

Network Working Group  
Internet-Draft  
Intended status: Informational  
Expires: May 7, 2020

A. Clemm  
Futurewei  
L. Ciavaglia  
Nokia  
L. Granville  
Federal University of Rio Grande do Sul (UFRGS)  
J. Tantsura  
Apstra, Inc.  
November 4, 2019

Intent-Based Networking - Concepts and Overview  
draft-clemm-nmrg-dist-intent-03

Abstract

Intent and Intent-Based Networking are taking the industry by storm. At the same time, those terms are used loosely and often inconsistently, in many cases overlapping and confused with other concepts such as "policy". This document is intended to clarify the concept of "Intent" and provide an overview of functionality that associated with it. The goal is to contribute towards a common and shared understanding of terms, concepts, and functionality which can be used as foundation to guide further definition of associated research and engineering problems and their solutions.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 7, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. Key Words . . . . .	4
3. Definitions and Acronyms . . . . .	4
4. Introduction of Concepts . . . . .	5
4.1. Intent and Intent-Based Management . . . . .	5
4.2. Related Concepts . . . . .	6
4.2.1. Service Models . . . . .	6
4.2.2. Policy and Policy-Based Management . . . . .	8
4.2.3. Distinguishing between Intent, Policy, and Service Models . . . . .	10
5. Principles . . . . .	11
6. Lifecycle . . . . .	14
7. Intent-Based Networking - Functionality . . . . .	18
7.1. Intent Fulfillment . . . . .	18
7.2. Intent Assurance . . . . .	18
8. Items for Discussion . . . . .	19
9. IANA Considerations . . . . .	19
10. Security Considerations . . . . .	19
11. References . . . . .	19
11.1. Normative References . . . . .	19
11.2. Informative References . . . . .	19
Authors' Addresses . . . . .	21

## 1. Introduction

Traditionally in the IETF, interest with regard to management and operations has focused on individual network and device features. Standardization emphasis has generally been put on management instrumentation that needed to be provided to a networking device. A prime example for this is SNMP-based management and the 200+ MIBs that have been defined by the IETF over the years. More recent examples include YANG data model definitions for aspects such as interface configuration, ACL configuration, or Syslog configuration.

There is a sense and reality that in modern network environments managing networks by configuring myriads of "nerd knobs" on a device-

by-device basis is no longer sustainable. Big challenges arise with keeping device configurations not only consistent across a network, but consistent with the needs of services and service features they are supposed to enable. Adoptability to changes at scale is a fundamental property of a well designed IBN system, that requires ability to consume and process analytics that are context/intent aware at near real time speeds. At the same time, operations need to be streamlined and automated wherever possible to not only lower operational expenses, but allow for rapid reconfiguration of networks at sub-second time scales and to ensure networks are delivering their functionality as expected.

Accordingly, IETF has begun to address end-to-end management aspects that go beyond the realm of individual devices in isolation. Examples include the definition of YANG models for network topology [RFC8345] or the introduction of service models used by service orchestration systems and controllers [RFC8309]. In addition, a lot of interest has been fueled by the discussion about how to manage autonomic networks as discussed in the ANIMA working group. Autonomic networks are driven by the desire to lower operational expenses and make management of the network as a whole exceptionally easy, putting it at odds with the need to manage the network one device and one feature at a time. However, while autonomic networks are intended to exhibit "self-management" properties, they still require input from an operator or outside system to provide operational guidance and information about the goals, purposes, and service instances that the network is to serve.

This vision has since caught on with the industry in a big way, leading to a significant number solutions that offer "intent-based management" that promise network providers to manage networks holistically at a higher level of abstraction and as a system that happens to consist of interconnected components, as opposed to a set of independent devices (that happen to be interconnected). Those offerings include IBN systems (offering full lifecycle of intent), SDN controllers (offering a single point of control and administration for a network) as well as network management and Operations Support Systems (OSS).

However, it has been recognized for a long time that comprehensive management solutions cannot operate only at the level of individual devices and low-level configurations. In this sense, the vision of "intent" is not entirely new. In the past, ITU-T's model of a Telecommunications Management Network, TMN, introduced a set of management layers that defined a management hierarchy, consisting of network element, network, service, and business management. High-level operational objectives would propagate in top-down fashion from upper to lower layers. The associated abstraction hierarchy was key

to decompose management complexity into separate areas of concerns. This abstraction hierarchy was accompanied by an information hierarchy that concerned itself at the lowest level with device-specific information, but that would, at higher layers, include, for example, end-to-end service instances. Similarly, the concept of "policy-based management" has for a long time touted the ability to allow users to manage networks by specifying high-level management policies, with policy systems automatically "rendering" those policies, i.e. breaking them down into low-level configurations and control logic.

What has been missing, however, is putting these concepts into a more current context and updating it to account for current technology trends. This document attempts to clarify the concepts behind intent. It differentiates it from related concepts. It also provides an overview of first-order principles of Intent-Based Networking as well as associated functionality. In addition, a number of research challenges are highlighted. The goal is to contribute to a common and shared understanding that can be used as a foundation to articulate research and engineering problems in the area of Intent-Based Networking.

## 2. Key Words

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

## 3. Definitions and Acronyms

ACL: Access Control List

Intent: An abstracted, declarative and vendor agnostic set of rules used to provide full lifecycle (Design/Build/Deploy/Validate) to a network and services it provides.

Policy: A rule, or set of rules, that governs the choices in behavior of a system.

SSoT: Single Source of Truth - A functional block in an IBN system that normalizes user' intent and serves as the single source of data for the lower layers.

IBA: Intent Based Analytics - Analytics that are defined and derived from user' intent and used to validate the intended state.

IBS: Intent Based System.

PDP: Policy Decision Point

PEP: Policy Enforcement Point

Service Model: A model that represents a service that is provided by a network to a user.

#### 4. Introduction of Concepts

The following section provides an overview of the concept of intent respectively intent-based management. It also provides an overview of the related concepts of service models, and of policies respectively policy-based management, and explains how they relate to intent and intent-based management.

##### 4.1. Intent and Intent-Based Management

In the context of Autonomic Networks, Intent is defined as "an abstract, high-level policy used to operate a network" [RFC7575]. According to this definition, an intent is a specific type of policy. However, to avoid using "intent" simply as a synonym for "policy, a clearer distinction needs to be introduced that distinguishes intent clearly from other types of policies.

For one, while Intent-Based Management clearly aims to lead towards networks that are dramatically simpler to manage and operate requiring only minimal outside intervention, the concept of "intent" is not limited to autonomic networks, but applies to any network. Networks, even when considered "autonomic", are not clairvoyant and have no way of automatically knowing particular operational goals nor what instances of networking services to support. In other words, they do not know what the "intent" of the network provider is that gives the network the purpose of its being. This still needs to be communicated by what informally constitutes "intent".

More specifically, intent is a declaration of operational goals that a network should meet and outcomes that the network is supposed to deliver, without specifying how to achieve them. Those goals and outcomes are defined in a manner that is purely declarative - they specify what to accomplish, not how to achieve it. "Intent" thus applies several important concepts simultaneously:

- o It provides data abstraction: Users and operators do not need to be concerned with low-level device configuration and nerd knobs.

- o It provides functional abstraction from particular management and control logic: Users and operators do not need to be concerned even with how to achieve a given intent. What is specified is a desired outcome, with the intent-based system automatically figuring out a course of action (e.g. a set of rules, an algorithm) for how to achieve the outcome.

In an autonomic network, intent should be rendered by the network itself, i.e. translated into device-specific rules and courses of action. Ideally, it should not even be orchestrated or broken down by a higher-level, centralized system, but by the network devices themselves using a combination of distributed algorithms and local device abstraction. Because intent holds for the network as a whole, not individual devices, it needs to be automatically disseminated across all devices in the network, which can themselves decide whether they need to act on it. This facilitates management even further, since it obviates the need for a higher-layer system to break down and decompose higher-level intent, and because there is no need to even discover and maintain an inventory of the network to be able to manage it.

Tentative definition for intent-based networks Networks configuring and adapting autonomously to the user or operator intentions (i.e., a desired state or behavior) without the need to specify every technical detail of the process and operations to achieve it (i.e., the "machines" will figure out on their own how to realize the user goal).

Other definitions of intent exist such as [TR523] and will be investigated in future revisions of this document. Likewise, some definitions of intent allow for the presence of a centralized function that renders the intent into lower-level policies or instructions and orchestrates them across the network. While to the end user the concept of "intent" appears the same regardless of its method of rendering, this interpretation opens a slippery slope of how to clearly distinguish "intent" from other higher-layer abstractions. Again, these notions will be further investigated in future revisions of this document and in collaboration with NMRG.

## 4.2. Related Concepts

### 4.2.1. Service Models

A service model is a model that represents a service that is provided by a network to a user. Per [RFC8309], a service model describes a service and its parameters in a portable/vendor agnostic way that can be used independent of the equipment and operating environment on which the service is realized. Two subcategories are distinguished:

a "Customer Service Model" describes an instance of a service as provided to a customer, possibly associated with a service order. A "Service Delivery Model" describes how a service is instantiated over existing networking infrastructure.

An example of a service could be a Layer 3 VPN service [RFC8299], a Network Slice, or residential Internet access. Service models represent service instances as entities in their own right. Services have their own parameters, actions, and lifecycles. Typically, service instances can be bound to end users, who might be billed for the service.

Instantiating a service typically involves multiple aspects:

- o A user (or northbound system) needs to define and/or request a service to be instantiated.
- o Resources need to be allocated, such as IP addresses, AS numbers, VLAN or VxLAN pools, interfaces, bandwidth, or memory.
- o How to map services to the resources needs to be defined. Multiple mappings are often possible, which to select may depend on context (such as which type of access is available to connect the end user with the service).
- o [I-D.ietf-teas-te-service-mapping-yang] is an example of such mapping - a data model to map customer service models (e.g., the L3VPM Service Model) to Traffic Engineering (TE) models (e.g., the TE Tunnel or the Abstraction and Control of Traffic Engineered Networks Virtual Network model)
- o Bindings need to be maintained between upper and lower-level objects.
- o Once instantiated, the service needs to be validated and assured to ensure that the network indeed delivers the service as requested.

They involve a system, such as a controller, that provides provisioning logic. Orchestration itself is generally conducted using a "push" model, in which the controller/manager initiates the operations as required, pushing down the specific configurations to the device. (In addition to instantiating and creating new instances of a service, updating, modifying, and decommissioning services need to be also supported.) The device itself typically remains agnostic to the service or the fact that its resources or configurations are part of a service/concept at a higher layer.

Instantiated service models map to instantiated lower-layer network and device models. Examples include instances of paths, or instances of specific port configurations. The service model typically also models dependencies and layering of services over lower-layer networking resources that are used to provide services. This facilitates management by allowing to follow dependencies for troubleshooting activities, to perform impact analysis in which events in the network are assessed regarding their impact on services and customers. Services are typically orchestrated and provisioned top-to-bottom, which also facilitates keeping track of the assignment of network resources. Service models might also be associated with other data that does not concern the network but provides business context. This includes things such as customer data (such as billing information), service orders and service catalogues, tariffs, service contracts, and Service Level Agreements (SLAs) including contractual agreements regarding remediation actions.

Like intent, service models provide higher layers of abstraction. Service models are often also complemented with mappings that capture dependencies between service and device or network configurations. Unlike intent, service models do not allow to define a desired "outcome" that would be automatically maintained by the intent system. Instead, management of service models requires development of sophisticated algorithms and control logic by network providers or system integrators.

#### 4.2.2. Policy and Policy-Based Management

Policy-based management (PBM) is a management paradigm that separates the rules that govern the behavior of a system from the functionality of the system. It promises to reduce maintenance costs of information and communication systems while improving flexibility and runtime adaptability. It is present today at the heart of a multitude of management architectures and paradigms including SLA-driven, Business-driven, autonomous, adaptive, and self-\* management [Boutaba07]. The interested reader is asked to refer to the rich set of existing literature which includes this and many other references. In the following, we will only provide a much-abridged and distilled overview.

At the heart of policy-based management is the concept of a policy. Multiple definitions of policy exist: "Policies are rules governing the choices in behavior of a system" [Sloman94]. "Policy is a set of rules that are used to manage and control the changing and/or maintaining of the state of one or more managed objects" [Strassner03]. Common to most definitions is the definition of a policy as a "rule". Typically, the definition of a rule consists of an event (whose occurrence triggers a rule), a set of conditions

(that get assessed and that must be true before any actions are actually "fired"), and finally a set of one or more actions that are carried out when the condition holds.

Policy-based management can be considered an imperative management paradigm: Policies specify precisely what needs to be done when and in which circumstance. Using policies, management can in effect be defined as a set of simple control loops. This makes policy-based management a suitable technology to implement autonomic behavior that can exhibit self-\* management properties including self-configuration, self-healing, self-optimization, and self-protection. In effect, policies define management as a set of simple control loops.

Policies typically involve a certain degree of abstraction in order to cope with heterogeneity of networking devices. Rather than having a device-specific policy that defines events, conditions, and actions in terms of device-specific commands, parameters, and data models, policy is defined at a higher-level of abstraction involving a canonical model of systems and devices to which the policy is to be applied. A policy agent on a controller or the device subsequently "renders" the policy, i.e., translates the canonical model into a device-specific representation. This concept allows to apply the same policy across a wide range of devices without needing to define multiple variants. In other words - policy definition is de-coupled from policy instantiation and policy enforcement. This enables operational scale and allows network operators and authors of policies to think in higher terms of abstraction than device specifics and be able to reuse the same, high level definition definition across different networking domains, WAN, DC or public cloud.

Policy-based management is typically "push-based": Policies are pushed onto devices where they are rendered and enforced. The push operations are conducted by a manager or controller, which is responsible for deploying policies across the network and monitor their proper operation. That said, other policy architectures are possible. For example, policy-based management can also include a pull-component in which the decision regarding which action to take is delegated to a so-called Policy Decision Point (PDP). This PDP can reside outside the managed device itself and has typically global visibility and context with which to make policy decisions. Whenever a network device observes an event that is associated with a policy, but lacks the full definition of the policy or the ability to reach a conclusion regarding the expected action, it reaches out to the PDP for a decision (reached, for example, by deciding on an action based on various conditions). Subsequently, the device carries out the decision as returned by the PDP - the device "enforces" the policy

and hence acts as a PEP (Policy Enforcement Point). Either way, PBM architectures typically involve a central component from which policies are deployed across the network, and/or policy decisions served.

Like Intent, policies provide a higher layer of abstraction. Policy systems are also able to capture dynamic aspects of the system under management through specification of rules that allow to define various triggers for certain courses of actions. Unlike intent, the definition of those rules (and courses of actions) still needs to be articulated by users. Since the intent is unknown, conflict resolution within or between policies requires interactions with a user or some kind of logic that resides outside of PBM.

#### 4.2.3. Distinguishing between Intent, Policy, and Service Models

What Intent, Policy, and Service Models all have in common is the fact that they involve a higher-layer of abstraction of a network that does not involve device-specifics, that generally transcends individual devices, and that makes the network easier to manage for applications and human users compared to having to manage the network one device at a time. Beyond that, differences emerge. Service models have less in common with policy and intent than policy and intent do with each other.

Summarized differences:

- o A service model is a data model that is used to describe instances of services that are provided to customers. A service model has dependencies on lower level models (device and network models) when describing how the service is mapped onto underlying network and IT infrastructure. Instantiating a service model requires orchestration by a system; the logic for how to orchestrate/manage/provide the service model, and how to map it onto underlying resources, is not included as part of the model itself.
- o Policy is a set of rules, typically modeled around a variation of events/conditions/actions, used to express simple control loops that can be rendered by devices themselves, without requiring intervention by outside system. Policy lets users define what to do under what circumstances, but it does not specify a desired outcome.
- o Intent is a higher-level declarative policy that operates at the level of a network and services it provides, not individual devices. It is used to define outcomes and high-level operational goals, without the need to enumerate specific events, conditions,

and actions. Which algorithm or rules to apply can be automatically "learned/derived from intent" by the intent system. In the context of autonomic networking, ideally, intent is rendered by the network itself; also the dissemination of intent across the network and any required coordination between nodes is resolved by the network itself without the need for outside systems.

One analogy to capture the difference between policy and intent systems is that of Expert Systems and Learning Systems in the field of Artificial Intelligence. Expert Systems operate on knowledge bases with rules that are supplied by users. They are able to make automatic inferences based on those rules, but are not able to "learn" on their own. Learning Systems (popularized by deep learning and neural networks), on the other hand, are able to learn without depending on user programming. However, they do require a learning or training phase and explanations of actions that the system actually takes provide a different set of challenges.

## 5. Principles

The following operating principles allow characterizing the intent-based/-driven/-defined nature of a system.

1. Single Source of Truth (SSoT) and Single Version/View of Truth (SVoT). The SSoT is an essential component of an intent-based system as it enables several important operations. The set of validated intent expressions is the system's SSoT. SSoT and the records of the operational states enable comparing the intended state and actual state of the system and determining drift between them. SSoT and the drift information provide the basis for corrective actions. If the intent-based is equipped with prediction capabilities or means, it can further develop strategies to anticipate, plan and pro-actively act on the diverging trends with the aim to minimize their impact. Beyond providing a means for consistent system operation, SSoT also allows for better traceability to validate if/how the initial intent and associated business goals have been properly met, to evaluate the impacts of changes in the intent parameters and impacts and effects of the events occurring in the system. Single Version (or View) of Truth derives from the SSoT and can be used to perform other operations such as query, poll or filter the measured and correlated information to create so-called "views". These views can serve the operators and/or the users of the intent-based system. To create intents as single sources of truth, the intent-based system must follow well-specified and well-documented processes and models. In other contexts

[Lenrow15], SSoT is also referred to as the invariance of the intent.

2. One touch but not one shot. In an ideal intent-based system, the user expresses its intents in one form or another and then the system takes over all subsequent operations (one touch). A zero-touch approach could also be imagined in case where the intent-based system has the capabilities or means to recognize intentions in any form of data. However, the zero- or one-touch approach should not be mistaken the fact that reaching the state of a well-formed and valid intent expression is not a one-shot process. On the contrary, the interfacing between the user and the intent-based system could be designed as an interactive and interactive process. Depending on the level of abstraction, the intent expressions will initially contain more or less implicit parts, and unprecise or unknown parameters and constraints. The role of the intent-based system is to parse, understand and refine the intent expression to reach a well-formed and valid intent expression that can be further used by the system for the fulfillment and assurance operations. An intent refinement process could use a combination of iterative steps involving the user to validate the proposed refined intent and to ask the user for clarifications in case some parameters or variables could not be deduced or learned by the means of the system itself. In addition, the Intent-Based System will need to moderate between conflicting intent, helping users to properly choose between intent alternatives that may have different ramifications.
3. Autonomy and Oversight. A desirable goal for an intent-based system is to offer a high degree of flexibility and freedom on both the user side and system side, e.g. by giving the user the ability to express intents using its own terms, by supporting different forms of expression of intents and being capable of refining the intent expressions to well-formed and exploitable expressions. The dual principle of autonomy and oversight allows to operate a system that will have the necessary levels of autonomy to conduct its tasks and operations without requiring intervention of the user and taking its own decisions (within its areas of concern and span of control) as how to perform and meet the user expiations in terms of performance and quality, while at the same time providing the proper level of oversight to satisfy the user requirements for reporting and escalation of relevant information. to be added: description for feedback, reporting, guarantee scope (check points, guard rails, dynamically provisioned, context rich, regular operation vs. exception/ abnormal, information zoom in-out, and link to SVoT. Accountable for decisions and efficiency, late binding (leave it to the

system where to place functionality, how to accomplish certain goals).

4. Learning. An intent-based system is a learning system. By contrast to imperative type of system, such as Event-Condition-Action policy rules, where the user define beforehand the expected behavior of the system to various event and conditions, in an intent-based system, the user only declare what the system should achieve and not how to achieve these goals. There is thus a transfer of reasoning/rationality from the human (domain knowledge) to the system. This transfer of cognitive capability implies also the availability in the intent-based system of capabilities or means for learning, reasoning and knowledge representation and management. The learning abilities of an intent-based systems can apply to different tasks such as optimization of the intent rendering or intent refinement processes. The fact that an intent-based system is a continuously evolving system creates the condition for continuous learning and optimization. Other cognitive capabilities such as planning can also be leveraged in an intent-based system to anticipate or forecast future system state and response to changes in intents or network conditions and thus elaboration of plans to accommodate the changes while preserving system stability and efficiency in a trade-off with cost and robustness of operations. Cope with unawareness of users (smart recommendations).
5. Explainability. Need expressive network capabilities, requirements and constraints to be able to compose/decompose intents, map user's expectation to system capabilities. capability exposure. not just automation of steps that need to be taken, but of bridging the semantic gap between "intent" and actionable levels of instructions Context: multi providers, need discovery and semantic descriptions Explainability: why is a network doing what it is doing
6. Abstraction - users do not need to be concerned with how intent is achieved

Additional principles will be described in future revision of this document addressing aspects such as: Target groups not individual devices, agnostic to implementation details, user-friendly, user vocabulary vs. language of the device/network, explainability, validation and troubleshooting, how to resolve and point out conflicts (between intents), reconcile the reality of what is possible with the fiction of what the user would want, "moderate", awareness of operating within system boundaries, outcome-driven

((what not how, for the user); (what and how/where, for the operator).not imperative/instruction based.)).

The above principles will be further used to understand implications on the design of intent-based systems and their supporting architecture, and derive functional and operational requirements.

## 6. Lifecycle

Intent is subject to a lifecycle: it comes into being, may undergo changes over the course of time, and may at some point be retracted. This lifecycle is closely tied to various interconnection functions that are associated with the intent concept.

Figure 1 depicts an intent lifecycle and its main functions. The functions are divided into two functional (horizontal) planes and into three (vertical) spaces.

The functional planes provide structure for the main functional concerns that are associated with intent: how to fulfill intent, and how to assure it.

- o Fulfillment is concerned with the functions that take intent from its origination by a user (generally, an administrator of the responsible organization) to its realization in the network. This includes:
  - \* Functions that recognize intent from interaction with the user and functions that allow users to refine their intent and articulate it in such ways so that it becomes actionable by an Intent-Based System. Those functions can involve unconventional human-machine interactions, in which a human will not simply give simple commands, but which may involve a human-machine dialog to provide clarifications, to explain ramifications and tradeoffs, and to facilitate refinements.
  - \* Functions that translate user intent into courses of actions and requests to take against the network, which will be meaningful to network configuration and provisioning systems. Possibly, this includes learning functions and algorithms that optimize the courses of actions to take in order to result in the best outcomes, specifically in cases where multiple ways of achieving those outcomes are conceivable.
  - \* Functions that perform and orchestrate the configuration and provisioning steps that were determined by the previous intent translation step.

- o Assurance is concerned with the functions that are necessary ensure that the network indeed complies with the desired intent once it has been fulfilled. This includes:
  - \* Functions that monitor and observe the network and its exhibited behavior.
  - \* Functions that assess and validate whether the observation indicate compliance with intent. This can include functions that perform analysis and aggregation of raw observation data.
  - \* Functions that trigger corrective action as needed.
  - \* Functions that abstract the observations and analysis results in a way that makes it possible for users to relate them to intent. In many cases, lower-level concepts such as detailed performance statistics and observations related to low-level settings need to be "up-leveled" to concepts the user can relate to and take action on.
  - \* Functions that report intent compliance status and that provide adequate summarization and visualization to the user.

The spaces indicate the different perspectives and interactions with different roles that are involved to address the functions:

- o The user space involves the functions that interface the network and intent-based system with the human user. It involves the functions that allow users to articulate and the intent-based system to recognize that intent. It also involves the functions that report back the status of the network relative to the intent and that allow users to assess whether their intent is having the desired effect.
- o The translation or Intent-Based System (IBS) space involves the functions that bridge the gap between intent users and network operations. This includes the functions used to translate an intent into a course of action, the algorithms used to plan and optimize those courses of actions also in consideration of feedback, the functions to analyze and abstract observations to validate compliance with intent and take corrective actions as necessary.
- o The Network Operations space, finally, involves the traditional orchestration, configuration, monitoring, and measurement functions which are used to effectuate the rendered intent and observe its effects on the network.

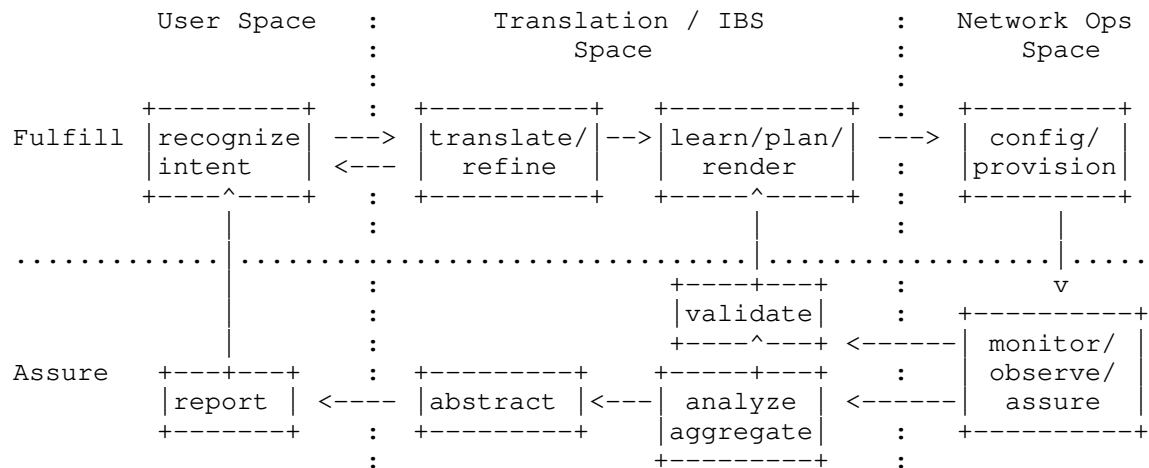


Figure 1: Intent Lifecycle

When inspecting the diagram carefully, it become apparent that the intent lifecycle in fact involves two cycles, or loops:

- o The "inner" intent control loop between IBS and Network Operations space is completely automated and does not involve any human in the loop. It involves automatic analysis and validation of intent based on observations from the network operations space, and feeding those observations into the function that plans the rendering of networking intent in order to make adjustments as needed in the configuration of the network.
- o The "outer" intent control loop involves also the user space and includes the user taking action and adjusting their intent based on feedback from the IBS.

Slight alternatives in intent lifecycles and the functions involved are conceivable. Figure 2 depicts one such alternative with an emphasis in intent fulfilment. (Todo: Intent attributes, intent states. Distinguish flow from users to network, and from network to user.)

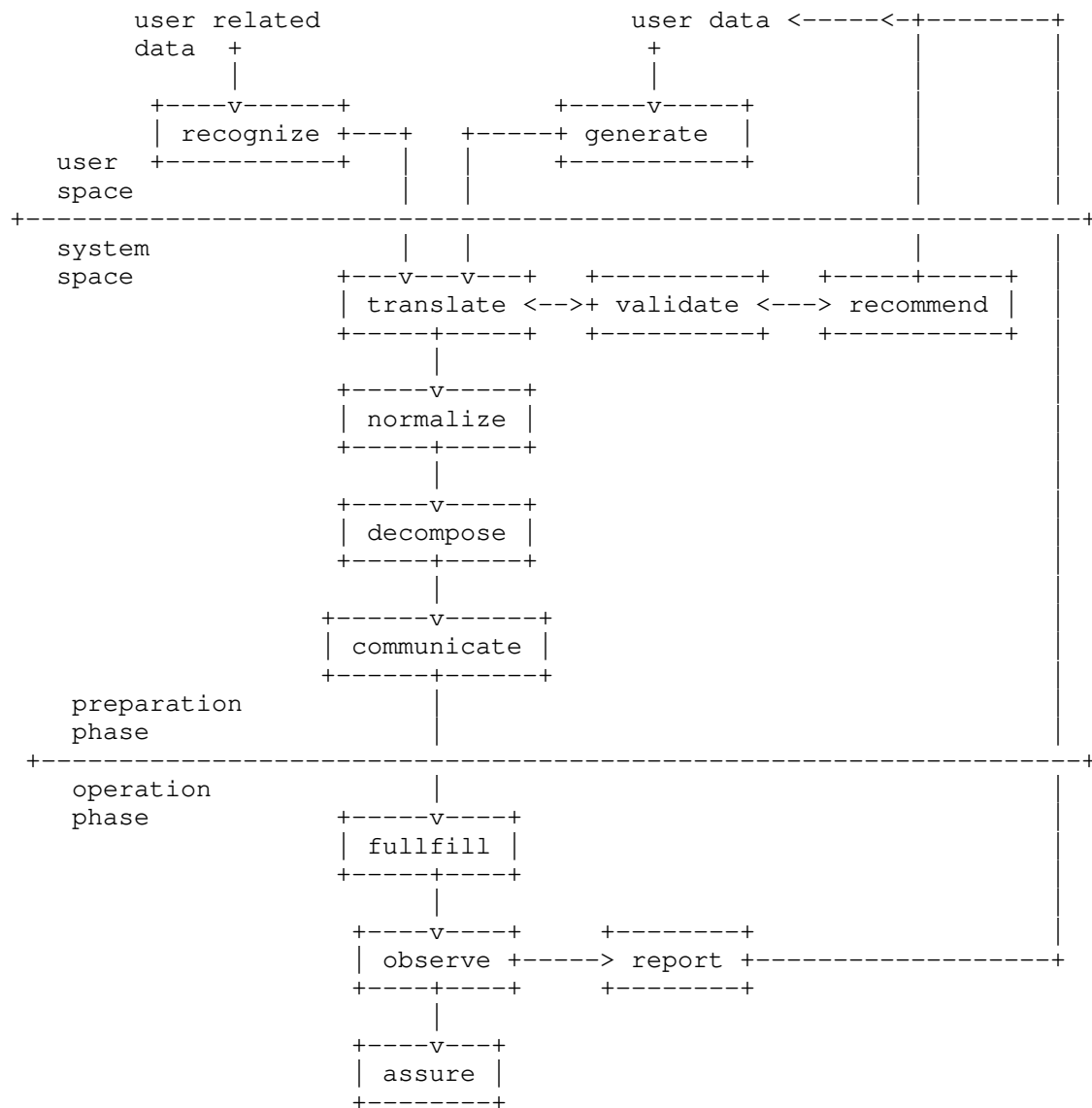


Figure 2: Intent Lifecycle (alt.)

## 7. Intent-Based Networking - Functionality

Intent-Based Networking involves a wide variety of functions which can be roughly divided into two categories:

- o Intent Fulfillment provides functions and interfaces that allow users to communicate intent to the network, and that orchestrates the intent, i.e. that breaks down intent abstractions into lower-level network and device abstractions and performs or coordinates the configuration operations across the network.
- o Intent Assurance provides functions and interfaces that allow users to validate and monitor that the network is indeed adhering to and complying with intent. Control plane or lower-level management operations can cause behavior that inadvertently conflicts with intent which was orchestrated earlier. Accordingly, "intent drift" may occur. Network operators need to be able to detect when such drift occurs, or is about to occur, and be provided with the necessary functions to resolve such conflicts. This can occur by either bringing the network back into compliance, or by articulating modifications to the original intent to moderate between conflicting interests.

The following sections provide a more comprehensive overview of those functions.

### 7.1. Intent Fulfillment

RBD

### 7.2. Intent Assurance

Ability to reason about system' state by employing closed-loop validation in the presence of an inevitable change is a fundamental property of an Intent Assurance part of an IBN system. Since service expectations are created during intent consumption and modeling phase, closed-loop intent validation should start immediately, with the service instantiation. Telemetry consumed could then be enriched with an additional context and must always be processed in context of the Intent it has been instantiated. Direct relationship between the Intent and telemetry gathered enables correlation between changes in states and the Intent and provides contextual base for reasoning about the changes.

## 8. Items for Discussion

Arguably, given the popularity of the term intent, its use could be broadened to encompass also known concepts ("intent-washing"). For example, it is conceivable to introduce intent-based terms for various concepts that, although already known, are related to the context of intent. Each of those terms could then designate an intent subcategory, for example:

- o Operational Intent: defines intent related to operational goals of an operator; corresponds to the original "intent" term.
- o Rule Intent: a synonym for policy rules regarding what to do when certain events occur.
- o Service intent: a synonym for customer service model [RFC8309].
- o Flow Intent: A synonym for a Service Level Objective for a given flow.

Whether to do so is an item for discussion by the Research Group.

## 9. IANA Considerations

Not applicable

## 10. Security Considerations

Not applicable

## 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>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

### 11.2. Informative References

- [Boutaba07] Boutaba, R. and I. Aib, "Policy-Based Management: A Historical perspective. Journal of Network and Systems Management (JNSM), Springer, Vol. 15 (4).", December 2007.
- [eTOM] TMForum, "GB 921 Business Process Framework, Release 17.0.1.", February 2018.
- [I-D.ietf-teas-te-service-mapping-yang] Lee, Y., Dhody, D., Fioccola, G., WU, Q., Ceccarelli, D., and J. Tantsura, "Traffic Engineering (TE) and Service Mapping Yang Model", draft-ietf-teas-te-service-mapping-yang-02 (work in progress), September 2019.
- [Lenrow15] Lenrow, D., "Intent As The Common Interface to Network Resources, Intent Based Network Summit 2015 ONF Boulder: IntentNBI", February 2015.
- [RFC7575] Behringer, M., Pritikin, M., Bjarnason, S., Clemm, A., Carpenter, B., Jiang, S., and L. Ciavaglia, "Autonomic Networking: Definitions and Design Goals", RFC 7575, DOI 10.17487/RFC7575, June 2015, <<https://www.rfc-editor.org/info/rfc7575>>.
- [RFC8299] Wu, Q., Ed., Litkowski, S., Tomotaki, L., and K. Ogaki, "YANG Data Model for L3VPN Service Delivery", RFC 8299, DOI 10.17487/RFC8299, January 2018, <<https://www.rfc-editor.org/info/rfc8299>>.
- [RFC8309] Wu, Q., Liu, W., and A. Farrel, "Service Models Explained", RFC 8309, DOI 10.17487/RFC8309, January 2018, <<https://www.rfc-editor.org/info/rfc8309>>.
- [RFC8345] Clemm, A., Medved, J., Varga, R., Bahadur, N., Ananthakrishnan, H., and X. Liu, "A YANG Data Model for Network Topologies", RFC 8345, DOI 10.17487/RFC8345, March 2018, <<https://www.rfc-editor.org/info/rfc8345>>.
- [Sloman94] Sloman, M., "Policy Driven Management for Distributed Systems. Journal of Network and Systems Management (JNSM), Springer, Vol. 2 (4).", December 1994.
- [Strassner03] Strassner, J., "Policy-Based Network Management. Elsevier.", 2003.

[TR523] Foundation, O. N., "Intent NBI - Definition and Principles. ONF TR-523.", October 2016.

Authors' Addresses

Alexander Clemm  
Futurewei  
2330 Central Expressway  
Santa Clara, CA 95050  
USA

Email: ludwig@clemm.org

Laurent Ciavaglia  
Nokia  
Route de Villejust  
Nozay 91460  
FR

Email: laurent.ciavaglia@nokia.com

Lisandro Zambenedetti Granville  
Federal University of Rio Grande do Sul (UFRGS)  
Av. Bento Goncalves  
Porto Alegre 9500  
BR

Email: granville@inf.ufrgs.br

Jeff Tantsura  
Apstra, Inc.

Email: jefftant.ietf@gmail.com

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.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 7, 2020.

## Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
2. Terminology . . . . .	4
3. Background . . . . .	4
3.1. Virtual Computer and Network Systems . . . . .	4
3.2. SDN and NFV . . . . .	4
3.3. Management and Control . . . . .	5
3.4. Slice Gateway (SLG) . . . . .	5
4. Applying AI to Network Management . . . . .	6
4.1. Beyond Machine Learning . . . . .	6
4.2. Briefing Artificial Intelligence . . . . .	6
5. Extended Management Operation . . . . .	7
5.1. Intelligent Network Management Process . . . . .	7
5.2. Closed Loop Management Approach . . . . .	8
6. Deep Exploitation of AI in Network Management . . . . .	9
6.1. From Data to Wisdom . . . . .	9
6.2. External Event Detectors . . . . .	9
6.3. Network Requirement Anticipation . . . . .	10
6.4. Intelligent Reasoning . . . . .	11
6.5. Gaps and Standardization Issues . . . . .	12
7. Relation to Other IETF/IRTF Initiatives . . . . .	13
8. IANA Considerations . . . . .	13
9. Security Considerations . . . . .	13
10. Acknowledgements . . . . .	13
11. References . . . . .	13
11.1. Normative References . . . . .	14
11.2. Informative References . . . . .	14
Appendix A. Information Model to Support Reasoning on External Events . . . . .	15
A.1. Tree Structure . . . . .	15
A.1.1. event-payloads . . . . .	16
A.1.1.1. basic . . . . .	16

A.1.1.2. seismometer . . . . .	16
A.1.1.3. bigdata . . . . .	17
A.1.2. external-events . . . . .	17
A.1.3. notifications/event . . . . .	17
A.2. YANG Module . . . . .	18
Appendix B. The Autonomic Resource Control Architecture (ARCA) .	19
Appendix C. ARCA Integration With ETSI-NFV-MANO . . . . .	21
C.1. Functional Integration . . . . .	21
C.2. Target Experiment and Scenario . . . . .	24
C.3. OpenStack Platform . . . . .	25
C.4. Initial Results . . . . .	27
Authors' Addresses . . . . .	29

## 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 enforcement 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 or 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 greatly 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 detector, 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 least, 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 operations are performed within the control and/or management planes, the security level of reasoning solutions is inherited and thus determined by the security measures 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.1.1.1. 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.1.1.1.1. 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. Additionally 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 formatted 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, if 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 limitations 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

Network Working Group  
Internet Draft  
Intended status: Informational  
Expires: January 2020

Q. Sun  
China Telecom  
W. Liu  
Huawei Technologies  
K. Xie  
Beijing University of Posts and Telecommunications  
July 8, 2019

An Intent-driven Management Framework  
draft-sun-nmrg-intent-framework-00

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 8, 2009.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this

document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Abstract

Intent was defined as an abstract high-level policy (or instruction) used to trigger network operations (not only configuration aspects, but also continuous tuning to adjust the measured/actual network state to an expected state). This document describes the intent-driven management architecture, its key elements, and interfaces.

## Table of Contents

1. Introduction .....	2
2. Acronyms .....	3
3. Framework for Generic Intent-driven Management .....	3
3.1. Overview .....	3
3.2. Operation .....	6
3.2.1 Intent layer .....	6
(1) Intent management .....	6
(2) Intent translation .....	7
(3) Verification module .....	8
(4) Decision module .....	9
3.2.2 Control layer .....	9
(1) Configuration delivery .....	10
(2) Network status collection .....	10
3.2.3 Network layer .....	10
(1) Configuration execution .....	10
(2) Network status awareness .....	10
4. Security Considerations .....	11
5. IANA Considerations .....	11
6. Contributors .....	11
7. Acknowledgments .....	12
8. References .....	12
8.1. Normative References .....	12
8.2. Informative References .....	12

## 1. Introduction

The digital transformation and booming applications and new services (e.g., AR/VR) lead to a rapid growth in the variety and volume of traffic forwarded by underlying networks (including data centers). Existing network technologies and the limited operation and management manpower, make it more challenging. To support efficient and faster service delivery (by means of high level of automation) ,

the network must evolve from a static resource management system to a more dynamic system that consistently and continuously meets business goals.

The concept of Intent-driven management was proposed to deal with this issue. This is a networking model in which high level abstracted business requirements or business request, i.e., "intents", are formally defined and then the network continuously monitors whether these "intentions" are met. Such high-level abstracted business request are translated into actions (including configuration) and maintained by a set of management function blocks in one or more network management systems.

[RFC7575] defines Intent as an abstract high-level policy used to operate the network. Intent management system includes an interface for users/applications to input requests and an engine to translate the intents into the network actions and manage their lifecycle. This document describes an Intent-driven architecture, its elements, and interfaces.

## 2. Acronyms

CLI: Command Line Interface

SDO: Standards Development Organization

VPN: Virtual Private Network

## 3. Framework for Generic Intent-driven Management

This section briefly describes the design and operation of the Intent-driven management framework.

### 3.1. Overview

Figure 1 shows a simplified functional architecture of how Intent is used to manage a network (e.g., network element configuration fault, performance and security management, etc.).

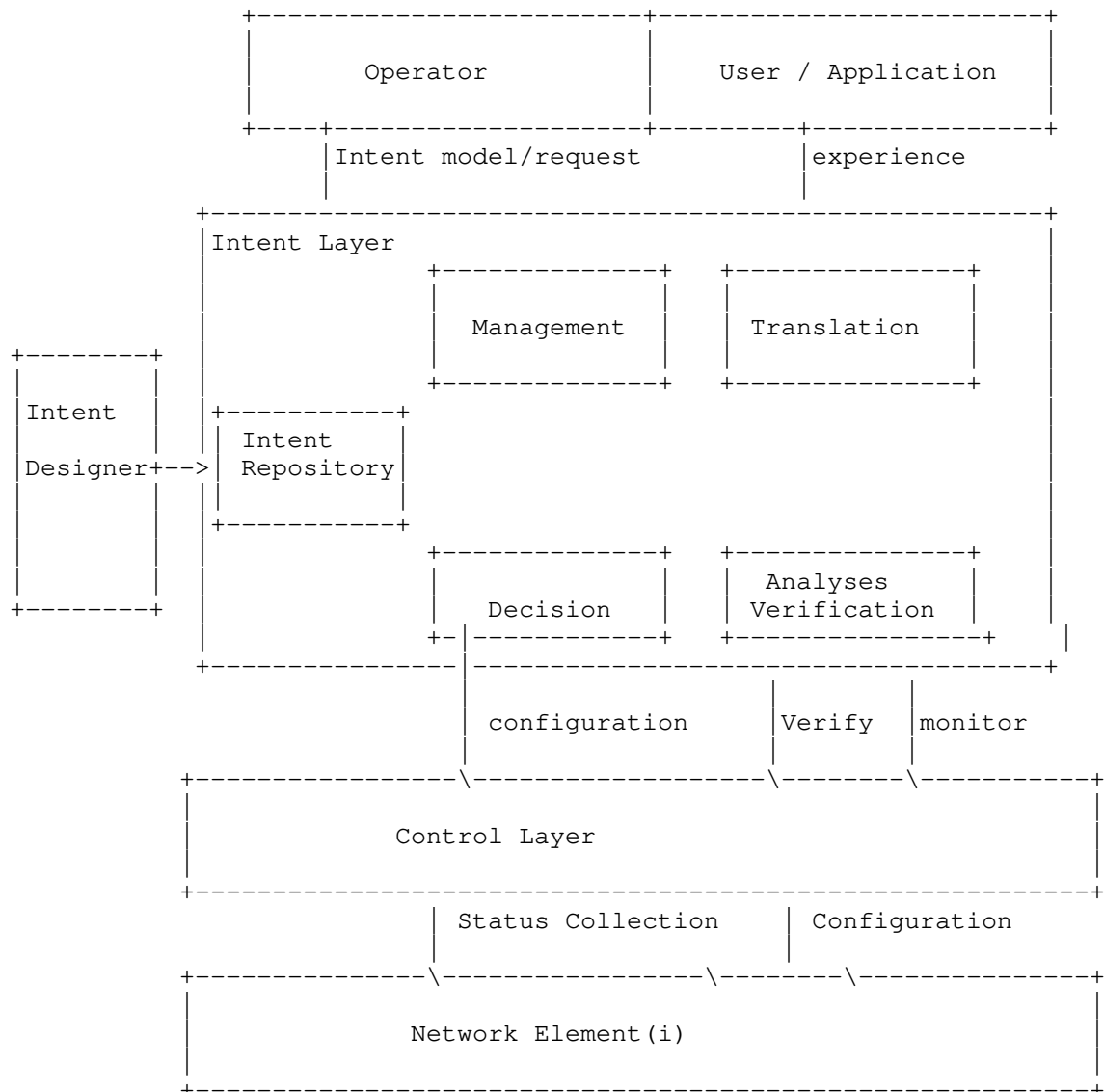


Figure 1 Intent-driven Management Framework

In the architecture of intent-driven management, the intent orchestration layer (hereinafter referred to as intent layer) consists of four functional modules, which are "intent management", "translation", "verification", and "decision" module.

Combined with the request/experience from upper layer (including operators, administrators, users or applications. The different roles of identities in the network may cause different levels of abstraction involved in the intent request. The term "user" in this framework refers to all the entities who make intent requests), the intent-driven management system can collect the intent request, map out appropriate policy, verify correctness and then deploy configuration automatically.

When the configuration is executed in the network (i.e. network element) the verification module should verify the validity of configuration implementation based on real-time status. Therefore, the effectiveness of the configuration is repeatedly verified and monitored to ensure that the user's intent is effectively processed and implemented. In case of any unexpected situation is encountered, it can automatically make remedial measures instantly, and provide feedback to the user to achieve a complete lifecycle.

In the intent-driven networking, the application layer or service layer no longer directly interacts with network layer, but communicates indirectly with the network layer through the intermediate intent layer with certain technical agreement.

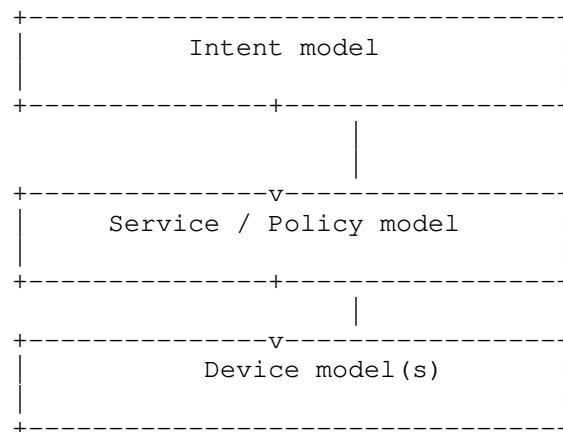


Figure 2 The Model Process of Intent Transformation

Figure 2 presents the overall intent transformation process. At the entrance, the user or administrator needs to express their desired "intent" according to the intent model, which is collected by management module in a high-level language.

With the help of policy database, the translation module and decision module can analyze the intent model and transform it into specific strategy or policy, namely service / policy model.

Eventually the policy will be mapped into concrete configuration action which can be deployed and executed in device level, therefore referred as device model.

The network elements used in this framework are:

Control Layer, which represents one or more entities that are able to control the operation and management of a network infrastructure (e.g., a network topology that consists of Network Elements).

Network Element, which can interact with local or remote Control Layer in order to exchange information, such as configuration information, policy enforcement capabilities, and network status.

### 3.2. Sample Operation

#### 3.2.1 Intent layer

##### (1) Intent management

After receiving the user/application intent from the business layer, the intent orchestration layer needs to manage and coordinate these intents. Eligible intent will be further submitted to the intent translation module for analysis and processing, and the ineligible intent will be fed back to the user by the management module, for further modification and adjustment; the management module also needs to interact with the decision module, and may receive the negative feedback from decision module when the configuration execution does not work as expected, in which cases the management module would require translation module to do secondary processing (take optimization procedure or remedial solution).

In the process of intent demand, the module of the intent layer not only processes once, but continuously verifies and updates the configuration in the closed loop to finally realize the user's intent demand, so the management module also needs to regulate the life cycle of the intent demand.

In addition, because the intent needs to be from different identities, they contain different levels of demands and

abstractions. The abstraction level of the intent demand can be roughly divided into the following levels:

Device level (Level 0): In a traditional networking, the configuration of the device can be manually performed by the network administrator, such as configuring the ACL (Access Control List) of the device to allow an IP X to access a port of the server;

Network level (level 1): In some partially automated networking, network administrators usually do not manually configure specific commands on specific underlying devices. Instead, they set corresponding network permission settings through the controller, such as setting an IP X can access a port of the server through an access point AP

Abstract intent level 1 (level 2): In a fully automated networking, we hope that the network configuration can be completed in the form of intent demands, but the intent demands should include specific details, for example: set " a user A can access a specific service of a server in the campus from the internal network or the external network. "

Abstract intent level 2 (level 3): In this level, we hope that the network can be optimized spontaneously, and the intent defined at this level is more abstract than the previous level, for example, a specific group of users can access a specific service in any way;

Abstract intent level 3 (level 4): In this level, the network belongs to a part of the autonomous network, and the network can automatically decide a more appropriate configuration scheme, and the expression of the intention can be more abstract, for example, " Every user can access a service in a suitable way ";

Abstract intent level 4 (level 5): At the highest level of the intent abstraction model, we hope that the network can achieve full autonomy, automatically decision-making and spontaneous optimization based on real-time network state information anytime and anywhere. This level expresses the expectation that the network will decide the solution autonomously and never fail.

According to different levels of intent demands, the configuration should also be classified into different levels. Therefore, the management module needs to handle and treat the levels of "intent"- "configuration" one by one.

## (2) Intent translation

The translation module needs to analyze the intent and formulate the corresponding configuration based on network status and analysis results in the intent-policy repository, and then output the configuration plan to the verification module. When the negative feedback of the management module is received subsequently, the intent analysis and translation module is also responsible for the dynamic adjustment and re-enactment of the strategy to finally achieve the user's intention; if the positive feedback is received, the configuration is handed over to the control layer. The scope of intent includes several aspects including access control, bandwidth management, QoS levels, security issue, energy consumption control, etc., which enables application providers and users to reasonably mobilize resources and self-service as needed.

In the process of analyzing intent demands, first, the translation module needs to translate and split the semantics contained in the intent demand, because user-initiated intent demands are often complex and more natural-oriented, and to facilitate subsequent processing, the translation module needs to be translated and split according to a specific model. In this step, the method and result of natural language processing (NLP) in the field of artificial intelligence (AI) can be used to train the AI agent as an auxiliary, and the analysis result of the AI auxiliary module is used as a reference for the translation module.

### (3) Verification module

The verification module has two functions: one is the execution verification of the configuration, that is, the configuration plan obtained during the translation process needs to be verified whether it can be executed in the actual network environment according to the real-time network status, and provide relevant information to the decision module as feedback; the second function is validity verification, for the reason that, when the configuration is actually executed, the network status may not change as expected, in which cases, it is necessary to verify whether the execution of the configuration works out as expected and whether the configuration meets the user's intent. If the expected effect is achieved, the verification result is fed back to the decision module; if the verification fails, the "intent"- "configuration" translation procedure needs to be performed again.

The network is dynamically changing, so network verification should be continuous in real time. The verification module is responsible for monitoring the global information in the network, especially those that have an impact on the intent of the implementation. The

verification module needs to quickly reflect the monitoring information to the decision module to ensure that the execution of the configuration does not cause conflicts.

#### (4) Decision module

The decision module is responsible for the overall monitoring of the network status and configuration. It can process the data transmitted by the verification module, and then decide whether the configuration can be delivered to control layer. The decision module supports empirical knowledge from top administrators to help make decisions. When the user's intent is not implemented or the network status is getting abnormal, the decision module needs to make optimization or remedial measures by notifying the intent management module and the translation module to make prompt response.

### 5 Policy repository

Regarding the storage of the intent model and the policy model by the policy repository, we must consider the different levels of the intent model and the policy model, and also consider under what conditions the data will be updated. Policy repository as a database of "intent" - "configuration" information should be able to interact with the intent management module and the translation module, provide relevant data for the intent model for the intent management module, and provide mapping between the "intent" and "configuration" for the translation module. According to the identity of the user, the abstraction level involved in different intent demands also needs to be graded. For the policy repository, the mapping between "intent" and "configuration" should be one-to-one correspondence at the abstract level.

When the translation module finds that an "intent"- "configuration" mapping relationship exists in the policy repository, it cannot be directly deployed to the network layer, but also through the executable verification of the verification module, and then the decision-making module makes a decision whether to issue the decision, because the execution of the configuration needs to consider the actual network environment and state. When the management module or the translation module obtains the intent model or the configuration model that is not in the policy repository, after the intent demand is implemented, the new mapping information should also be updated by the management module into the policy repository to ensure subsequent use.

#### 3.2.2 Control layer

#### (1) Configuration delivery

In an intent-driven networking, in consideration of the configuration set of the intent layer output, the intent layer performs configuration control and further delivery to the control layer through the downward interface to ensure that the configuration can be smoothly delivered to the network layer for execution. The configuration delivery module requires further fine-grained distribution, according to different network structures such as wired access or wireless access, virtual network infrastructure or physical network facilities, to achieve the ability of different networks to work together.

#### (2) Network status collection

The downward interface of the control layer should be able to perceive the physical or logical topology of the underlying network layer, network traffic, network device status, etc., and the upward interface can transmit information to the intent layer to assist the intent orchestration layer to formulate a reasonable scheduling strategy and better utilize the capabilities of the intelligent communication network.

### 3.2.3 Network layer

#### (1) Configuration execution

The configuration execution function can be configured with pre-defined rules and templates, or can be fully dynamically configured according to the controller: for example, through the control protocol between the controller and the network forwarding device, the network layer receives the result of the configuration control function and performs the forwarding and processing of the configuration.

#### (2) Network status awareness

In an intent-driven intelligent communication networking, the sensing information collection function supports the collection of network topology information, network traffic information, and business path information, and can interact in real time through dynamic protocols. Especially when the configuration is actually executed, feedback information can be output to the network state collection function for subsequent verification.

#### 4. Security Considerations

This document does not have any Security Considerations.

#### 5. IANA Considerations

This document has no actions for IANA.

#### 6. Contributors

The following people all contributed to creating this document,  
listed in alphabetical order:

Xueying Han  
Institute of Network Technology,  
Beijing University of Posts and Telecommunications,  
Xitucheng Road, Haidian District, Beijing 100876

Email: hanxueying@bupt.edu.cn

Ruoshui Zhang  
China Telecom,  
No.118, Xizhimennei Street, Beijing 100035  
P. R. China

Email: zrs\_1995@bupt.edu.cn

Qiuhuan Ren  
China Telecom,  
No.118, Xizhimennei Street, Beijing 100035  
P. R. China

Email: renqiuhuan123@bupt.edu.cn

## 7. Acknowledgments

This document has benefited from reviews, suggestions, comments and proposed text provided by the following members, listed in alphabetical order: Mohamed Boucadair.

## 8. References

### 8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

### 8.2. Informative References

[RFC3198] Westerinen, A., Schnizlein, J., Strassner, J., Scherling, M., Quinn, B., Herzog, S., Huynh, A., Carlson, M., Perry, J., Waldbusser, S., "Terminology for Intent-driven Management", RFC 3198, November 2001

[RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.

[RFC7285] R. Alimi, R. Penno, Y. Yang, S. Kiesel, S. Previdi, W. Roome, S. Shalunov, R. Woundy "Application-Layer Traffic Optimization (ALTO) Protocol", September 2014.

[RFC8328] Liu, W., Xie, C., Strassner, J., Karagiannis, G., Klyus, M., Bi, J., Cheng, Y., and D. Zhang, "Policy-Based Management Framework for the Simplified Use of Policy Abstractions (SUPA)", March 2018.

Authors' Addresses

Qiong Sun  
China Telecom  
No.118, Xizhimennei Street, Beijing 100035  
P. R. China

Email: [sunqiong.bri@chinatelecom.cn](mailto:sunqiong.bri@chinatelecom.cn)

Will(Shucheng) Liu  
Huawei Technologies  
Bantian, Longgang District, Shenzhen 518129  
P.R. China

Email: [liushucheng@huawei.com](mailto:liushucheng@huawei.com)

Kun Xie  
School of Software Engineering,  
Beijing University of Posts and Telecommunications  
Xitucheng Road, Haidian District, Beijing 100876  
P.R. China

Email: [pat@bupt.edu.cn](mailto:pat@bupt.edu.cn)

