by developers and stored in the iMPROVE NetApp Catalog. Afterward, this module finds out if the NetApps have been installed in the PVN and determines which dependencies are needed. In this way, for each dependent NetApp, the Verifier determines the nested dependencies forming the NetApps list. If a dependent NetApp is already installed, it is not included in the list.

The Conflict Estimator module conducts the conflict analysis. This module obtains the necessary data, such as the installed NetApps and the dependency tree. The Estimator then compiles the conflicts of all used NetApps in a known conflicts list. Next, the Estimator establishes if there is any dependent or installed NetApp in the known conflicts list. Whenever the Estimator finds a conflict, the selection process restarts and returns to the Version Resolver that ignores the used version. If none of the available versions fulfill the requirements by the end of the process, the PVN owner is requested to make an intervention. Also, this module registers a new bug for the developers with the employed data. Note that the developers and PVN owners can employ several mechanisms to identify the conflicts and find other means to migate known conflicts, but, these mechanisms are outside the scope of this paper.

D. NetApp Descriptor

The ETSI specifies the NSD file [13] for the NFV technology, which provides details of the network service in a simple XML notation, including the vendor identification, version, dependencies, and virtual hardware requirements. We extended the ETSI NSD to support all the features of the `iMPROVE`. Thus, our extension, called NetApp Descriptor (NAD), introduces some new tags to support distinct programmability technologies (*e.g.*, NFV, SDN, and P4) and it allows developers to describe specific features of NetApps and VNAs (*e.g.*, conflicts and different supported versions).

TABLE I: Extended Tags for the NetApp Descriptor.

Tag	Parent Tag	Description
category	categories	Adds a category (a.k.a. keyword) related to the network service. This enables the similar network services to be grouped together as well as helping the end-users to find a correct service.
technology	packages	The main tag to specify information about the technologies that are supported by the network service.
identifier	technology	The string that identifies a programmability technology, for instance, openflow, nfv, and cisco-onepk.
version	technology	A supported version of the programmability technology.
location	technology	The main tag to describe the URIs of the NetApp packages or VNA images.
ee	technology	The execution environment necessary to run the NetApp.
type	technology	Describes the type of package in detail. Two values are supported "netapp" and "vna".
hash	technology	Informs the MD5 hash about the package. This hash is used in the validation process.
conflicts	technology	Map all known conflicts among the NetApps and VNAs; this means, this NetApp or VNA operates improperly when the NetApp/VNA target is used.
initial_config	technology	Describes the initial functional settings of a NetApp, which might be a simple command or a set of commands.
manage_action	technology	Main tag to specify the commands and parameters that are needed to manage a NetApp (*e.g.*, install, start, restart, pause, resume, uninstall, and configure). Note that only the supported actions are described.

Our extension includes two tags under the NSD: *categories* and *packages*. The *categories* tag allows us to add keywords related to the network service. This tag enables us to group similar network services as well as helps end-users to find a desired service. The *packages* tag describes all the data about the packages including conflicts, supported technologies, initial settings, available management actions, and necessary EE. In Table I is described the proposed tags of our extension.

E. Other iMPROVE Elements

iMPROVE requires several important elements to support its processes, including Databases and Repositories, the Adapter, the iMPROVE Web Portal, and iMPROVE RESTful API. All these elements enable the iMPROVE platform to: (*i*) interact with developers and PVN owners; (*ii*) store relevant data about PVNs, NetApps, users, and the relationships among them; and (*iii*) provide an open API that allows users to expand and automate several actions. The main features of each element are described in detail below.

1) Databases and Repositories: All the necessary data are stored in three distinct elements: DB & Statistics, NetApp Catalog, and NetApp Code Repository (NACR). In fact, these elements can be implemented in one database; but, we have decided to split them up to allow developers and companies to host the NetApp packages and VNA images in their repositories, and thus enhance the NetApp/VNA distribution. The iMPROVE DB & Statistics stores all the system data and NetApp statistics, which includes the following: users feedback, updates, description of services, bugs reported, user credentials, data of the repository, and asymmetric keys.

iMPROVE platform saves the content of the NAD files in the NetApp Catalog, including categories, dependencies, conflicts, requirements, supported technologies, initial settings, and required commands. The platform stores all the NetApp packages and VNA images in the NACR. As mentioned above, there may be multiple External Online Repositories, which are managed by third-party developers or companies. These repositories must contain NAD files so that iMPROVE can import them, and register the related NetApps and VNAs.

2) Adapter: This component performs the communication between iMPROVE and the GDP that will deploy/install necessary network services. In the first step, the Adapter validates the repositories' authenticity through the asymmetric keys. Next, this component checks the integrity of all used packages, by comparing the calculated hash with those stored in Catalog. Also, the Adapter can send the hash data to the GDP; thus, the GDP can check the integrity of the NetApps after downloading them. When the deployment process has ended, the Adapter requests the data about the deployment from the GDP and stores it in the iMPROVE DB. Note that iMPROVE uses the GDP via the Adapter in a transparent way. In other words, the GDP could be unaware about iMPROVE.

When installing a new network service, the Adapter sends the NetApps list (including the dependent packages) to the GDP and supplies information about the nodes. Then, the GDP carries out the necessary actions to install these NetApps. However, if there is no communication between the repositories and the GDP (*e.g.*, due to unauthorized access or connectivity problems), the Adapter must retrieve all the necessary files from the repositories (*i.e.*, the NetApp and its dependent packages) and send them to the GDP, which deploys them in the PVN nodes.

3) The iMPROVE Web Portal: This component makes all the interactions between the platform and the system users, and thus allows them to register elements, record PVN data, and trigger actions in iMPROVE. The Web Portal is divided into three modules: (*i*) the Developer's Portal that provides interfaces and forms, which allow the developers to register NetApps, VNAs, and external repositories; (*ii*) the User Access Controller that authenticates and grants permission to users, by limiting access and avoiding improper handling of PVNs and services; and (*iii*) the Marketplace, a set of Web interfaces that enable the users to find and choose the NetApps. By using the Marketplace, a user can also install the required packages.

4) iMPROVE RESTful API: The core functions of iMPROVE are exposed through a RESTful API, which can be extended by interested third-party developers. Due to its modular architecture, an interested developer can replace an existing element (*e.g.*, Web Portal) by a new one. The new element must be able to perform at least the same actions as the replaced element. Furthermore, new elements that contain additional features can be plugged into iMPROVE (*e.g.*, a visualization module [15]). In Table II, there is a detailed account of the exposed methods.

III. Case Study

We conducted a case study to provide a detailed description of the interactions and the necessary actions for a developer when publishing a network service in the iMPROVE. Following this, we explored the interactions of a PVN owner to purchase and deploy this network service. Finally, there is a discussion about the management of this new network service via the iMPROVE RESTful API. The main elements and interations are presented in the Figure 2.

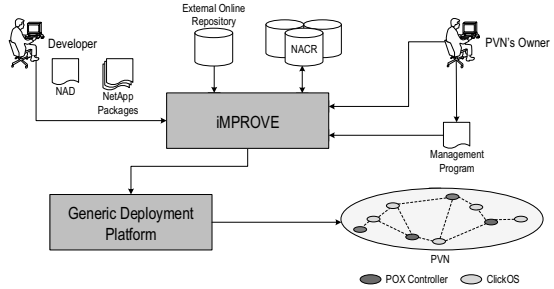

Fig. 2: Interactions between iMPROVE and external elements.

Let us suppose that an arbitrary developer programs a new network service, which provides a *trust-based multicasting*. This network service supports two different technologies: NFV and OpenFlow. However, there are three distinct packages, each one coded for a specific programmability technology: (*i*) one for general purpose NFV; (*ii*) one for NFV with clickOS; and (*iii*) one for OpenFlow using a POX controller.

TABLE II: Exposed methods by the iMPROVE RESTful API.

Method	Parameters	Operation	Output
publish	NAD file; URI to packages/images	Triggers the NetApp or VNA registration process	Success or failure
unpublish	NAD file	Removes a NetApp or VNA from the marketplace	Success or failure
retrieveService	Parameters; PVN identifier	Returns a network service list based on the parameters	List of network services
getDependencies	NetApp or VNA identifier	Generates the dependency tree	List of known dependencies
getConflicts	NetApp or VNA identifier; PVN identifier	Produces a conflict list in accordance with the installed NetApps and dependency tree	List of all known conflicts
notifyUsers	Message; NetApp or VNA identifier	Sends notifications to subscribed users via the Web Portal	Success or failure
selectPackage	Network service identifier; PVN	Selects the correct version of the package by taking account of conflicts, dependencies, and supported technologies	URI to packages/images; list of dependent packages
getStatisticsAndInfo	Network service identifier	Returns data about the network service, including the vendor, the number of downloads, related packages, supported technologies, known conflicts, and dependencies	Dictionary with the related information
checkUpdates	PVN or Network service identifier	Generates a list of all pending updates	List of pending updates
retrieveInstalledPackages	PVN	Returns a list of all installed NetApps and VNAs containing identifiers and technologies	List of all installed NetApps and VNAs
retrieveActions	NetApp or VNA identifier	Generates a list of all supported actions	List of available actions
runAction	PVN; NetApp or VNA identifier; Action; Parameters	Execute a specific action of a NetApp or VNA over a PVN	Success or failure

As well as the packages, the developer must create a NAD file that contains all the data about the NetApp. Although there are three packages (one per technology), only one NAD file is required to describe all packages that appear under the technology tag. Figure 3 shows the core tags of the NAD file, especially those related to our proposed extension.

In the case of NFV, two distinct packages are provided. The first package is a VNA image that contains the whole framework to support the service (lines 9 to 17). This image is a virtual machine containing the NetApp package and the necessary EE. The developer does not send this image because it is available in an External Online Repository (Figure 2 and lines 14 to 16). The second package for the NFV is a NetApp coded for the ClickOS EE (lines 18 to 39) that will be hosted in the NACR. Thus, this package can be installed in an existing node with support for NFV and ClickOS. In addition, the developer defined an initial settings (lines 24 to 26), which denotes that all the nodes can be regarded as "trusted", as well as two different management actions (lines 30 to 38). Note that the expression `$value0$` (line 35) will be replaced by the information provided by the user (trusted-nodes). Likewise, the `initial_config` (line 25) is a partial command, which is used together with the install action. In this action, the expression `$initial_config$` (line 31) will be replaced by the initial settings.

In the case of the POX (lines 40 to 81), the NetApp package also will be hosted in the NACR. Again, the developer defined an initial settings (lines 46 to 48). However, the PVN owner can change these settings during the installation process. In addition to the management actions of the NFV NetApp (lines 30 to 38), the POX version has four additional actions: (*i*) *start*, initiating the execution of the NetApp according with the performed settings (line 51); (*ii*) *stop*, shutting down the execution of the NetApp and saving the used settings (line 52); (*iii*) the configuration *force-add-node* allows to add a node as "trusted" without performing the verification process (lines 58 to 61); and (*iv*) the configuration *force-remove-node* allows to remove a node as "trusted" without performing the verification process (lines 62 to 65). Finally, there are two known conflicts in the POX version, which include all the versions of *multicast-ufrgs* (lines 72 to 75) and previous versions of the *dpi-telefonica* 4.0.2 (lines 76 to 79). In other words, this *trust-based multicasting* operates improperly when *multicast-ufrgs* or *dpi-telefonica* ($<$ 4.0.2) are used. Note that in XML notation, the "$<$" is represented by "<". Moreover, the expression `$value0$` (lines 56, 60, and 64) will also be replaced by the values informed for each action.

When publishing the NetApp for download by the PVN owners, the developer must register it in `iMPROVE`. First, the developer must log in, fill in the registration form, and submit the two NetApp packages (one to OpenFlow using POX and one to NFV using ClickOS) and the NAD file. After this, the Portal sends these files to the NetApp Register, which reads the NAD file and stores the data in the NetApp Catalog. Then, the Register creates a new record in the iMPROVE DB with the data about the service obtained from the registration form and NAD file. In the next step, the Publisher component uploads the NetApp packages to the local NACR. At this time, the NetApp is published and available for installation.

Assuming that the sent packages update an existing NetApp, the Update Checker finds out if any PVNs are using these NetApps. The Checker arranges these PVNs into two groups: (*i*) the PVNs that can be updated automatically; and (*ii*) the PVNs that require human intervention. While the PVNs in the first group trigger the update processes of the NetApp, the PVNs in the second group wait for human actions. Thus, the User Notifier can either inform the owners (first group) or request a decision from the owners (second group) about the update process. Before installing this update, NetApp Selector carries out a check of the dependencies (Dependency Verifier) and conflicts (Conflict Estimator). These processes determine all the new dependencies and find out if any conflict is likely to occur among the installed services and new NetApps.

If there is no problem, `iMPROVE` carries out the necessary actions in the GDP to install the NetApp, such as sending commands and triggerring actions via Adapter. However, if there is any problem, `iMPROVE` requests the owner to perform an intervention. This intervention can involve other actions, for example, the replacement of NetApps that are causing unresolved conflicts. After the PVN owner has tackled the problems, `iMPROVE` can request the NetApps installation for the GDP based on the owner's definitions. Even though the

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 ...
3 <categories>
4   <category>security</category>
5   <category>multicasting</category>
6   <category>communication</category>
7 </categories>
8 <packages>
9   <technology>
10     <identifier>nfv</identifier>
11     <version>all</version>
12     <hash>3546705163bc40eef80ddf2cd11918aa</hash>
13     <type>vna</type>
14     <location>
15       <uri>http://myrepository.com/services/nfv/trust_multicast_45.img</uri>
16     </location>
17   </technology>
18   <technology>
19     <identifier>nfv</identifier>
20     <version>all</version>
21     <ee>clickos</ee>
22     <type>netapp</type>
23     <hash>e48f694fc62f4512317eda7e40ca2dac0447da10</hash>
24     <initial_config>
25       <command>--trusted-nodes=all</command>
26     </initial_config>
27     <location>
28       <uri>file:///repository/services/nfv/clickos/trust_multicast_45.click</uri>
29     </location>
30     <manage_action>
31       <install>click-install $initial_config$ trust_multicast_45.click</install>
32       <configure>
33         <parameter>
34           <value0>trusted-nodes</value0>
35           <command>click-install --trusted-nodes=$value0$ trust_multicast_45.click</command>
36         </parameter>
37       </configure>
38     </manage_action>
39   </technology>
40   <technology>
41     <identifier>openflow</identifier>
42     <version>1.0</version>
43     <ee>pox</ee>
44     <type>netapp</type>
45     <hash>f8af16331688337725ded1835ffdcec44</hash>
46     <initial_config>
47       <command>./trust_multicast_45.py --config={"trusted-nodes": "all"}</command>
48     </initial_config>
49     <manage_action>
50       <install>./trust_multicast_45.py --install --file=${POX_HOME}/startup.py</install>
51       <start>./trust_multicast_45.py --start</start>
52       <stop>./trust_multicast_45.py --terminate</stop>
53       <configure>
54         <parameter>
55           <value0>trusted-nodes</value0>
56           <command>./trust_multicast_45.py --trusted-nodes=$value0$</command>
57         </parameter>
58         <parameter>
59           <value0>force-add-node</value0>
60           <command>./trust_multicast_45.py --add-node=$value0$</command>
61         </parameter>
62         <parameter>
63           <value0>force-remove-node</value0>
64           <command>./trust_multicast_45.py --remove-node=$value0$</command>
65         </parameter>
66       </configure>
67     </manage_action>
68     <location>
69       <uri>file:///repository/services/openflow/pox/trust_multicast_45.py</uri>
70     </location>
71     <conflicts>
72       <conflict>
73         <identifier>multicast-ufrgs</identifier>
74         <version>all</version>
75       </conflict>
76       <conflict>
77         <identifier>dpi-telefonica</identifier>
78         <version> &lt; 4.0.2</version>
79       </conflict>
80     </conflicts>
81   </technology>
82 </packages>
83 ...
```

Fig. 3: Partial NAD file for *trust-based multicasting*.

PVN owners may not have addressed the problems or have decided not to make an update, the new network service is still available and can be installed in compatible PVNs.

Now, let us suppose the case of a PVN owner that wants to install the *trust-based multicasting* service in his/her PVN. The owner uses the iMPROVE to select and install a NetApp that provides this service. For this, the owner provides data about the PVN, such as supported technologies, EEs, nodes, and IP addresses. As can be observed in Figure 2, there are two distinct technologies in the PVN, namely NFV using ClickOS EE and OpenFlow using POX EE. In addition, Table III presents the installed network services in the PVN. Most services are installed in a single technology, but, some of them (*e.g.*, *cache-squid*) are installed in both technologies.

The Catalog Explorer retrieves the registered PVN data. In the Marketplace, the owner selects two different categories: security and multicasting. He/She also informs two keywords: trust-based and communication. Following this, Explorer uses the categories and keywords to filter the NetApps list. Therefore, the Marketplace only shows compatible NetApps (which use OpenFlow with POX or NFV with ClickOs) where the categories and keywords match the stored NAD files. Finally, the owner selects the network service *trust-based multicasting* and triggers the selection process.

TABLE III: Network Services Installed in the PVN.

Network Service	Version	Execution Environment
cache-squid	3.1	ClickOS, POX Controller
firewall-netfilter	10.0.1	ClickOS
ids-telofonica	1.9.8	ClickOS
nat-cisco	1.0	ClickOS
dpi-telefonica	4.0	POX Controller
load-balancer-ufrgs	0.98	POX Controller

In the selection process, the Version Resolver retrieves the last version of the NetApp and carries out checks of the dependencies (Dependency Verifier) and conflicts (Conflict Estimator). In this case, the Estimator finds a conflict with an installed NetApp (*dpi-telefonica* 4.0). Then, the process returns to the Version Selector to retrieve the previous version of the NetApp. As there is no previous version, the PVN owner is notified about the detected conflict. On the one hand, the PVN owner could remove the *dpi-telefonica*, and thus restarting the installation process. On the other hand, the PVN owner could upgrade the *dpi-telefonica*, which also migates the conflict because the last version of the *dpi-telefonica* (4.3) is compatible with the *trust-based multicasting* (there is no known conflict with this version mapped in the NAD file).

To upgrade the *dpi-telefonica*, the owner just triggers the upgrade process in the Portal. In that way, the NetApp Selector will retrieve the last version of the *dpi-telefonica* (4.3). Then, the Adapter performs the necessary actions to upgrade this NetApp, and the PVN owner can continue with the *trust-based multicasting* installation. Now, without conflicts, iMPROVE interacts with the Repository and the GDP to install the chosen NetApp and its dependencies. After this, iMPROVE sends the initial settings to the GDP, which installs (lines 31 and 50) and configures (lines 25 and 47) the *trust-based multicasting* in the ClickOS for NFV and POX controller for OpenFlow. Thus, two distinct packages are installed (one per technology).

The owner can write a management program, and thus adapt the NetApp execution to the nuances of the environment. For example, due to performance and security constraints, the *trust-based multicasting* must be executed from 8:00 AM to 6:00 PM, and, at every hour the trusted nodes must be changed. For this, the owner obtains from the RESTful API (retrieveActions) all available actions in each NetApp package. Then, the management program triggers the necessary actions (*e.g.*, start, stop, and configure) via iMPROVE (runAction) according to the constraints coded by the PVN owner. For each action, iMPROVE can run distinct commands in the nodes depending on supported technology (POX or ClickOS).

IV. Related Work

Several solutions enable third-party and certified developers to publish their applications in different areas. The main solutions set out here are from large-scale companies and academia, and describe core features and key approaches. These solutions are compared with iMPROVE, and focus on aspects that benefit both the developers and end-users. In Table IV, there is a summary of the general information provided by these solutions, including: (*i*) the domain of applications that are supported; (*ii*) the need for user registration; (*iii*) the opening up to third-party developers; and (*iv*) the question of technology dependency, which represents hardware and software requirements for running the available applications.

In the context of mobile devices, two successful solutions stand out: Google Play [16] and Apple App Store [17]. These solutions opened up the market for mobile applications and enabled third-party developers to program and offer a broad range of new mobile applications directly to end-users. By means of these solutions, an end-user can purchase and download innovative mobile applications from the store. In a similar way, Package Managers (*e.g.*, yum and aptitude) are broadly used to obtain end-user applications and services from repositories in UNIX-based operating systems.

Several companies are investing in cloud computing to provide services hosted on virtual machines. Due to the number of customers and services available, two solutions are worth highlighting: Amazon Web Service (AWS) Marketplace [18] and OpenStack Catalog [14]. AWS Marketplace contains a collection of cloud computing services that comprise the on-demand computing platform offered by Amazon. This solution allocates and configures all the infrastructure that complies with the contracted service. Similarly, OpenStack Catalog is a collection of ready-to-use applications that customers can deploy within private or public OpenStack clouds.

Recent advances in networking technologies, such as SDN and NFV, have paved the new way for potential business opportunities and increased the amount of innovation in the network. Nevertheless, operators must have an extensive knowledge of device instructions and NetApps before they can deploy the network services. To fill this gap, Hewlett-Packard created the HPE SDN App Store [19] for SDN applications, which allows customers to deploy services, by aligning the network with their business needs. Likewise, Juniper Developer Network (JDN) [20] offers a community for developers interested in building NetApps on Juniper platforms. However, Juniper has a rigorous process to determine if a developer is a viable candidate to publish an application.

The scientific and academic communities are also involved in the distribution of NetApps which can be applied in emerging technologies. Among the advances made in the academic world, Pinheiro *et al.* [21] defined the RepoSDN to allow administrators to manage NetApps in an SDN environment. In the case of NFV-enabled networks, the T-NOVA project [12] proposed an integrated architecture designed to enable network operators to deploy VNFs for their customers.

In Table IV is described a comparison of the main solutions for application distribution based on two aspects: (*i*) the Application Descriptor; and (*ii*) Application Install and Instantiation. Regarding to the Application Descriptor, an attempt has been made to check how a developer gives a detailed account of the applications in each solution, including the file format, the description of the network service features, and the description of conflicts among the applications. In the case of the second aspect, we carried out an investigation into which solutions provide and configure infrastructure for users, who run their applications. A check was made to determine whether the solutions allow upgrades/updates after the installation or instantiation process. Likewise, the user notification process was compared during the application lifecycle.

Google Play, App Store, and Package Manager are widely used to obtain end-user applications. However, they do not support NetApps or programmability technologies. Other successful solutions for general purpose services (*e.g.*, AWS and OpenStack Catalog) focus on cloud services but fail to include the distribution of the NetApps into the PVNs. There are well-defined solutions for PVNs such as JDN, HPE SDN, RepoSDN, and T-NOVA. Nevertheless, all of them are technology-dependent. For instance, JDN applications can only be installed for users who use Juniper devices. Moreover, HPE SDN Apps require the deployment of an environment with the HP VAN SDN Controller and OpenFlow protocol.

In the T-NOVA and iMPROVE solutions, developers are able to describe network services, which allows the automation of several tasks, such as the deployment, configuration, and management of NetApps and VNAs. In particular, the NAD describes conflicts and dependencies in detail, which ensures the applications are more compatible and avoids recursive errors caused by known problems. Thus, the developers can describe several packages for different technologies, by means of a single NAD file. When the NAD file is added to the modular architecture, iMPROVE becomes the only solution that can support distinct programmability technologies. Finally, AWS and T-NOVA provide an infrastructure for customers that are running applications. Notwithstanding, these solutions do not allow developers or PVN owners to use their own infrastructures. With regard to application updates and user notifications, it should be noted that only iMPROVE contains these features in the network management and services domain.

V. Final Remarks and Future Work

PVNs enable network services to migrate from middleboxes and closed software to VNAs hosted on COTS servers, and thus, they introduce innovation, flexibility, and reduce costs. However, some shortcomings hamper the PVNs wider adoption, which includes: (*i*) the extensive knowledge required to configure and manage the NetApps; (*ii*) the lack of descriptors to detail all nuances of the NetApps; and (*iii*) there is no solution that enables to detect conflicts, distribute, and configure NetApps over distinct technologies.

In this paper, we have introduced iMPROVE to tackle the shortcomings mentioned above. Thus, iMPROVE empowers

TABLE IV: Comparison among solutions for applications distribution.

-	General Information				App Descriptor			App Install and Instantiation		
Solution	**Domain**	**Requires Registration**	**Third-Party Developers**	**Technology-Dependent**	**Main Characteristics**	**Details Conflicts**	**Describes NS**	**Infrastructure**	**Upgrades and Updates**	**User Notification**
Google Play/App Store	End-user applications	Yes	Yes	Android OS/iOS	XML with requeriments and dependencies	No	No	No	Manual	Yes
Package Manager	General purpose services and applications	No	Yes	UNIX based OS	XML with repositories and dependencies data	Yes	No	No	Manual	Yes
AWS Marketplace	General purpose services	Yes	Yes	Amazon Infrastructure	XML with requirements and app chunks data	No	No	Yes	Manual	Yes
OpenStack Catalog	General purpose services	Yes	Yes	OpenStack Cloud	YAMl manifest	No	No	No	Manual	No
JDN	Network services	Yes	Yes, but limited	Juniper Platforms	Not specified	No	No	No	Manual	No
HPE SDN	Network services	Yes	Yes	HP controller and OpenFlow	Not specified	No	No	No	Manual	No
RepoSDN	Network services	Yes	Yes	OpenFlow	Not specified	No	No	No	Manual	No
T-NOVA	Network services	Yes	Yes	NFV	YAML or XML file, containing ETSI NSD	No	Yes	Yes	Manual	No
iMPROVE	Network services	No	Yes	No	XML file with NAD extension	Yes	Yes	No	Manual and Automatic	Yes

PVN owners to choose and deploy network services in PVNs with distinct EEs, even though they neither know the specificities about the NetApps nor the devices. We also designed an architecture to a marketplace and code repositories that enables developers to store, publish, and distribute NetApps. Finally, we proposed the NAD, an extension of the ETSI NSD, to describe the nuances of the NetApps of distinct technologies.

A case study was conducted to provide a detailed description of the necessary actions for a third-party developer to publish and distribute a new network service. It was shown that a single NAD file can describe packages for several technologies, including the initial settings, management actions, and known conflicts. Also, we showed the steps for a PVN owner to deploy and manage a new network service. Moreover, `iMPROVE` was able to install one network service with several packages in a PVN supporting two different technologies, namely ClickOS for NFV and POX for OpenFlow. We compared `iMPROVE` with the main solutions currently offered, by focusing on aspects that benefit developers and PVN owners. As a result, `iMPROVE` demonstrated to be a viable alternative for distributing and describing network services in PVNs that support distinct programmability technologies.

As future work, we intend to: (*i*) extend the NAD for supporting new concepts such as affinities and anti-affinities among NetApps or VNAs, and thus enhancing the performance and preventing the interference in others services; (*ii*) consider the virtual hardware requirements and physical resources in the NetApp Selector; (*iii*) design and develop a driver that discovers all installed NetApps in a PVN; and (*iv*) conduct an user friendliness evaluation.

REFERENCES

[1] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn: An intellectual history of programmable networks," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 87–98, Apr. 2014.

[2] NFV ISG, "Network Functions Virtualisation(NFV): An Introduction, Benefits, Enablers, Challenges & Call for Action," ETSI, Tech. Rep., 2012, Available at: https://portal.etsi.org/nfv/nfv_white_paper.pdf. Accessed: December, 2015.

[3] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, February 2013.

[4] A. Clemm, R. Wolter, and J. Kelly, *Network-embedded management and applications understanding programmable networking infrastructure*, 1st ed. NY, USA: Springer, 2013.

[5] S. Kiran and G. Kinghorn, "Cisco Open Network Environment: Bring the Network Closer to Applications," 2015, Available at: http://www.cisco.com/c/en/us/products/collateral/switches/nexus-1000v-switch-vmware-vsphere/white_paper_c11-728045.pdf. Accessed: January, 2016.

[6] N. McKeown *et al.*, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008. [Online]. Available: http://doi.acm.org/10.1145/1355734.1355746

[7] R. Dantu, T. Anderson, and R. Gopal, "Forwarding and control element separation (forces) framework (rfc 3746)," United States, 2004.

[8] R. L. dos Santos, O. M. Caicedo Rendon, J. A. Wickboldt, and L. Z. Granville, "App2net A Platform to Transfer and Configure Applications on Programmable Virtual Networks," in *20th IEEE Symposium on Computers and Communication (ISCC 2015)*, Larnaca, Cyprus, Jul. 2015, pp. 335–342.

[9] N. Chowdhury and R. Boutaba, "Network virtualization: state of the art and research challenges," *Communications Magazine, IEEE*, vol. 47, no. 7, pp. 20–26, July 2009.

[10] T. Wood, K. Ramakrishnan, J. Hwang, G. Liu, and W. Zhang, "Toward a software-based network: integrating software defined networking and network function virtualization," *Network, IEEE*, vol. 29, no. 3, pp. 36–41, May 2015.

[11] S. Sezer, S. Scott-Hayward, P. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, "Are we ready for sdn? implementation challenges for software-defined networks," *Communications Magazine, IEEE*, vol. 51, no. 7, pp. 36–43, July 2013.

[12] A. Ramos *et al.*, "Open source software for building private and public clouds," 2014, Available at: http://www.t-nova.eu/wp-content/uploads/2016/03/TNOVA_D2.42_Specification_of_the_Network_Function_Framework_and_T-NOVA_Marketplace.pdf. Accessed: January, 2016.

[13] NFV ISG, "Network Functions Virtualisation (NFV): Management and Orchestration," ETSI, Tech. Rep., 2014, Available at: http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_nfv-man001v010101p.pdf. Accessed: February, 2016.

[14] OpenStack, "Open source software for building private and public clouds," 2016, Available at: http://www.openstack.org/. Accessed: January, 2016.

[15] M. F. Franco, R. L. d. Santos, A. Schaeffer-Filho, and L. Z. Granville, "Vision – interactive and selective visualization for management of nfv-enabled networks," in *2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA)*, March 2016, pp. 274–281.

[16] Google, "Google Play," 2016, Available at: https://play.google.com/store. Accessed: February, 2016.

[17] Apple Inc, "Apple App Store," 2016, Available at: https://itunes.apple.com. Accessed: February, 2016.

[18] Amazon Company, "Amazon Web Services," 2016, Available at: https://aws.amazon.com. Accessed: February, 2016.

[19] Hewlett Packard Enterprise, "Hewlett-Packard Enterprise SDN App Store," 2016, Available at: https://saas.hpe.com/marketplace/sdn. Accessed: February, 2016.

[20] Juniper Networks, "Juniper Developer Network," 2016, Available at: https://developer.juniper.net/. Accessed: February, 2016.

[21] R. S. Pinheiro, B. A. Pinheiro, R. P. Esteves, and A. J. G. Abelém, "Reposdn: An repository organization and coordination method of software defined networks applications," in *NOMS 2014*, 2014, pp. 1–4.