A NovaGenesis Proxy/Gateway/Controller for OpenFlow Software Defined Networks

Antonio M. Alberti, Victor H. de O. Fernandes, Marco A. F. Casaroli, Lúcio H. de Oliveira, Frederico M. Pedroso Júnior
NovaGenesis Project, ICT Lab, Instituto Nacional de Telecomunicações, INATEL
37540-000, Santa Rita do Sapucaí, Minas Gerais, Brazil
{alberti@inatel.br; victorh@gee.inatel.br; marco.casaroli@gmail.com; lucio.henrique@inatel.br; fredericom@gec.inatel.br}

Dhananjay Singh
Department of Electronics Engineering, Hankuk University of Foreign Studies, Yongin, South Korea
{dan.usn@ieee.org}

***Abstract*—Software-defined networking (SDN) is a promising approach to deal with complexity in new generation networks. The idea is to "extract simplicity" from what we have learned in the last decades while "mastering complexity" at designing and deploying network infrastructures. The idea is to decouple control and data planes. In this sense, OpenFlow is a protocol for remote control of switches' forwarding tables, replacing the traditional distributed network control model by a centralized one. An open problem in OpenFlow, and more generally on SDN, is how to integrate network control with services orchestration, i.e. to enable service frameworks to negotiate with network representatives in order to create service-aware networks. In this paper, we employ the design principles of a new architecture called NovaGenesis to implement a proxy/gateway/controller for OpenFlow networks. This service represents, interoperates, and controls a Python OpenFlow controller (POX) in order to expose its resources directly to NovaGenesis services. The POX Agent (POXA), as it is called, innovates on exposing OpenFlow resources to NovaGenesis name-oriented service orchestration, enabling the direct establishment of service level agreements among POX and NovaGenesis services.**

Keywords—Software-defined networking, OpenFlow, NovaGenesis, proxy, gateway, controller

I. INTRODUCTION

The current Internet TCP/IP protocols were proposed in the 70's. At that time, computing technology and digital communications had just started and technological perception of the society was very limited, since there were no mobile devices, tablets, social networks, or even the Internet. Due to the fast technological evolution and the increase on the number of connected devices, the Internet architecture has become an amazing artifact, invading all aspects of our lives. However, in many ways the current Internet and its related technologies are a victim of their own success. Networking infrastructure is becoming more and more complex, requiring a lot of human intervention (operational costs). New protocols are created to fulfill gaps in previous designs without adequate analysis of unintended effects. Many design inconsistencies were created, increasing inefficiency and complexity of the current protocol stacks. We became masters at keeping this complex network running and little has been done to extract simplicity from the lessons learned in the last decades [1][2]

In this context, some networking researchers, like Scott Shenker, Nick McKeown, and Martin Casado, to name a few of them, have proposed a revision of the current networking design abstractions. They defend the idea that abstractions have played a big role on computer science for decades, shielding high-level software from the complexity existing in lower levels. Therefore, they contend that it is time to apply the "power of abstractions" for networking, since the role of software on the contemporary technologies is increasing considerably. They argued that it is time to review the key abstractions we are employing in network design. In this scenario, they proposed a "software engineering-like" approach to design communication networks. What was called software-defined networking (SDN) [2].

The first target of SDN proposal is on simplifying network control. For this aim, four abstractions have been proposed: (i) forwarding abstraction, which encompasses a flexible, software-controlled, frame forwarding model to decouple data plane from control plane; (ii) a state distribution abstraction, which comprehends a centralized control program that operates over a summarized network view, aiming at reducing the inconsistencies of distributed states (as in the current Internet data plane); (iii) a configuration abstraction, where the output of the control program is transported to the controlled equipment by a control protocol, creating what can be called a network operating system (NOS); and (iv) a specification abstraction, that enables the generation of abstract configurations for network devices, enabling the creation of virtual topologies over the physical ones.

OpenFlow (OF) [3] has emerged as a configuration abstraction implementation. It defines the structure of a switch, as well as the protocol used by the control program (controller) to generate the network view and to configure forwarding tables on physical equipment. A diversity of network controllers can be used together with OpenFlow, among them NOX, HyperFlow, DevoFlow, Onix, POX, etc. The specification abstraction is left for controller clients, such as FlowVisor, which enables creation of isolated slices of network resources through orches-

tration of OF controllers. It can be seen as a network "hypervisor", analogous to a virtual machine monitor in computer virtualization technology. The aim is to provide an ecosystem of network applications that run over this NOS.

While OF facilitates the development of network engineering applications over NOS, there is a gap between this ecosystem (for network control) and the **software-as-a-service** (SaaS) ecosystem. Service-oriented architecture (SOA) [10] provides dynamic orchestration of services, supporting their entire lifecycling, such as service and substrate resources discovery, negotiation, contracting, and releasing. SOA provides a business-oriented environment where services placement is negotiated with physical resources representatives. SOA abstractions enable to expose and negotiate computing and network resources to create service-awareness, i.e. customized networks that are tuned for better supporting applications requirements.

In this context, there is no impediment to think on OF, or more generally, on SDN software, as services in SOA context. The advantage of this approach would be a better integration of SOA and SDN approaches. Network operating systems can be implemented as SOA services, enabling dynamic exposition, discovery, contracting, and releasing of networking controllers to create a **service-defined architecture** (SDA). SDN software is implemented as services, and all their capabilities are transparently exposed to SOA environment. As a result, any user application can search, negotiate, and contract SDN software to create service-aware networks.

We are developing an information and communications technologies (ICT) architecture called NovaGenesis (NG) [4], which envisions the convergence of SOA, SDN, and content-centric networking (CCN) [11]. In NG, contents and services are uniquely named. Named-content and named-services lifecycling is provided, including search, negotiation, contracting, and releasing. NG provides a powerful environment for resource management and control. The idea is to create an environment where services and content can be self-organized according to users' objectives, rules, regulations, expectations, etc. Distributed, named-services self-organize to provide self-emergent behavior, like resource management, networking control, etc. Thus, in NovaGenesis point of view, optimal network control is obtained as a result of fair dynamic negotiation among services and network representatives, controllers, operating systems, and hypervisors.

In this scenario, we propose a NovaGenesis service called **Python OpenFlow controller agent** (POXA) to fulfill the gap between NG services and OF-based SDN. The POXA accumulates proxy/gateway/controller (PGC) functionalities, which will be further explained in this paper. The aim is to represent OF software inside the NovaGenesis, enabling NG services to dynamically contract OF resources. The paper also contributes with a SOA, SDN, and CCN convergence model, which can be used by researchers as a reference to integrate these paradigms. The remaining of the paper is organized as follows. Section II introduces the NovaGenesis architecture. Section III briefly describes the Python OpenFlow controller (POX). Section IV presents POXA, which allows the integration of NG proposal with OF controlled networks. Section V reports the implementation of the POXA and its validation together with POX. Finally, the Section VI concludes the paper.

II. NovaGenesis Architecture

NovaGenesis architecture [4] is a set of distributed systems aimed at convergent information processing, storage, and exchanging. Therefore, it includes the scope of the current Internet, SDN, CCN, and many others information architectures. A complete review of the technologies considered on NG design can be found in reference [7]. The synergies among these technologies have been explored in reference [8].

In NG design, every information processor is seen as a service, which includes networking applications, network protocol implementations, cloud services, peer-to-peer applications, machine-to-machine services, etc. These services organize themselves based on names and service contracts to meet "semantics rich" goals, policies, regulations, etc. By "semantics rich" we mean the employment of expressive languages that can properly feed artificial intelligence decision making, such as rule-based or fuzzy logic systems.

A. Fundamental Concepts and Design Abstractions

1) Individual Existences

An "individual" existence is anything that can be classified as independent or separated from others. It is an individual in a population. For example, a process in a distributed system or bit in a message.

2) Naming and Naming Structure

To support "semantics rich" expressiveness, NG adopts natural language names (NLNs), as well as self-certifying names (SCNs). NG rethinks naming and the role of naming on ICT architectures. By definition, a name is a set of symbols that denote some existence, e.g. a car, a switch, or a controller. NLNs can be in English or any other language. For example, one can call its local domain OF controller as "my OF controller". Meanwhile, SCNs are the output of hash functions – the so called hash codes – that can be attributed to individual or group existences. An example of SCN is `76B3622BE503236C F03F55CB32860108`, which is a 128 bits *Murmurhash* 3 output for a host English name attributed by Linux. NG assumes that all existences will be present on future information architectures, thus all names are important. In contrast, the current Internet naming is very limited, since the idea of important "things" to be named in the 70's was completely different than today.

3) Contents and Name Bindings

The binary pattern of a digital content can be used as the input to generate a hash code to name this content. Unique, perennial attributes can be used to generate SCNs for almost every existence. NovaGenesis considers **name bindings** (NBs), i.e. the link between names and "things", physical or virtual, as the foundation to represent existences "semantics" relationships, including "contains", "is contained", and "same type" semantics operators. NBs are also seen as links among names. Thus, NG is grounded on name bindings to represent all entities relationships. Content, services, operating systems, network interfaces, hosts, switches, networks, domains, people, etc., can have their NLNs and/or SCNs linked to represent a diversity of "semantics" relations.

4) Identification and Localization

Names that observe certain rules can be used as identifiers and locators. An unique name in a certain scope can be used as an identifier for communication in this scope, while a locator is a name that provides a notion of distance among existences in a certain space. For example, consider a scope where a person's name (let's say "John") and its mobile device's name are unique. In this scope, the person's name can be related to the device's name in order to express an "ownership" relation. Another person, let's say "Mary", can use "John" as the target for a communication. From a NB between "John" and its mobile device's name, the architecture can determine a possible device for this communication. Also, the two person's names, i.e. "John" and "Mary", can be bound to a certain room's name, e.g. "Laboratory 1". Thus, the room's name can be used to limit the space of this communication. Since both persons have named devices inside the same room, a notion of distance can be derived from devices' SCNs and the NBs among the persons names and the room name. Therefore, with proper scoping and space definitions it is possible to use NLNs and SCNs as identifiers and locators.

5) Name Binding and Content Publishing and Subscribing

New architectures have the opportunity to change the traditional "receiver accepts all" paradigm to a publish/subscribe one. In the pub/sub paradigm, contents are published by services and subscribed by others. Thus, a service publishes a content (or a NB) and authorizes other services to subscribe. A target communicating service needs to subscribe the information before transferring it. This enables authorized content and name bindings distribution.

6) Software-as-a-Service

This paradigm proposes that software can be seen as a service, i.e. an individual existence (or a group of existences) aimed at processing, exchanging, or storing information. In this paper, we assume that all information handling is done by software implemented as services, following a SOA terminology. In this context, even protocol implementations are seen as services. A protocol is a shared language. They are the rules – the previous knowledge required to communicate. Thus, we propose the concept of **protocol-implemented-as-a-service** (PIaaS). Protocol implementations, SDN controllers, operating systems, virtualization software, etc., are seen as services.

7) Name-Binding-Based Service and Content Lyfe-cycling

The life-cycle of digital contents and services is name-binding-based. In other words, life-cycling is based on name bindings and their publishing and subscribing. The search for service partners and their contents is name-based. Services publish and subscribe content by their names. The pub/sub service creates a distributed middleware where services can self-organize using NLNs, SCNs, and their bindings.

8) Forwarding and Routing

SCN-based forwarding is employed. Every host has an unique SCN, which is used as a target for communication. A gateway service translates SCNs to legacy addresses when NG run over already established technologies, such as Ethernet, Wi-Fi, and even TCP/IP. During bootstrapping, other technologies' addresses are informed to NG gateways. The gateway publishes mappings as name bindings. Therefore, frame forwarding can be done using SCNs and/or NLNs, e.g. named content, services, operating systems, domains, networks, etc. Routing is supported by recursive subscription of name bindings. Unknown identifiers or locators can be subscribed via pub/sub service.

9) Proxy Services

Physical existences are represented by **proxy services** that can search, negotiate, and contract other services. Dynamic service level agreements (SLAs) can be established among physical existences' representatives and other architecture services, such as network management and controlling services or resource orchestration brokers. Network awareness can be obtained through the establishment of SLAs with these proxy services. Also, high level services can search, negotiate, and contract networking, storage, and processing resources via their representative services.

The SLA is an information object that contains service requirements, clauses to be respected, evaluation and finalization criteria, among other data. All the information required to describe and regulate a service offer could be included.

B. Current Implementation

1) Hash Table Service (HTS)

Several instances of this service at different hosts store name bindings using a hash table data structure. The result is a distributed hash table that follows the naming and services conventions of NG. The HTS is a network cache. Two methods have been implemented: store, get, and delivery.

2) Generic Indirection Resolution Service (GIRS)

This service aims at selecting the proper HTS instance to store a certain name binding and/or content. In the current implementation, GIRS decides to which HTS a NB or a content must be send using SCN binary patterns. Future versions will explore NB and content placement according to their usage, i.e. to create a localization-aware storage solution.

3) Publish/Subscribe Service (PSS)

Implements name binding and content publishing and subscribing. In the current version, services publish NBs and contents to other services. Interested services should subscribe them using NLNs and/or SCNs. In future versions, the PSS will implement the secure rendezvous among publishers and subscribers. Figure 1 illustrates the relationship among HTS, GIRS, and PSS. Three methods have been implemented: pub, sub, and notify.

4) Proxy/Gateway Service (PGS)

The communication among NG services is done using ASCII messages. To forward messages among different host computers, we implemented a proxy-gateway service (PGS). The PGS acts as a gateway, encapsulating NG messages via Linux sockets. A PGS also represents and exposes other NG core services to other domain PGSs, enabling NG protocols to initialize properly.

5) Message Format

Messages contain several command lines and a payload (not obligatory). There is a bank line separating both portions, when necessary. The command lines portion is composed by a

set of command lines, which can be dynamically expanded according to the need. They are formatted as:

ng -command --alternative version [*vectorial arguments*]

Where:

- *command* is the action to be done.
- *alternative* selects among alternatives of how the action can be done.
- *version* is the desired version of implementation.
- *vectorial arguments* are the arguments of the command.

Each command can have one or more vectorial arguments, which are structured as follows:

< n type E1 E2 E3 E4 ... En >

Where:

- *n* is the number of elements in the argument vector.
- *type* is the type of the elements in the argument vector.
- *E1 E2 E3 E4 ... En* are the elements of the argument vector.

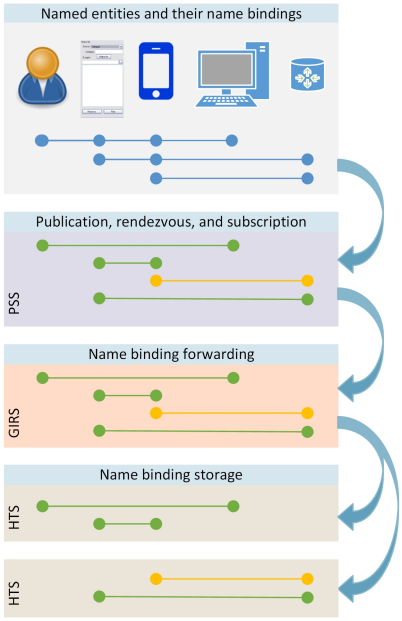

Fig. 1. NovaGenesis services for distributed publishing/subcribing of name-bindings and content.

III. POX – Python OpenFlow Controller

The POX [5] had arisen as an OF controller that provides the fast development of networking control using Python programming language. The components embedded in the POX are modules developed in Python, which combined may offer many alternatives to the development of networking applications. Following the POX components example, it is possible to create new components to be used as networking control applications in the OF network operational system, on the same way that it has been developed for computers operational systems nowadays.

IV. POXA – POX Agent

The POXA is a proxy/gateway/controller (PGC) service that represents a POX inside the NG environment. It exposes POX capabilities, as well as its internal structure using NLNs and SCNs. More specifically, POXA publishes named information objects that describe POX capabilities. This description can be in any language. However, extensible markup language (XML) is a good candidate. These descriptive information objects are published by POXA using keywords that foster understanding with possible clients. A NovaGenesis service that is searching for the services provided by POXA, can subscribe similar keywords, generating a ranking of possible peer services. When a possible client discovers a promising POXA, it can publishes a service request. Or, in the opposite direction, the POXA can offer its services for candidate peers, enabling them to decide on accepting or not a service offer.

The POXA is also a gateway between NG and the Python command line interface provided by POX. In other words, NG messages contain in its payloads generic SDN instructions that are parsed and translated to Python by the POXA. Therefore, NG enables the creation of a generic SDN language for network control, which can be translated to OF protocol or any other SDN implementation. After translating the generic SDN instructions to POX compatible commands, the POXA forwards the commands to POX and captures the obtained results.

In this implementation, the SLA is very simple. It just confirms the identification of involved peers. However, more sophisticated SLA models can be easily implemented. For example, availability, response time, and reliability requirements of a SDN controller can be inserted in the SLA to enable quality of service agreement. Complex negotiation can be introduced among POXA and other NG services. In this case, client services can subscribe and evaluate POXA published service offers and decide on accepting them as they are or proposing modifications. The SLA is about performance, availability, and security requirements.

The POXA represents POX interests at negotiating SLAs in NG ecosystem. Any other NG service can negotiate SLAs with the POXA to make changes at the network level using OpenFlow. The POXA demonstrates the NG distributed, self-organizing, network controlling approach, where several SDN controllers (OpenFlow or other) are represented and control network resources according to negotiated SLAs. The joint NG PGC and OF approach provides a more broad and flexible solution that makes network control a reflection of the SLAs. In this paper, we propose a resource management agent (RMA) as the client of POXA service. An SLA is established between the RMA and the POXA before control information exchanging.

V. Experimental Results

We tested POXA together with other NG services and the POX controller in laboratory as illustrates the Fig. 2. The objective was to validate our proposal to fulfill the gap between SOA and OF-based SDN.

A. Network Setup Scenario

The network setup chosen test scenario is formed by:

- An OpenFlow access point (SWITCH-OF), based on *OFSoftSwitch* [9], which runs over OpenWRT [6] in a Linksys WRT54GL™ wireless router;

- An OpenFlow controller (OF-C), based on POX, and a NovaGenesis service (POXA), running on a Fedora Linux at Dell™ Ultrabook;
- A NG Node (Host NG), running a RMA that determines the need for installing a forwarding flow on the SWITCH-OF. The RMA orchestrates the combined resources via OF and NG implementations;
- A number of hosts connected to the SWITCH-OF.

Fig. 2. Picture of the experimental scenario at the Inatel's ICT laboratory.

Fig. 3 shows a sequence diagram of the messages exchanged in this scenario to install the flow on the switch, all requested by the RMA running on the host where the left host contains a RMA that sends to the POXA a command to create a frame forwarding flow using NG protocol. The command contains the source and destination MAC addresses. The POXA translates the command to a classic interprocess communication (IPC) message and forwards it to the POX. The POX creates a *flow_mod* message and forwards it to the access point (AP) using the OF. The AP answers back with an OK which is propagated to the POXA first and finally to the RMA.

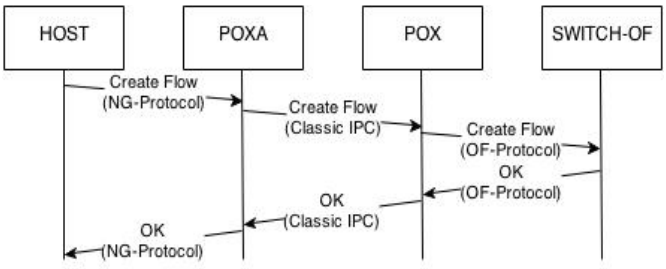

Fig. 3. Messages flow between Host and Switch-OF.

Fig. 4 illustrates the proposed stack. Not only the RMA, but also the POXA are interconnected by PGSs. In this scenario, the NG cloud employs only the Ethernet protocol to forward its messages - there is no TCP/IP inside the NG cloud. However, the POX uses TCP/IP over Ethernet to talk outside the NG cloud with the access point. There is no TCP/IP inside the NG cloud. The NG messages are transported directly over Ethernet.

B. Testbed Results

After running the previous scenario setup, we checked the flow creation in the SWITCH-OF using the Linux *'dpctl dump-flows'* command-line utility, which sends an OF message to the SWITCH-OF and gets the requested information about the flow entries:

```
# dpctl dump-flows tcp:127.0.0.1
stats_reply (xid=0x25156283): flags=none type=1(flow)
```

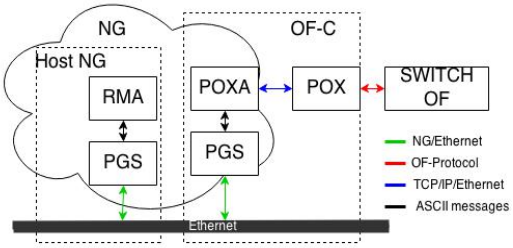

Fig. 4. Protocol stack inside and outside the NG cloud.

This status shows that there are no flows allocated on the access point (SWITCH-OF). Using NG protocol, the RMA sends to the POXA the command line to ask for a flow entry creation in the SWITCH-OF. This command line goes inside a textual file called POXAFile1.txt and has 14 bytes. The following log shows the RMA creating a message with this command line to the POXA. The *–msg* command line is for forwarding and routing, while the *–scn* is for message integrity. The *–run --publishpoxadata* command runs a user defined code to publish the RMA control file to the POXA via PSS.

```
ng -msg --cl 0.1 [ < 1 string
6C7C2B32CBE39452B07EA3812C4E4D9A_RMA_PID > < 1 string
9FDB455D86A9691C2F41268964417375_Core_BID > < 1 string
9FDB455D86A9691C2F41268964417375_Core_BID > ]

ng -scn --seq 0.1 [ < 1 string
B03CB1A465916480A98D6E8F076E32EA > ]
ng -run --publishpoxadata 0.1 [ ]
        -msg --cl 0.1
        -msg --cl 0.1
        -scn --seq 0.1
(Preparing POXAData Publish.)
(Aware of POXA data values. Preparing to publish.)
 (Pushing a message to the InputQueue. Size = 1)
```

The following log shows a fragment of the message delivered to the POXA:

```
ng -delivery --bind 0.1 [ < 1 string 18 > < 1 string
F3D7692F6D4418D45B458E54083574FC > < 1 string POXAFile1.txt
> ]

ng -info --payload 0.1 [ < 1 string POXAFile1.txt > ]

ng -delivery --bind 0.1 [ < 1 string 2 > < 1 string
F3D7692F6D4418D45B458E54083574FC > < 3 string
9FDB455D86A9691C2F41268964417375_Core_BID
7251FC2E62B8AF6C9B5969FC611FB90C_OSID
6C7C2B32CBE39452B07EA3812C4E4D9A_RMA_PID > ]

ng -delivery --bind 0.1 [ < 1 string 9 > < 1 string
F3D7692F6D4418D45B458E54083574FC > < 1 string
76B3622BE503236CF03F55CB32860108_HID > ]

ng -message --type 0.1 [ < 1 string 1 > ]
ng -scn --ack 0.1 [ < 2 string
13A4A3F0342AC622FC2F291EFD807C09
9BB4A6879CADC53CC6445E53E03D5FDF >]


There is a payload of 14 bytes
```

This message follows a NG specific format, where addresses are 128 bits hashing codes [4] plus a debug *tag* at the end. The *–delivery* command delivers a name binding, while the *–info --payload* contains the NLN of the content being carried in the message. The *–message --type* is used to differentiate user messages from NG protocols. The *–scn --ack* has two SCNs:

one to acknowledge a previous successful processed message and other for current message integrity check. The next log shows the POXA parsing a message subscribed from PSS with the POXAFile1.txt file inside.

```
ng -msg --cl 0.1 [ < 1 string
88BD829D1F5C067304351EB0014AB174_POXA_PID > < 1 string
BDC352CC8A0B650739B1997F3D3A903C_Core_BID > < 1 string
BDC352CC8A0B650739B1997F3D3A903C_Core_BID > ]
ng -run --evaluate 0.1 [ ]
ng -scn --seq 0.1 [ < 1 string
BFAED45F3919479EF0A3761E32A94308 > ]
          -msg --cl 0.1
                    -msg --cl 0.1
                    -run --evaluate 0.1
 (Aware of the Application 0 )

(HID = 76B3622BE503236CF03F55CB32860108_HID)
(OSID = 7251FC2E62B8AF6C9B5969FC611FB90C_OSID)
(PID = 6C7C2B32CBE39452B07EA3812C4E4D9A_RMA_PID)
(BID = 9FDB455D86A9691C2F41268964417375_Core_BID)
(4. Check subscriptions)
(Testing subscription 0)
(Subscription status is processing required)
(The publisher is already known and has the index 0)
(HID = 76B3622BE503236CF03F55CB32860108_HID)
(OSID = 7251FC2E62B8AF6C9B5969FC611FB90C_OSID)
(PID = 6C7C2B32CBE39452B07EA3812C4E4D9A_RMA_PID)
(BID = 9FDB455D86A9691C2F41268964417375_Core_BID)
(Checking the file received from the peer with name POXA-
File1.txt)
(File extension = txt)
 (Command sent to OpenFlow: #□wf#######1)
 (Status received from OpenFlow: OK)
(Deleting the subscription with index = 0)
```

The next log shows the POX processing of the message received by the POXA.

```
no message
Wait...
\x01\x88\x77\x66\x03\x04\x05\x00\x01\x02\x03\x04\x06\x31

DEBUG:ng.pox_agent:installing flow
```

The log below shows the message transmitted with OF protocol, requesting the action at the SWITCH-OF. To conclude the experimental results, after the flow entry creation, we checked the flow creation status using the *'dpctl dump-flows'* command line again. It is possible to see now that there is a new flow entry created in the SWITCH-OF. This log demonstrates that we successfully integrated NG and POX, enabling a RMA to create flows inside a Wi-Fi Access Point via OF.

```
# dpctl dump-flows tcp:127.0.0.1:6634
stats_reply (xid=0xc5da1d53): flags=none type=1(flow)
cookie=0, duration_sec=125s,
duration_nsec=649000000s,table_id=1,priority=32768,
n_packets=0,n_bytes=0,idle_timeout=0,
hard_timeout=0,dl_src=00:01:02:03:04:06,
dl_dst=88:77:66:03:04:05,actions=output:1
```

VI. Conclusion

This paper proposes an approach to reduce the gap between SDN and SOA. It proposes the concept and reports the implementation of a proxy/gateway/controller for OpenFlow resources named POXA. The POXA represents a Python OF controller inside the NovaGenesis architecture. This **controller-as-a-service** (CaaS) approach enables the exposition, negotiation, contracting, and releasing of OpenFlow capabilities inside the NG cloud. The POXA receives NG messages containing the desired physical network configurations from a resource management agent (RMA) and maps these configurations to POX commands, making the required changes via OF protocol.

The obtained solution merges NovaGenesis SOA with OF-based SDN to create a new paradigm that we named **service-defined architecture** (SDA). Therefore, this paper contributes for advancing SDN technology by proposing integrative SDN and SOA abstractions, such as name-based exposition, controller-as-a-service, SLA-based orchestration of SDN exposed resources, and PGC-based interoperability. The consistency of distributed states is provided by a common view of configurations that is distributed using NovaGenesis published/subscribe service. The exchange of ASCII messages will be replaced in future versions by a public cryptography approach already designed for NovaGenesis.

Although the distributed nature of POXA is one of the main contributions of this paper, there are several other contributions inherited from NovaGenesis that we have experimentally demonstrated: name-based content and services orchestration, contract-based and proxy-oriented management and control, as well as OF resources exposition. Future work includes a full scalability and performance evaluation of several POXA+POX instances, as well as the implementation of NG services that can dynamically explore OF-based SDN resources.

Acknowledgment

We would like to thanks CNPq and INATEL for supporting POXA design and development.

References

[1] Singh D. "Developing an Architecture: Scalability, Mobility, Control, and Isolation on Future Internet Services", Second International Conference on Advances in Computing, Communications and Informatics (ICACCI-2013), Mysore, India, pp.1873-1877, 2013.

[2] https://www.opennetworking.org, Accessed on April 20th, 2014.

[3] N. McKeown et al., "OpenFlow: Enabling Innovation in Campus Networks," SIGCOMM Computer Communications Review, vol. 38, nº 2, March 2008, pp. 60-69.

[4] http://www.inatel.br/novagenesis, Accessed on April 20th, 2014.

[5] https://github.com/noxrepo/pox, Accessed on April 20th, 2014.

[6] https://openwrt.org/, Accessed on April 20th, 2014.

[7] Alberti, A. M., "A Conceptual-Driven Survey on Future Internet Requirements, Technologies, and Challenges". Journal of the Brazilian Computer Society, v. 19, p. 291-311, 2013.

[8] Alberti, A. M., "Searching for Synergies among Future Internet Ingredients", International Conference on Convergence and Hybrid Information Technology, pp.61-68, 2012.

[9] https://github.com/CpqD/ofsoftswitch13, Accessed on April 20th, 2014.

[10] Papazoglou M., Traverso, P., Dustdar, S., Leymann, F., "Service-Oriented Computing: State of the Art and Research Challenges," Computer 40 (2007) 38 –45.

[11] Jacobson, V. et al. "Networking Named Content," CoNEXT '09, ACM, New York, NY, USA, 2009, pp. 1–12.