NMRG
Internet-Draft
Updates: draft-pedro-nmrg-intelligent-reasoning-00 (if approved)
Intended status: Informational
Expires: September 7, 2020

P. Martinez-Julia, Ed.
NICT
S. Homma
NTT

March 06, 2020

Intelligent Reasoning on External Events for Network Management
draft-pedro-nmrg-intelligent-reasoning-01

Abstract

The adoption of AI in network management solutions is becoming a reality. It is mainly supported by the need to resolve complex problems arisen from the acceptance of SDN/NFV technologies as well as network slicing. This allows current computer and network system infrastructures to constantly grow in complexity, in parallel to the demands of users. However, exploiting the possibilities of AI is not an easy task. There has been a lot of effort to make Machine Learning (ML) solutions reliable and acceptable but, at the same time, other mechanisms have been forgotten. It is the particular case of reasoning. Although it can provide enormous benefits to management solutions by, for example, inferring new knowledge and applying different kind of rules (e.g. logical) to choose from several actions, it has received little attention. While ML solutions work with data, so their only requirement from the network infrastructure is data retrieval, reasoning solutions work in collaboration to the network they are managing. This makes the challenges arisen from intelligent reasoning to be a key for the evolution of network management towards the full adoption of AI.

Table of Contents

1. Introduction

The current network ecosystem is quickly evolving from an almost fixed network to a highly flexible, powerful, and somehow hybrid system. Network slicing, Software Defined Networking (SDN), and Network Function Virtualization (NFV) provide the basis for such evolution. The need to automate the management and control of such systems has motivated the move towards autonomic networking (ANIMA) and the inclusion of AI solutions alongside the management plane of the network, enough justified by the increasing size and complexity of the network, which exposes complex problems that must be resolved in scales that escape human possibilities. Therefore, in order to allow current computer and network system infrastructures to constantly grow in complexity, in parallel to the demands of users, the AI solutions must work together with other network management solutions.

However, exploiting the possibilities of AI is not an easy task. There has been a lot of effort to make Machine Learning (ML) solutions reliable and acceptable but, at the same time, other mechanisms have been forgotten. It is the particular case of reasoning. Although it can provide enormous benefits to management solutions by, for example, inferring new knowledge and applying different kind of rules (e.g. logical) to choose from several actions, it has received little attention. While ML solutions work with data, so their only requirement from the network infrastructure is data retrieval, reasoning solutions work in collaboration to the network they are managing. This makes the challenges arisen from intelligent reasoning to be a key for the evolution of network management towards the full adoption of AI.

The present document aims to gather the necessary information for getting the most benefits from the application of intelligent reasoning to network management, including, but not limited to,

defining the gaps that must be covered for reasoning to be correctly integrated into network management solutions.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

3. Background

3.1. Virtual Computer and Network Systems

The continuous search for efficiency and cost reduction to get the most optimum exploitation of available resources (e.g. CPU power and electricity) has conducted current physical infrastructures to move towards virtualization infrastructures. Also, this trend enables end systems to be centralized and/or distributed, so that they are deployed to best accomplish customer requirements in terms of resources and qualities.

One of the key functional requirements imposed to computer and network virtualization is a high degree of flexibility and reliability. Both qualities are subject to the underlying technologies but, while the latter has been always enforced to computer and network systems, flexibility is a relatively new requirement, which would not have been imposed without the backing of virtualization and cloud technologies.

3.2. SDN and NFV

SDN and NFV are conceived to bring high degree of flexibility and conceptual centralization qualities to the network. On the one hand, with SDN, the network can be programmed to implement a dynamic behavior that changes its topology and overall qualities. Moreover, with NFV the functions that are typically provided by physical network equipment are now implemented as virtual appliances that can be deployed and linked together to provide customized network services. SDN and NFV complements to each other to actually implement the network aspect of the aforementioned virtual computer and network systems.

Although centralization can lead us to think on the single-point-of-failure concept, it is not the case for these technologies. Conceptual centralization highly differs from centralized deployment. It brings all benefits from having a single point of decision but retaining the benefits from distributed systems. For instance, control decisions in SDN can be centralized while the mechanisms that

enforce such decisions into the network (SDN controllers) can be implemented as highly distributed systems. The same approach can be applied to NFV. Network functions can be implemented in a central computing facility, but they can also take advantage of several replication and distribution techniques to achieve the properties of distributed systems. Nevertheless, NFV also allows the deployment of functions on top of distributed systems, so they benefit from both distribution alternatives at the same time.

3.3. Management and Control

The introduction of virtualization into the computer and network system landscape has increased the complexity of both underlying and overlying systems. On the one hand, virtualizing underlying systems adds extra functions that must be managed properly to ensure the correct operation of the whole system, which not just encompasses underlying elements but also the virtual elements running on top of them. Such functions are used to actually host the overlying virtual elements, so there is an indirect management operation that involves virtual systems. Moreover, such complexities are inherited by final systems that get virtualized and deployed on top of those virtualization infrastructures.

In parallel, virtual systems are empowered with additional, and widely exploited, functionality that must be managed correctly. It is the case of the dynamic adaptation of virtual resources to the specific needs of their operation environments, or even the composition of distributed elements across heterogeneous underlying infrastructures, and probably providers.

Taking both complex functions into account, either separately or jointly, makes clear that management requirements have greatly surpassed the limits of humans, so automation has become essential to accomplish most common tasks.

3.4. Slice Gateway (SLG)

A slice gateway (SLG) (see [I-D.homma-nfvrg-slice-gateway]) is basically a component in the data plane and has the roles of data packet processing. Moreover, it provides an interface to export its functions for interacting with control and management components, so that it is quite relevant for implementing the requirements described above within the network slicing domain.

Furthermore, an SLG might be required to support handling services provided on network slices in addition to controlling them because an SLG is the edge node on an end-to-end network slice (E2E-NS).

Therefore, the SLG exposes the following requirements:

Data plane for NSs as infrastructure.

Control/management plane for NSs as infrastructure.

Data plane for services on NSs.

Control/management plane for services on NSs.

In summary, SLG provides the required functions for the enforcement of AI decisions in multi-domain (and federated) network slices, so it will play a key role in general network management.

4. Applying AI to Network Management

4.1. Beyond Machine Learning

ML is not AI. AI has a broader spectrum of methods, some of them are already exploited in the network for a long time. Perception, reasoning, and planning are still not fully exploited in the network.

4.2. Briefing Artificial Intelligence

Intelligence does not directly imply intelligent. On the one hand, intelligence emphasizes data gathering and management, which can be processed by systematic methods or intelligent methods. On the other hand, intelligent emphasizes the reasoning and understanding of data to actually "posses" the intelligence.

The justification of applying AI in network (and) management is sometimes overseen. First, management decisions are more and more complex. We have moved from asking simple questions ("Is there a problem in my system?") to much more complex ones ("Where should I migrate this VM to accomplish my goals?"). Moreover, operation environments are more and more dynamic. On the one hand, softwarization and programmability elevate flexibility and allow networks to be totally adapted to their static and/or dynamic requirements. On the other hand, network virtualization highly enables network automation.

The new functions and possibilities allow network devices to become autonomic. However, they must take complex decisions by themselves, without human intervention, realizing the "dream" of Zero-Touch Networks (ZTM), which exploit fully programmable elements and advanced automation methods (ETSI ZSM). Nevertheless, we have to remember that AI methods are just resources, not solutions. They

will not replace the human decisions, just complement and "automate" them.

5. Extended Management Operation

5.1. Intelligent Network Management Process

In general, the correct and pertinent application of AI to network management provides enormous benefits, mainly in terms of making complex management operations feasible and improving the performance of typically expensive tasks. By taking advantage of these benefits, the amount of data that can be analyzed to make decisions on the network can be hugely increased.

As a result, AI makes possible to enlarge the management process towards the Intelligent Network Management Process (INMP). Instead of just being focused on the analysis of performance measurements retrieved from the managed network and the subsequent decision (proaction or reaction), the extension of management operation enabled by INMP encompasses different sub-processes.

First, INMP has a sub-process for retrieving the performance measurements from the managed network. This is the same found in typical management processes. Moreover, INMP encourages the application of the same ML techniques to obtain some insight of the situation of the managed network.

Second, INMP incorporates a reasoning sub-process. It receives both the output of the previous sub-process and additional context information, which can be provided by an external event detector, as described below. Then, this sub-process finds out and particularizes the rules that are governing the situation described above. Such rules are semantically constructed and will abstract the situation of the network in terms of logical and other semantic concepts, together with actions and transformations that can be applied to those rules. All such constructions will be stored in the Intelligent Network Management Knowledge Base (INMKB), which will follow a pre-determined ontology and will also extend the knowledge by applying basic and atomic logic inference statements.

Third, INMP defines the solving sub-process. It works as follows. Once obtained the abstracted situation of the managed network and the rules to it, the solving subprocess builds a graph with all semantic constructions. It reflects the managed network, since all network elements have their semantic counterpart, but it also has all situations, rules, actions, and even the measurements. The solving sub-process applies ontology transformations to find a graph that is

acceptable in terms of the associated situation and its adherence to administrative goals.

Fourth, INMP incorporates the planning sub-process. It receives the solution graph obtained by the previous sub-process and makes a linear plan of actions to execute in order to enforce the required changes into the network. The actions used by this planning sub-process are the building blocks of the plan. Each block will be defined with a precondition, invariant, and postcondition. A planning algorithm should be used to obtain such plan of actions by linking the building blocks so they can be enforced to finally adapt the managed network to get the desired situation.

All these processes must be executed in parallel, using strong inter-process communication and synchronization constraints. Moreover, the requests to the underlying infrastructure for the adaptation of the managed network will be sent to the corresponding controllers without waiting for finishing the deliberation cycle. This way, the time required by the whole cycle is highly reduced. This can be possible because of the assumptions and anticipations tied to INMP and the intelligence it denotes.

5.2. Closed Loop Management Approach

Beginning with INMP, a key approach for achieving proper network management goals is to follow the closed control loop methodology. It ensures that the objectives are not just accomplished at certain moment but kept in future cycles of both management and network life-cycle.

To obtain the benefits from integrating AI within the closed loop, INMP processes must be re-wired to connect their outputs to their inputs, so obtaining feedback analysis. Moreover, an additional process must be defined for ensuring that the objectives defined in the last steps of INMP are actually present in the near future situation of the managed network.

In addition, the data plane elements, such as the SLG described above, must provide some capabilities to make them coherent to the closed control loop. Particularly, they must provide symmetric enforcement and telemetry interfaces, so that the elements composing the managed network can be modified and monitored using the same identifiers and having the same assumptions about their topology and context. For instance, SLG must be able to provide the needed functionality to enable INMP to request SLG to set up and connect the necessary structures for telemetry collection and request slice switching.

6. Deep Exploitation of AI in Network Management

6.1. From Data to Wisdom

As AI methods gain access to a huge amount of (intelligence) data from the systems they manage, they become more and more able to take strategic decisions, mainly deriving such data to knowledge towards wisdom. This supports the well known DIKW process (Data, Information, Knowledge, Wisdom) that enables elements to operate autonomously, subject to the goals established by administrators.

In such way, AI methods can be guided by the events or situations found in underlying networks in a constantly evolving model. We can call it the Knowledge (and Intelligence) Driven Network. In this new network architecture, the structure itself of the network results from reasoning on intelligence data. The network adapts to new situations without requiring human involvement but administrative policies are still enforced to decisions. Nevertheless, intelligence data must be managed properly to exploit all its potential. Data with high accuracy and high frequency will be processed in real-time. Meanwhile, fast and scalable methods for information retrieval and decision enfrocement become essential to the objectives of the network.

To achieve such goals, AI algorithms must be adapted to work on network problems. Joint physical and virtual network elements can form a multi-agent system focused on achieving such system goals. It can be applied to several use-cases. For instance, it can be used for predicting traffic behaviour, iterative network optimization, and assessment of administrative policies.

6.2. External Event Detectors

As mentioned above, current mechanisms used to achieve automated management and control rely only on the continuous monitoring of the resources they control or the underlying infrastructure that host them. However, there are several other sources of information that can be exploited to make the systems more robust and efficient. It is the case of the notifications that can be provided by physical or virtual elements or devices that are watching for specific events, hence called external event detectors.

More specifically, although the notifications provided by these external event detectors are related to successes that occur outside the boundaries of the controlled system, such successes can affect the typical operation of controlled systems. For instance, a heavy rainfall or snowfall can be detected and correlated to a huge

increase in the amount of requests experienced by some emergency support service.

6.3. Network Requirement Anticipation

One of the main goals of the MANO mechanisms is to ensure the virtual computer and network system they manage meets the requirements established by their owners and administrators. It is currently achieved by observing and analyzing the performance measurements obtained either by directly asking the resources forming the managed system of by asking the controllers of the underlying infrastructure that hosts such resources. Thus, under changing or eventual situations, the managed system must be adapted to cope with the new requirements, increasing the amount of resources assigned to it, or to make efficient use of available infrastructures, reducing the amount of resources assigned to it.

However, the time required by the infrastructure to make effective the adaptations requested by the MANO mechanisms is longer than the time required by client requests to overload the system and make it discard further client requests. This situation is generally undesired but particularly dangerous for some systems, such as the emergency support system mentioned above. Therefore, in order to avoid the disruption of the service, the change in requirements must be anticipated to ensure that any adaptation has finished as soon as possible, preferably before the target system gets overloaded or underloaded.

Here we link the application of AI to network management to ARCA (Appendix B). It is integrated to NFV-MANO to enable the latter to take advantage of the events notified by the external event detectors, by correlating them to the target amount of resources required by the managed system and enforcing the necessary adaptations beforehand, particularly before the system performance metrics have actually changed.

The following abstract algorithm formalizes the workflow expected to be followed by the different implementations of the operation proposed here.

```
while TRUE do
    event = GetExternalEventInformation()
    if event != NONE then
        anticipated_resource_amount = Anticipator.Get(event)
        if IsPolicyCompliant(anticipated_resource_amount) then
            current_resource_amount = anticipated_resource_amount
            anticipation_time = NOW
        end if
    end if
    anticipated_event = event
    if anticipated_event != NONE and
            (NOW - anticipation_time) > EXPIRATION_TIME then
        current_resource_amount = DEFAULT_RESOURCE_AMOUNT
        anticipated_event = NONE
    end if
    state = GetSystemState()
    if not IsAcceptable(state, current_resource_amount) then
        current_resource_amount = GetResourceAmountForState(state)
        if anticipated_event is not NONE then
            Anticipator.Set
                (anticipated_event, current_resource_amount)
            anticipated_event = NONE
        end if
    end if
end while
```

This algorithm considers both internal and external events to determine the necessary control and management actions to achieve the proper anticipation of resources assigned to the target system. We propose the different implementations to follow the same approach so they can guess what to expect when they interact. For instance, a consumer, such as an Application Service Provider (ASP), can expect some specific behavior of the Virtual Network Operator (VNO) from which it is consuming resources. This helps both the ASP and VNO to properly address resource fluctuations.

6.4. Intelligent Reasoning

It is trivial for anybody to understand that the behavior or the network results from user activity. For instance, more users means more traffic. However, it is not commonly considered that user activity has a direct dependency on events that occur outside the boundaries of the networks they use. For example, if a video becomes trendy, the load of the network that hosts the video increases, but

also the load of any network with users watching the video. In the same way, if a natural incident occurs (e.g. heavy rainfall, earthquake), people try to contact their relatives and the load of a telephony network increases. From this we can easily find out that there is a clear causality relation between events occurring in the real and digital world and the behaviour of the network (aka. The Internet).

Network management outcomes, in terms of system stability, performance, reliability, etc., would greatily improve by exploiting such causality relation. An easy and straightforward way to do so is to apply AI reasoning methods. These methods can be used to "guess" the effect for a given cause. Moreover, reasoning can be used to choose the specific events that can impact the system, so being the cause for some effect.

Meanwhile, reasoning on network behavior from performance measurements and external events places some challenges. First, external event information must cross the administrative domain of the network to which it is relevant. This means that there must be interfaces and security policies that regulate how information is exchanged between the external event detecthor, which can be some sensor deployed in some "smart" place (e.g. smart city, smart building), and the management solution, which resides inside the administrative domain of the managed network. This function must be highly conformed and regulated, and the protocols used to achieve it must be widely accepted and tested, in order for it to exploit the overall potential of external events.

Second, enough meta-data must be associated to performance measurements to clearly identify all aspects of the effects, so that they can be traced back to the causes (events). Such meta-data must follow an ontology (information model) that is somewhat common and widely accepted or, at leaset, to be able to easily transform it among the different formats and models used by different vendors and software.

Third, the management ontology must be extended by all concepts from the boundaries of the managed network, its external environment (surroundings), and any entity that, albeit being far away, can impact on the function of the managed network.

6.5. Gaps and Standardization Issues

Several gaps and standardization issues arise from applying AI and reasoning to network management solutions:

Methods from different providers/vendors must be able to coexist and work together, either directly or by means of a translator. They must, however, use the same concepts, albeit using different naming, so they actually share a common ontology.

Information retrieval must be assessed for quality so that the outputs from AI reasoning, and thus management solutions, can be reliable.

Ontological concepts must be consistent so that the types and qualities of information that is retrieved from a system or object are as expected.

The protocols used to communicate (or disseminate, or publish) the information must respond to the constraints of their target usage.

7. Relation to Other IETF/IRTF Initiatives

TBD

8. IANA Considerations

This memo includes no request to IANA.

9. Security Considerations

As with other AI mechanisms, the major security concern for the adoption of intelligent reasoning on external events to manage network slices and SDN/NFV systems is that the boundaries of the control and management planes are crossed to introduce information from outside. Such communications must be highly and heavily secured since some malfunction or explicit attacks might compromise the integrity and execution of the controlled system. However, it is up to implementers to deploy the necessary countermeasures to avoid such situations. From the design point of view, since all oprations are performed within the control and/or management planes, the security level of reasoning solutions is inherited and thus determined by the security masures established by the systems conforming such planes.

10. Acknowledgements

TBD

11. References

11.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

11.2. Informative References

[ETSI-NFV-IFA-004] ETSI NFV GS NFV-IFA 004, "Network Functions Virtualisation (NFV); Acceleration Technologies; Management Aspects Specification", 2016.

[ETSI-NFV-IFA-005] ETSI NFV GS NFV-IFA 005, "Network Functions Virtualisation (NFV); Management and Orchestration; Or-Vi reference point - Interface and Information Model Specification", 2016.

[ETSI-NFV-IFA-006] ETSI NFV GS NFV-IFA 006, "Network Functions Virtualisation (NFV); Management and Orchestration; Vi-Vnfm reference point - Interface and Information Model Specification", 2016.

[ETSI-NFV-IFA-019] ETSI NFV GS NFV-IFA 019, "Network Functions Virtualisation (NFV); Acceleration Technologies; Management Aspects Specification; Release 3", 2017.

[ETSI-NFV-MANO] ETSI NFV GS NFV-MAN 001, "Network Functions Virtualisation (NFV); Management and Orchestration", 2014.

[I-D.geng-coms-architecture] Geng, L., Qiang, L., Lucena, J., Ameigeiras, P., Lopez, D., and L. Contreras, "COMS Architecture", draft-geng-coms-architecture-02 (work in progress), March 2018.

[I-D.homma-nfvrg-slice-gateway] Homma, S., Foy, X., and A. Galis, "Gateway Function for Network Slicing", draft-homma-nfvrg-slice-gateway-00 (work in progress), July 2018.

[I-D.qiang-coms-netslicing-information-model]
Qiang, L., Galis, A., Geng, L., kiran.makhijani@huawei.com, k., Martinez-Julia, P., Flinck, H., and X. Foy, "Technology Independent Information Model for Network Slicing", draft-qiang-coms-netslicing-information-model-02 (work in progress), January 2018.

[I-D.song-ntf]
Song, H., Zhou, T., Li, Z., Fioccola, G., Li, Z., Martinez-Julia, P., Ciavaglia, L., and A. Wang, "Toward a Network Telemetry Framework", draft-song-ntf-02 (work in progress), July 2018.

[ICIN-2017]
P. Martinez-Julia, V. P. Kafle, and H. Harai, "Achieving the autonomic adaptation of resources in virtualized network environments, in Proceedings of the 20th ICIN Conference (Innovations in Clouds, Internet and Networks, ICIN 2017). Washington, DC, USA: IEEE, 2018, pp. 1--8", 2017.

[ICIN-2018]
P. Martinez-Julia, V. P. Kafle, and H. Harai, "Anticipating minimum resources needed to avoid service disruption of emergency support systems, in Proceedings of the 21th ICIN Conference (Innovations in Clouds, Internet and Networks, ICIN 2018). Washington, DC, USA: IEEE, 2018, pp. 1--8", 2018.

[OPENSTACK]
The OpenStack Project, "http://www.openstack.org/", 2018.

Appendix A. Information Model to Support Reasoning on External Events

In this section we introduce the basic model needed to support reasoning on external events. It basically includes the concepts and structures used to describe external events and notify (communicate) them to the interested sink, the network controller/manager, through the control and management plane, depending on the specific instantiation of the system.

A.1. Tree Structure

```
module: ietf-nmrg-nict-ai-reasoning
  +--rw events
     +--rw event-payloads
     +--rw external-events

  notifications:
    +---n event
```

The main models included in the tree structure of the module are the events and notifications. On the one hand, events are structured in payloads and the content of events itself (external-events). On the other hand, there is only one notification, which is the event itself.

A.1.1. event-payloads

```
+--rw event-payloads
   +--rw event-payloads-basic
   +--rw event-payloads-seismometer
   +--rw event-payloads-bigdata
```

The event payloads are, for the time being, composed of three types. First, we have defined the basic payload, which is intended to carry any arbitrary data. Second, we have defined the seismometer payload to carry information about seisms. Third, we have defined the bigdata payload that carries notifications coming from BigData sources.

A.1.1.1. basic

```
+--rw event-payloads-basic* [plid]
   +--rw plid    string
   +--rw data?   union
```

The basic payload is able to hold any data type, so it has a union of several types. It is intended to be used by any source of events that is (still) not covered by other model. In general, any source of telemetry information (e.g. OpenStack [OPENSTACK] controllers) can use this model as such sources can encode on it their information, which typically is very simple and plain. Therefore, the current model is tightly interrelated to a framework to retrieve network telemetry (see [I-D.song-ntf]).

A.1.1.2. seismometer

```
   +--rw event-payloads-seismometer* [plid]
      +--rw plid         string
      +--rw location?    string
      +--rw magnitude?   uint8
```

The seismometer model includes the main information related to a seism, such as the location of the incident and its magnitude. Additional fields can be defined in the future by extending this model.

A.1.1.3. bigdata

```
   +--rw event-payloads-bigdata* [plid]
      +--rw plid           string
      +--rw description?   string
      +--rw severity?      uint8
```

The bigdata model includes a description of an event (or incident) and its estimated general severity, unrelated to the system. The description is an arbitrary string of characters that would normally carry information that describes the event using some higher level format, such as Turtle or N3 for carrying RDF knowlege items.

A.1.2. external-events

```
   +--rw external-events* [id]
      +--rw id           string
      +--rw source?      string
      +--rw context?     string
      +--rw sequence?    int64
      +--rw timestamp?   yang:date-and-time
      +--rw payload?     binary
```

The model defined to encode external events, which encapsulates the payloads introduced above, is completed with an identifier of the message, a string describing the source of the event, a sequence number and a timestamp. Additionaly it includes a string describing the context of the event. It is intended to communicate the required information about the system that detected the event, its location, etc. As the description of the BigData payload, this field can be formated with a high level format, such as RDF.

A.1.3. notifications/event

```
notifications:
  +---n event
     +--ro id?          string
     +--ro source?      string
     +--ro context?     string
     +--ro sequence?    int64
     +--ro timestamp?   yang:date-and-time
     +--ro payload?     binary
```

The event notification inherits all the fields from the model of external events defined above. It is intended to allow software and hardware elements to send, receive, and interpret not just the events that have been detected and notified by, for instance, a sensor, but also the notifications issued by the underlying infrastructure controllers, such as the OpenStack Controller.

A.2. YANG Module

.

```
module ietf-nmrg-nict-ai-reasoning {
  namespace "urn:ietf:params:xml:ns:yang:ietf-nmrg-nict-ainm";
  prefix rant;
  import ietf-yang-types { prefix yang; }

  grouping external-event-information {
    leaf id { type string; }
    leaf source { type string; }
    leaf context { type string; }
    leaf sequence { type int64; }
    leaf timestamp { type yang:date-and-time; }
    leaf payload { type binary; }
  }

  grouping event-payload-basic {
    leaf plid { type string; }
    leaf data { type union { type string; type binary; } }
  }

  grouping event-payload-seismometer {
    leaf plid { type string; }
    leaf location { type string; }
    leaf magnitude { type uint8; }
  }

  grouping event-payload-bigdata {
    leaf plid { type string; }
    leaf description { type string; }
```

```
      leaf severity { type uint8; }
    }

    notification event {
      uses external-event-information;
    }

    container events {
      container event-payloads {
        list event-payloads-basic {
          key "plid";
          uses event-payload-basic;
        }
        list event-payloads-seismometer {
          key "plid";
          uses event-payload-seismometer;
        }
        list event-payloads-bigdata {
          key "plid";
          uses event-payload-bigdata;
        }
      }
      list external-events {
        key "id";
        uses external-event-information;
      }
    }

  }

  .
```

Appendix B. The Autonomic Resource Control Architecture (ARCA)

As deeply discussed in ICIN 2018 [ICIN-2018], ARCA leverages the elastic adaptation of resources assigned to virtual computer and network systems by calculating or estimating their requirements from the analysis of load measurements and the detection of external events. These events can be notified by physical elements (things, sensors) that detect changes on the environment, as well as software elements that analyze digital information, such as connectors to sources or analyzers of Big Data. For instance, ARCA is able to consider the detection of an earthquake or a heavy rainfall to overcome the damages it can make to the controlled system.

The policies that ARCA must enforce will be specified by administrators during the configuration of the control/management engine. Then, ARCA continues running autonomously, with no more

human involvement unless some parameter must be changed. ARCA will adopt the required control and management operations to adapt the controlled system to the new situation or requirements. The main goal of ARCA is thus to reduce the time required for resource adaptation from hours/minutes to seconds/milliseconds. With the aforementioned statements, system administrators are able to specify the general operational boundaries in terms of lower and upper system load thresholds, as well as the minimum and maximum amount of resources that can be allocated to the controlled system to overcome any eventual situation, including the natural crossing of such thresholds.

ARCA functional goal is to run autonomously while the performance goal is to keep the resources assigned to the controlled resources as close as possible to the optimum (e.g. 5 % from the optimum) while avoiding service disruption as much as possible, keeping client request discard rate as low as possible (e.g. below 1 %). To achieve both goals, ARCA relies on the Autonomic Computing (AC) paradigm, in the form of interconnected micro-services. Therefore, ARCA includes the four main elements and activities defined by AC, incarnated as:

Collector Is responsible of gathering and formatting the heterogeneous observations that will be used in the control cycle.

Analyzer Correlates the observations to each other in order to find the situation of the controlled system, especially the current load of the resources allocated to the system and the occurrence of an incident that can affect to the normal operation of the system, such as an earthquake that increases the traffic in an emergency-support system, which is the main target scenario studied in this paper.

Decider Determines the necessary actions to adjust the resources to the load of the controlled system.

Enforcer Requests the underlying and overlying infrastructure, such as OpenStack, to make the necessary changes to reflect the effects of the decided actions into the system.

Being a micro-service architecture means that the different components are executed in parallel. This allows such components to operate in two ways. First, their operation can be dispatched by receiving a message from the previous service or an external service. Second, the services can be self-dispatched, so they can activate some action or send some message without being previously stimulated by any message. The overall control process loops indefinitely and it is closed by checking that the expected effects of an action are

actually taking place. The coherence among the distributed services involved in the ARCA control process is ensured by enforcing a common semantic representation and ontology to the messages they exchange.

ARCA semantics are built with the Resource Description Framework (RDF) and the Web Ontology Language (OWL), which are well known and widely used standards for the semantic representation and management of knowledge. They provide the ability to represent new concepts without requiring to change the software, just plugin extensions to the ontology. ARCA stores all its knowledge is stored in the Knowledge Base (KB), which is queried and kept up-to-date by the analyzer and decider micro-services. It is implemented by Apache Jena Fuseki, which is a high-performance RDF data store that supports SPARQL through an HTTP/REST interface. Being de-facto standards, both technologies enable ARCA to be easily integrated to virtualization platforms like OpenStack.

Appendix C. ARCA Integration With ETSI-NFV-MANO

In this section we describe how to fit ARCA on a general SDN/NFV underlying infrastructure and introduce a showcase experiment that demonstrates its operation on an OpenStack-based experimentation platform. We first describe the integration of ARCA with the NFV-MANO reference architecture. We contextualize the significance of this integration by describing an emergency support scenario that clearly benefits from it. Then we proceed to detail the elements forming the OpenStack platform and finally we discuss some initial results obtained from them.

C.1. Functional Integration

The most important functional blocks of the NFV reference architecture promoted by ETSI (see ETSI-NFV-MANO [ETSI-NFV-MANO]) are the system support functions for operations and business (OSS/BSS), the element management (EM) and, obviously. the Virtual Network Functions (VNFs). But these functions cannot exist without being instantiated on a specific infrastructure, the NFV infrastructure (NFVI), and all of them must be coordinated, orchestrated, and managed by the general NFV-MANO functions.

Both the NFVI and the NFV-MANO elements are subdivided into several sub-components. The NFVI has the underlying physical computing, storage, and network resources, which are sliced (see[I-D.qiang-coms-netslicing-information-model] and [I-D.geng-coms-architecture]) and virtualized to conform the virtual computing, storage, and network resources that will host the VNFs. In addition, the NFV-MANO is subdivided in the NFV Orchestrator (NFVO), the VNF manager (VNFM) and the Virtual Infrastructure Manager

(VIM). As their name indicates, all high-level elements and sub-components have their own and very specific objective in the NFV architecture.

During the design of ARCA we enforced both operational and interfacing aspects to its main objectives. From the operational point of view, ARCA processes observations to manage virtual resources, so it plays the role of the VIM mentioned above. Therefore, ARCA has been designed with appropriate interfaces to fit in the place of the VIM. This way, ARCA provides the NFV reference architecture with the ability to react to external events to adapt virtual computer and network systems, even anticipating such adaptations as performed by ARCA itself. However, some interfaces must be extended to fully enable ARCA to perform its work within the NFV architecture.

Once ARCA is placed in the position of the VIM, it enhances the general NFV architecture with its autonomic management capabilities. In particular, it discharges some responsibilities from the VNFM and NFVO, so they can focus on their own business while the virtual resources are behaving as they expect (and request). Moreover, ARCA improves the scalability and reliability of the managed system in case of disconnection from the orchestration layer due to some failure, network split, etc. It is also achieved by the autonomic capabilities, which, as described above, are guided by the rules and policies specified by the administrators and, here, communicated to ARCA through the NFVO. However, ARCA will not be limited to such operation so, more generally, it will accomplish the requirements established by the Virtual Network Operators (VNOs), which are the owners of the slice of virtual resources that is managed by a particular instance of NFV-MANO, and therefore ARCA.

In addition to the operational functions, ARCA incorporates the necessary mechanisms to engage the interfaces that enable it to interact with other elements of the NFV-MANO reference architecture. More specifically, ARCA is bound to the Or-Vi (see ETSI-NFV-IFA-005 [ETSI-NFV-IFA-005]) and the Nf-Vi (see ETSI-NFV-IFA-004 [ETSI-NFV-IFA-004] and ETSI-NFV-IFA-019 [ETSI-NFV-IFA-019]). The former is the point of attachment between the NFVO and the VIM while the latter is the point of attachment between the NFVI and the VIM. In our current design we decided to avoid the support for the point of attachment between the VNFM and the VIM, called Vi-Vnfm (see ETSI-NFV-IFA-006 [ETSI-NFV-IFA-006]). We leave it for future evolutions of the proposed integration, that will be enabled by a possible solution that provides the functions of the VNFM required by ARCA.

Through the Or-Vi, ARCA receives the instructions it will enforce to the virtual computer and network system it is controlling. As

mentioned above, these are specified in the form of rules and policies, which are in turn formatted as several statements and embedded into the Or-Vi messages. In general, these will be high-level objectives, so ARCA will use its reasoning capabilities to translate them into more specific, low-level objectives. For instance, the Or-Vi can specify some high-level statement to avoid CPU overloading and ARCA will use its innate and acquired knowledge to translate it to specific statements that specify which parameters it has to measure (CPU load from assigned servers) and which are their desired boundaries, in the form of high threshold and low threshold. Moreover, the Or-Vi will be used by the NFVO to specify which actions can be used by ARCA to overcome the violation of the mentioned policies.

All information flowing the Or-Vi interface is encoded and formatted by following a simple but highly extensible ontology and exploiting the aforementioned semantic formats. This ensures that the interconnected system is able to evolve, including the replacement of components, updating (addition or removal) the supported concepts to understand new scenarios, and connecting external tools to further enhance the management process. The only requirement to ensure this feature is to ensure that all elements support the mentioned ontology and semantic formats. Although it is not a finished task, the development of semantic technologies allows the easy adaptation and translation of existing information formats, so it is expected that more and more software pieces become easily integrable with the ETSI-NFV-MANO [ETSI-NFV-MANO] architecture.

In contrast to the Or-Vi interface, the Nf-Vi interface exposes more precise and low-level operations. Although this makes it easier to be integrated to ARCA, it also makes it to be tied to specific implementations. In other words, building a proxy that enforces the aforementioned ontology to different interface instances to homogenize them adds undesirable complexity. Therefore, new components have been specifically developed for ARCA to be able to interact with different NFVIs. Nevertheless, this specialization is limited to the collector and enforcer. Moreover, it allows ARCA to have optimized low-level operations, with high improvement of the overall performance. This is the case of the specific implementations of the collector and enforcer used with Mininet and Docker, which are used as underlying infrastructures in previous experiments described in ICIN 2017 [ICIN-2017]. Moreover, as discussed in the following section, this is also the case of the implementations of the collector and enforcer tied to OpenStack telemetry and compute interfaces, respectively. Hence it is important to ensure that telemetry is properly addressed, so we insist in the need to adopt a common framework in such endpoint (see [I-D.song-ntf]).

Although OpenStack still lacks some functionality regarding the construction of specific virtual networks, we use it as the NFVI functional block in the integrated approach. Therefore, OpenStack is the provider of the underlying SDN/NFV infrastructure and we exploited its APIs and SDK to achieve the integration. More specifically, in our showcase we use the APIs provided by Ceilometer, Gnocchi, and Compute services as well as the SDK provided for Python. All of them are gathered within the Nf-Vi interface. Moreover, we have extended the Or-Vi interface to connect external elements, such as the physical or environmental event detectors and Big Data connectors, which is becoming a mandatory requirement of the current virtualization ecosystem and it conforms our main extension to the NFV architecture.

C.2. Target Experiment and Scenario

From the beginning of our work on the design of ARCA we are targeting real-world scenarios, so we get better suited requirements. In particular we work with a scenario that represents an emergency support service that is hosted on a virtual computer and network system, which is in turn hosted on the distributed virtualization infrastructure of a medium-sized organization. The objective is to clearly represent an application that requires high dynamicity and high degree of reliability. The emergency support service accomplishes this by being barely used when there is no incident but also being heavily loaded when there is an incident.

Both the underlying infrastructure and virtual network share the same topology. They have four independent but interconnected network domains that form part of the same administrative domain (organization). The first domain hosts the systems of the headquarters (HQ) of the owner organization, so the VNFs it hosts (servants) implement the emergency support service. We defined them as ``servants'' because they are Virtual Machine (VM) instances that work together to provide a single service by means of backing the Load Balancer (LB) instances deployed in the separate domains. The amount of resources (servants) assigned to the service will be adjusted by ARCA, attaching or detaching servants to meet the load boundaries specified by administrators.

The other domains represent different buildings of the organization and will host the clients that access to the service when an incident occurs. They also host the necessary LB instances, which are also VNFs that are controlled by ARCA to regulate the access of clients to servants. All domains will have physical detectors to provide external information that can (and will) be correlated to the load of the controlled virtual computer and network system and thus will affect to the amount of servants assigned to it. Although the

underlying infrastructure, the servants, and the ARCA instance are the same as those those used in the real world, both clients and detectors will be emulated. Anyway, this does not reduce the transferability of the results obtained from our experiments as it allows to expand the amount of clients beyond the limits of most physical infrastructures.

Each underlying OpenStack domain will be able to host a maximum of 100 clients, as they will be deployed on a low profile virtual machine (flavor in OpenStack). In general, clients will be performing requests at a rate of one request every ten seconds, so there would be a maximum of 30 requests per second. However, under the simulated incident, the clients will raise their load to reach a common maximum of 1200 requests per second. This mimics the shape and size of a real medium-size organization of about 300 users that perform a maximum of four requests per second when they need some support.

The topology of the underlying network is simplified by connecting the four domains to the same, high-performance switch. However, the topology of the virtual network is built by using direct links between the HQ domain and the other three domains. These are complemented by links between domains 2 and 3, and between domains 3 and 4. This way, the three domains have three paths to reach the HQ domain: a direct path with just one hop, and two indirect paths with two and three hops, respectively.

During the execution of the experiment, the detectors notify the incident to the controller as soon as it happens. However, although the clients are stimulated at the same time, there is some delay between the occurrence of the incident and the moment the network service receives the increase in the load. One of the main targets of our experiment is to study such delay and take advantage of it to anticipate the amount of servants required by the system. We discuss it below.

In summary, this scenario highlights the main benefits of ARCA to play the role of VIM and interacting with the underlying OpenStack platform. This means the advancement towards an efficient use of resources and thus reducing the CAPEX of the system. Moreover, as the operation of the system is autonomic, the involvement of human administrators is reduced and, therefore, the OPEX is also reduced.

C.3. OpenStack Platform

The implementation of the scenario described above reflects the requirements of any edge/branch networking infrastructure, which are composed of several distributed micro-data-centers deployed on the

wiring centers of the buildings and/or storeys. We chose to use OpenStack to meet such requirements because it is being widely used in production infrastructures and the resulting infrastructure will have the necessary robustness to accomplish our objectives, at the time it reflects the typical underlying platform found in any SDN/NFV environment.

We have deployed four separate network domains, each one with its own OpenStack instantiation. All domains are totally capable of running regular OpenStack workload, i.e. executing VMs and networks, but, as mentioned above, we designate the domain 1 to be the headquarters of the organization. The different underlying networks required by this (quite complex) deployment are provided by several VLANs within a high-end L2 switch. This switch represents the distributed network of the organization. Four separated VLANs are used to isolate the traffic within each domain, by connecting an interface of OpenStack's controller and compute nodes. These VLANs therefore form the distributed data plane. Moreover, other VLAN is used to carry the control plane as well as the management plane, which are used by the NFV-MANO, and thus ARCA. It is instantiated in the physical machine called ARCA Node, to exchange control and management operations in relation to the collector and enforcer defined in ARCA. This VLAN is shared among all OpenStack domains to implement the global control of the virtualization environment pertaining to the organization. Finally, other VLAN is used by the infrastructure to interconnect the data planes of the separated domains and also to allow all elements of the infrastructure to access the Internet to perform software installation and updates.

Installation of OpenStack is provided by the Red Hat OpenStack Platform, which is tightly dependent on the Linux operating system and closely related to the software developed by the OpenStack Open Source project. It provides a comprehensive way to install the whole platform while being easily customized to meet our specific requirements, while it is also backed by operational quality support.

The ARCA node is also based on Linux but, since it is not directly related to the OpenStack deployment, it is not based on the same distribution. It is just configured to be able to access the control and management interfaces offered by OpenStack, and therefore it is connected to the VLAN that hosts the control and management planes. On this node we deploy the NFV-MANO components, including the micro-services that form an ARCA instance.

In summary, we dedicate nine physical computers to the OpenStack deployment, all are Dell PowerEdge R610 with 2 x Xeon 5670 2.96 GHz (6 core / 12 thread) CPU, 48 GiB RAM, 6 x 146 GiB HD at 10 kRPM, and 4 x 1 GE NIC. Moreover, we dedicate an additional computer with the

same specification to the ARCA Node. We dedicate a less powerful computer to implement the physical router because it will not be involved in the general execution of OpenStack nor in the specific experiments carried out with it. Finally, as detailed above, we dedicate a high-end physical switch, an HP ProCurve 1810G-24, to build the interconnection networks.

C.4. Initial Results

Using the platform described above we execute an initial but long-lasting experiment based on the target scenario introduced at the beginning of this section. The objective of this experiment is twofold. First, we aim to demonstrate how ARCA behaves in a real environment. Second, we aim to stress the coupling points between ARCA and OpenStack, which will raise the limitations of the existing interfaces.

With such objectives in mind, we define a timeline that will be followed by both clients and external event detectors. It forces the virtualized system to experience different situations, including incidents of many severities. When an incident is found in the timeline, the detectors notify it to the ARCA-based VIM and the clients change their request rates, which will depend on the severity of the incident. This behavior is widely discussed in ICIN 2018 [ICIN-2018], remarking how users behave after occurring a disaster or another similar incident.

The ARCA-based VIM will know the occurrence of the incident from two sources. First, it will receive the notification from the event detectors. Second, it will notice the change of the CPU load of the servants assigned to the target service. In this situation, ARCA has different opportunities to overcome the possible overload (or underload) of the system. We explore the anticipation approach deeply discussed in ICIN 2018 [ICIN-2018]. Its operation is enclosed in the analyzer and decider and it is based on an algorithm that is divided in two sub-algorithms.

The first sub-algorithm reacts to the detection of the incident and ulterior correlation of its severity to the amount of servants required by the system. This sub-algorithm hosts the regression of the learner, which is based on the SVM/SVR technique, and predicts the necessary resources from two features: the severity of the incident and the time elapsed from the moment it happened. The resulting amount of servants is established as the minimum amount that the VIM can use.

The second sub-algorithm is fed with the CPU load measurements of the servants assigned to the service, as reported by the OpenStack

platform. With this information it checks whether the system is within the operating parameters established by the NFVO. If not, it adjusts the resources assigned to the system. It also uses the minimum amount established by the other sub-algorithm as the basis for the assignation. After every correction, this algorithm learns the behavior by adding new correlation vectors to the SVM/SVR structure.

When the experiment is running, the collector component of the ARCA-based VIM is attached to the telemetry interface of OpenStack by using the SDK to access the measurement data generated by Ceilometer and stored by Gnocchi. In addition, it is attached to the external event detectors in order to receive their notifications. On the other hand, the enforcer component is attached to the Compute interface of OpenStack by also using its SDK to request the infrastructure to create, destroy, query, or change the status of a VM that hosts a servant of the controlled system. Finally, the enforcer also updates the lists of servers used by the load balancers to distribute the clients among the available resources.

During the execution of the experiment we make the ARCA-based VIM to report the severity of the last incident, if any, the time elapsed since it occurred, the amount of servants assigned to the controlled system, the minimum amount of servants to be assigned, as determined by the anticipation algorithm, and the average load of all servants. In this instance, the severities are spread between 0 (no incident) and 4 (strongest incident), the elapsed times are less than 35 seconds, and the minimum server assignation (MSA) is below 10, although the hard maximum is 15.

With such measurements we illustrate how the learned correlation of the three features (dimensions) mentioned above is achieved. Thus, when there is no incident (severity = 0), the MSA is kept to the minimum. In parallel, regardless of the severity level, the algorithm learned that there is no need to increase the MSA for the first 5 or 10 seconds. This shows the behavior discussed in this paper, that there is a delay between the occurrence of an event and the actual need for updated amount of resources, and it forms one fundamental aspect of our research.

By inspecting the results, we know that there is a burst of client demands that is centered (peak) around 15 seconds after the occurrence of an incident or any other change in the accounted severity. We also know that the burst lasts longer for higher severities, and it fluctuates a bit for the highest severities. Finally, we can also notice that for the majority of severities, the increased MSA is no longer required after 25 seconds from the time the severity change was notified.

All that information becomes part of the knowledge of ARCA and it is stored both by the internal structures of the SVM/SVR and, once represented semantically, in the semantic database that manages the knowledge base of ARCA. Thus, it is used to predict any future behavior. For instance, is an incident of severity 3 has occurred 10 seconds ago, ARCA knows that it will need to set the MSA to 6 servants. In fact, this information has been used during the experiment, so we can also know the accuracy of the algorithm by comparing the anticipated MSA value with the required value (or even the best value). However, the analysis of such information is left for the future.

While preparing and executing the experiment we found several limitation intrinsic to the current OpenStack platform. First, regardless of the CPU and memory resources assigned to the underlying controller nodes, the platform is unable to record and deliver performance measurements at a lower interval than every 10 seconds, so it is currently not suitable for real time operations, which is important for our long-term research objectives. Moreover, we found that the time required by the infrastructure to create a server that hosts a somewhat heavy servant is around 10 seconds, which is too far from our targets. Although these limitations can be improved in the future, they clearly justify that our anticipation approach is essential for the proper working of a virtual system and, thus, the integration of external information becomes mandatory for future system management technologies, especially considering the virtualization environments.

Finally, we found it difficult for the required measurements to be pushed to external components, so we had to poll for them. Otherwise, some component of ARCA must be instantiated along the main OpenStack components and services so it has first-hand and prompt access to such features. This way, ARCA could receive push notifications with the measurements, as it is for the external detectors. This is a key aspect that affects the placement of the NFV-VIM, or some subpart of it, on the general architecture. Therefore, for future iterations of the NFV reference architecture, an integrated view between the VIM and the NFVI could be required to reflect the future reality.

Authors' Addresses

Pedro Martinez-Julia (editor)
NICT
4-2-1, Nukui-Kitamachi
Koganei, Tokyo 184-8795
Japan

Phone: +81 42 327 7293
Email: pedro@nict.go.jp

Shunsuke Homma
NTT
Japan

Email: shunsuke.homma.fp@hco.ntt.co.jp