Abstract

This document describes an architecture for Service Assurance for Intent-based Networking (SAIN). This architecture aims at assuring that service instances are correctly running. As services rely on multiple sub-services by the underlying network devices, getting the assurance of a healthy service is only possible with a holistic view of network devices. This architecture not only helps to correlate the service degradation with the network root cause but also the impacted services when a network component fails or degrades.

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 October 25, 2021.
1. Terminology

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.

SAIN Agent: Component that communicates with a device, a set of devices, or another agent to build an expression graph from a received assurance graph and perform the corresponding computation.

Assurance Graph: DAG representing the assurance case for one or several service instances. The nodes (also known as vertices in the context of DAG) are the service instances themselves and the subservices, the edges indicate a dependency relations.

SAIN collector: Component that fetches or receives the computer-consumable output of the agent(s) and displays it in a user friendly form or process it locally.

DAG: Directed Acyclic Graph.

ECMP: Equal Cost Multiple Paths

Expression Graph: Generic term for a DAG representing a computation in SAIN. More specific terms are:

- Subservice Expressions: expression graph representing all the computations to execute for a subservice.
- Service Expressions: expression graph representing all the computations to execute for a service instance, i.e. including the computations for all dependent subservices.
- Global Computation Graph: expression graph representing all the computations to execute for all services instances (i.e. all computations performed).

Dependency: The directed relationship between subservice instances in the assurance graph.

Informational Dependency: Type of dependency whose score does not impact the score of its parent subservice or service instance(s) in the assurance graph. However, the symptoms should be taken into account in the parent service instance or subservice instance(s), for informational reasons.

Impacting Dependency: Type of dependency whose score impacts the score of its parent subservice or service instance(s) in the assurance graph. The symptoms are taken into account in the parent service instance or subservice instance(s), as the impacting reasons.
Metric: Information retrieved from a network device.

Metric Engine: Maps metrics to a list of candidate metric implementations depending on the target model.

Metric Implementation: Actual way of retrieving a metric from a device.

Network Service YANG Module: describes the characteristics of service, as agreed upon with consumers of that service [RFC8199].

Service Instance: A specific instance of a service.

Service configuration orchestrator: Quoting RFC8199, "Network Service YANG Modules describe the characteristics of a service, as agreed upon with consumers of that service. That is, a service module does not expose the detailed configuration parameters of all participating network elements and features but describes an abstract model that allows instances of the service to be decomposed into instance data according to the Network Element YANG Modules of the participating network elements. The service-to-element decomposition is a separate process; the details depend on how the network operator chooses to realize the service. For the purpose of this document, the term "orchestrator" is used to describe a system implementing such a process."

SAIN Orchestrator: Component of SAIN in charge of fetching the configuration specific to each service instance and converting it into an assurance graph.

Health status: Score and symptoms indicating whether a service instance or a subservice is healthy. A non-maximal score MUST always be explained by one or more symptoms.

Health score: Integer ranging from 0 to 100 indicating the health of a subservice. A score of 0 means that the subservice is broken, a score of 100 means that the subservice is perfectly operational.

Subservice: Part of an assurance graph that assures a specific feature or subpart of the network system.

Symptom: Reason explaining why a service instance or a subservice is not completely healthy.
2. Introduction

Network Service YANG Modules [RFC8199] describe the configuration, state data, operations, and notifications of abstract representations of services implemented on one or multiple network elements.

Quoting RFC8199: "Network Service YANG Modules describe the characteristics of a service, as agreed upon with consumers of that service. That is, a service module does not expose the detailed configuration parameters of all participating network elements and features but describes an abstract model that allows instances of the service to be decomposed into instance data according to the Network Element YANG Modules of the participating network elements. The service-to-element decomposition is a separate process; the details depend on how the network operator chooses to realize the service. For the purpose of this document, the term "orchestrator" is used to describe a system implementing such a process."

In other words, service configuration orchestrators deploy Network Service YANG Modules through the configuration of Network Element YANG Modules. Network configuration is based on those YANG data models, with protocol/encoding such as NETCONF/XML [RFC6241], RESTCONF/JSON [RFC8040], gNMI/gRPC/protobuf, etc. Knowing that a configuration is applied doesn’t imply that the service is running correctly (for example the service might be degraded because of a failure in the network), the network operator must monitor the service operational data at the same time as the configuration. The industry has been standardizing on telemetry to push network element performance information.

A network administrator needs to monitor her network and services as a whole, independently of the use cases or the management protocols. With different protocols come different data models, and different ways to model the same type of information. When network administrators deal with multiple protocols, the network management must perform the difficult and time-consuming job of mapping data models: the model used for configuration with the model used for monitoring. This problem is compounded by a large, disparate set of data sources (MIB modules, YANG models [RFC7950], IPFIX information elements [RFC7011], syslog plain text [RFC3164], TACACS+ [RFC8907], RADIUS [RFC2865], etc.). In order to avoid this data model mapping, the industry converged on model-driven telemetry to stream the service operational data, reusing the YANG models used for configuration. Model-driven telemetry greatly facilitates the notion of closed-loop automation whereby events from the network drive remediation changes back into the network.
However, it proves difficult for network operators to correlate the service degradation with the network root cause. For example, why does my L3VPN fail to connect? Why is this specific service slow? The reverse, i.e. which services are impacted when a network component fails or degrades, is even more interesting for the operators. For example, which service(s) is(are) impacted when this specific optic dBm begins to degrade? Which application is impacted by this ECMP imbalance? Is that issue actually impacting any other customers?

Intent-based approaches are often declarative, starting from a statement of the "The service works correctly" and trying to enforce it. Such approaches are mainly suited for greenfield deployments.

Instead of approaching intent from a declarative way, this framework focuses on already defined services and tries to infer the meaning of "The service works correctly". To do so, the framework works from an assurance graph, deduced from the service definition and from the network configuration. This assurance graph is decomposed into components, which are then assured independently. The root of the assurance graph represents the service to assure, and its children represent components identified as its direct dependencies; each component can have dependencies as well. The SAIN architecture maintains the correct assurance graph when services are modified or when the network conditions change.

When a service is degraded, the framework will highlight where in the assurance service graph to look, as opposed to going hop by hop to troubleshoot the issue. Not only can this framework help to correlate service degradation with network root cause/symptoms, but it can deduce from the assurance graph the number and type of services impacted by a component degradation/failure. This added value informs the operational team where to focus its attention for maximum return.

This architecture provides the building blocks to assure both physical and virtual entities and is flexible with respect to services and subservices, of (distributed) graphs, and of components (Section 3.8).

3. Architecture

SAIN aims at assuring that service instances are correctly running. The goal of SAIN is to assure that service instances are operating correctly and if not, to pinpoint what is wrong. More precisely, SAIN computes a score for each service instance and outputs symptoms explaining that score, especially why the score is not maximal. The score augmented with the symptoms is called the health status.
The SAIN architecture is a generic architecture, applicable to multiple environments. Obviously wireline but also wireless, including 5G, virtual infrastructure manager (VIM), and even virtual functions. Thanks to the distributed graph design principle, graphs from different environments/orchestrator can be combined together.

As an example of a service, let us consider a point-to-point L2VPN connection (i.e. pseudowire). Such a service would take as parameters the two ends of the connection (device, interface or subinterface, and address of the other end) and configure both devices (and maybe more) so that a L2VPN connection is established between the two devices. Examples of symptoms might be "Interface has high error rate" or "Interface flapping", or "Device almost out of memory".

To compute the health status of such a service, the service is decomposed into an assurance graph formed by subservices linked through dependencies. Each subservice is then turned into an expression graph that details how to fetch metrics from the devices and compute the health status of the subservice. The subservice expressions are combined according to the dependencies between the subservices in order to obtain the expression graph which computes the health status of the service.

The overall architecture of our solution is presented in Figure 1. Based on the service configuration, the SAIN orchestrator deduces the assurance graph. It then sends to the SAIN agents the assurance graph along some other configuration options. The SAIN agents are responsible for building the expression graph and computing the health statuses in a distributed manner. The collector is in charge of collecting and displaying the current inferred health status of the service instances and subservices. Finally, the automation loop is closed by having the SAIN Collector providing feedback to the network orchestrator.
In order to produce the score assigned to a service instance, the architecture performs the following tasks:

- Analyze the configuration pushed to the network device(s) for configuring the service instance and decide which information is needed from the device(s), such a piece of information being called a metric, which operations to apply to the metrics for computing the health status.
Stream (via telemetry [RFC8641]) operational and config metric values when possible, else continuously poll.

Continuously compute the health status of the service instances, based on the metric values.

3.1. Decomposing a Service Instance Configuration into an Assurance Graph

In order to structure the assurance of a service instance, the service instance is decomposed into so-called subservice instances. Each subservice instance focuses on a specific feature or subpart of the network system.

The decomposition into subservices is an important function of this architecture, for the following reasons:

- The result of this decomposition provides a relational picture of a service instance, that can be represented as a graph (called assurance graph) to the operator.

- Subservices provide a scope for particular expertise and thereby enable contribution from external experts. For instance, the subservice dealing with the optics health should be reviewed and extended by an expert in optical interfaces.

- Subservices that are common to several service instances are reused for reducing the amount of computation needed.

The assurance graph of a service instance is a DAG representing the structure of the assurance case for the service instance. The nodes of this graph are service instances or subservice instances. Each edge of this graph indicates a dependency between the two nodes at its extremities: the service or subservice at the source of the edge depends on the service or subservice at the destination of the edge.

Figure 2 depicts a simplistic example of the assurance graph for a tunnel service. The node at the top is the service instance, the nodes below are its dependencies. In the example, the tunnel service instance depends on the peer1 and peer2 tunnel interfaces, which in turn depend on the respective physical interfaces, which finally depend on the respective peer1 and peer2 devices. The tunnel service instance also depends on the IP connectivity that depends on the IS-IS routing protocol.
Fig. 2: Assurance Graph Example

Depicting the assurance graph helps the operator to understand (and assert) the decomposition. The assurance graph shall be maintained during normal operation with addition, modification and removal of service instances. A change in the network configuration or topology shall be reflected in the assurance graph. As a first example, a change of routing protocol from IS-IS to OSPF would change the assurance graph accordingly. As a second example, assuming that ECMP is in place for the source router for that specific tunnel; in that case, multiple interfaces must now be monitored, on top of the monitoring the ECMP health itself.

3.2. Intent and Assurance Graph

The SAIN orchestrator analyzes the configuration of a service instance to:

- Try to capture the intent of the service instance, i.e. what is the service instance trying to achieve,

- Decompose the service instance into subservices representing the network features on which the service instance relies.
The SAIN orchestrator must be able to analyze configuration from various devices and produce the assurance graph.

To schematize what a SAIN orchestrator does, assume that the configuration for a service instance touches 2 devices and configure on each device a virtual tunnel interface. Then:

- Capturing the intent would start by detecting that the service instance is actually a tunnel between the two devices, and stating that this tunnel must be functional. This is the current state of SAIN, however it does not completely capture the intent which might additionally include, for instance, on the latency and bandwidth requirements of this tunnel.

- Decomposing the service instance into subservices would result in the assurance graph depicted in Figure 2, for instance.

In order for SAIN to be applied, the configuration necessary for each service instance should be identifiable and thus should come from a "service-aware" source. While the Figure 1 makes a distinction between the SAIN orchestrator and a different component providing the service instance configuration, in practice those two components are mostly likely combined. The internals of the orchestrator are currently out of scope of this document.

3.3. Subservices

A subservice corresponds to subpart or a feature of the network system that is needed for a service instance to function properly. In the context of SAIN, subservice is actually a shortcut for subservice assurance, that is the method for assuring that a subservice behaves correctly.

Subservices, just as with services, have high-level parameters that specify the type and specific instance to be assured. For example, assuring a device requires the specific deviceId as parameter. For example, assuring an interface requires the specific combination of deviceId and interfaceId.

A subservice is also characterized by a list of metrics to fetch and a list of computations to apply to these metrics in order to infer a health status.

3.4. Building the Expression Graph from the Assurance Graph

From the assurance graph is derived a so-called global computation graph. First, each subservice instance is transformed into a set of subservice expressions that take metrics and constants as input (i.e.
sources of the DAG) and produce the status of the subservice, based on some heuristics. Then for each service instance, the service expressions are constructed by combining the subservice expressions of its dependencies. The way service expressions are combined depends on the dependency types (impacting or informational). Finally, the global computation graph is built by combining the service expressions. In other words, the global computation graph encodes all the operations needed to produce health statuses from the collected metrics.

Subservices shall be device independent. To justify this, let’s consider the interface operational status. Depending on the device capabilities, this status can be collected by an industry-accepted YANG module (IETF, Openconfig), by a vendor-specific YANG module, or even by a MIB module. If the subservice was dependent on the mechanism to collect the operational status, then we would need multiple subservice definitions in order to support all different mechanisms. This also implies that, while waiting for all the metrics to be available via standard YANG modules, SAIN agents might have to retrieve metric values via non-standard YANG models, via MIB modules, Command Line Interface (CLI), etc., effectively implementing a normalization layer between data models and information models.

In order to keep subservices independent from metric collection method, or, expressed differently, to support multiple combinations of platforms, OSes, and even vendors, the framework introduces the concept of "metric engine". The metric engine maps each device-independent metric used in the subservices to a list of device-specific metric implementations that precisely define how to fetch values for that metric. The mapping is parameterized by the characteristics (model, OS version, etc.) of the device from which the metrics are fetched.

3.5. Building the Expression from a Subservice

Additionally, to the list of metrics, each subservice defines a list of expressions to apply on the metrics in order to compute the health status of the subservice. The definition or the standardization of those expressions (also known as heuristic) is currently out of scope of this standardization.

3.6. Open Interfaces with YANG Modules

The interfaces between the architecture components are open thanks to the YANG modules specified in YANG Modules for Service Assurance [I-D.claise-opsawg-service-assurance-yang]; they specify objects for assuring network services based on their decomposition into so-called subservices, according to the SAIN architecture.
This module is intended for the following use cases:

- **Assurance graph configuration:**
  
  * Subservices: configure a set of subservices to assure, by specifying their types and parameters.
  
  * Dependencies: configure the dependencies between the subservices, along with their types.

- **Assurance telemetry:** export the health status of the subservices, along with the observed symptoms.

### 3.7. Handling Maintenance Windows

Whenever network components are under maintenance, the operator wants to inhibit the emission of symptoms from those components. A typical use case is device maintenance, during which the device is not supposed to be operational. As such, symptoms related to the device health should be ignored, as well as symptoms related to the device-specific subservices, such as the interfaces, as their state changes is probably the consequence of the maintenance.

To configure network components as "under maintenance" in the SAIN architecture, the ietf-service-assurance model proposed in [I-D.claise-opsawg-service-assurance-yang] specifies an "under-maintenance" flag per service or subservice instance. When this flag is set and only when this flag is set, the companion field "maintenance-contact" must be set to a string that identifies the person or process who requested the maintenance. Any symptom produced by a service or subservice under maintenance, or by one of its dependencies MUST NOT be be reported. A service or subservice under maintenance MAY propagate a symptom "Under Maintenance" towards services or subservices that depend on it.

We illustrate this mechanism on three independent examples based on the assurance graph depicted in Figure 2:

- **Device maintenance**, for instance upgrading the device OS. The operator sets the "under-maintenance" flag for the subservice "Peer1" device. This inhibits the emission of symptoms from "Peer1 Physical Interface", "Peer1 Tunnel Interface" and "Tunnel Service Instance". All other subservices are unaffected.

- **Interface maintenance**, for instance replacing a broken optic. The operator sets the "under-maintenance" flag for the subservice "Peer1 Physical Interface". This inhibits the emission of
symptoms from "Peer 1 Tunnel Interface" and "Tunnel Service Instance". All other subservices are unaffected.

- Routing protocol maintenance, for instance modifying parameters or redistribution. The operator sets the "under-maintenance" flag for the subservice "IS-IS Routing Protocol". This inhibits the emission of symptoms from "IP connectivity" and "Tunnel Service Instance". All other subservices are unaffected.

### 3.8. Flexible Architecture

The SAIN architecture is flexible in terms of components. While the SAIN architecture in Figure 1 makes a distinction between two components, the SAIN configuration orchestrator and the SAIN orchestrator, in practice those two components are mostly likely combined. Similarly, the SAIN agents are displayed in Figure 1 as being separate components. Practically, the SAIN agents could be either independent components or directly integrated in monitored entities. A practical example is an agent in a router.

The SAIN architecture is also flexible in terms of services and subservices. Most examples in this document deal with the notion of Network Service YANG modules, with well-known services such as L2VPN or tunnels. However, the concepts of services is general enough to cross into different domains. One of them is the domain of service management on network elements, with also requires its own assurance. Examples includes a DHCP server on a Linux server, a data plane, an IPFIX export, etc. The notion of "service" is generic in this architecture. Indeed, a configured service can itself be a service for someone else. Exactly like an DHCP server/ data plane/IPFIX export can be considered as services for a device, exactly like an routing instance can be considered as a service for a L3VPN, exactly like a tunnel can considered as a service for an application in the cloud. The assurance graph is created to be flexible and open, regardless of the subservice types, locations, or domains.

The SAIN architecture is also flexible in terms of distributed graphs. As shown in Figure 1, our architecture comprises several agents. Each agent is responsible for handling a subgraph of the assurance graph. The collector is responsible for fetching the subgraphs from the different agents and gluing them together. As an example, in the graph from Figure 2, the subservices relative to Peer 1 might be handled by a different agent than the subservices relative to Peer 2 and the Connectivity and IS-IS subservices might be handled by yet another agent. The agents will export their partial graph and the collector will stitch them together as dependencies of the service instance.
And finally, the SAIN architecture is flexible in terms of what it monitors. Most, if not all examples, in this document refer to physical components but this is not a constrain. Indeed, the assurance of virtual components would follow the same principles and an assurance graph composed of virtualized components (or a mix of virtualized and physical ones) is well possible within this architecture.

3.9.  Timing

The SAIN architecture requires the Network Time Protocol (NTP) [RFC5905] between all elements: monitored entities, SAIN agents, Service Configuration Orchestrator, the SAIN Collector, as well as the SAIN Orchestrator. This guarantees the correlations of all symptoms in the system, correlated with the right assurance graph version.

The SAIN agent might have to remove some symptoms for specific subservice symptoms, because there are outdated and not relevant any longer, or simply because the SAIN agent needs to free up some space. Regardless of the reason, it’s important for a SAIN collector (re-)connecting to a SAIN agent to understand the effect of this garbage collection. Therefore, the SAIN agent contains a YANG object specifying the date and time at which the symptoms history starts for the subservice instances.

3.10. New Assurance Graph Generation

The assurance graph will change along the time, because services and subservices come and go (changing the dependencies between subservices), or simply because a subservice is now under maintenance. Therefore an assurance graph version must be maintained, along with the date and time of its last generation. The date and time of a particular subservice instance (again dependencies or under maintenane) might be kept. From a client point of view, an assurance graph change is triggered by the value of the assurance-graph-version and assurance-graph-last-change YANG leaves. At that point in time, the client (collector) follows the following process:

- Keep the previous assurance-graph-last-change value (let’s call it time T)
- Run through all subservice instance and process the subservice instances for which the last-change is newer that the time T
- Keep the new assurance-graph-last-change as the new referenced date and time
4. Security Considerations

The SAIN architecture helps operators to reduce the mean time to detect and mean time to repair. As such, it should not cause any security threats. However, the SAIN agents must be secure: a compromised SAIN agents could be sending wrong root causes or symptoms to the management systems.

Except for the configuration of telemetry, the agents do not need "write access" to the devices they monitor. This configuration is applied with a YANG module, whose protection is covered by Secure Shell (SSH) [RFC6242] for NETCONF or TLS [RFC8446] for RESTCONF.

The data collected by SAIN could potentially be compromising to the network or provide more insight into how the network is designed. Considering the data that SAIN requires (including CLI access in some cases), one should weigh data access concerns with the impact that reduced visibility will have on being able to rapidly identify root causes.

If a closed loop system relies on this architecture then the well known issue of those system also applies, i.e., a lying device or compromised agent could trigger partial reconfiguration of the service or network. The SAIN architecture neither augments or reduces this risk.

5. IANA Considerations

This document includes no request to IANA.

6. Contributors

- Youssef El Fathi
- Eric Vyncke

7. Open Issues

Refer to the Intent-based Networking NMRG documents

8. References

8.1. Normative References


Claise, et al. Expires October 25, 2021
8.2. Informative References

[Internetservice-assurance-yang]

[RFC2865]

[RFC3164]

[RFC6241]

[RFC6242]

[RFC7011]

[RFC7950]

[RFC8040]
Internet-DraService Assurance for Intent-based Networking Ar April 2021


Appendix A. Changes between revisions

v02 - v03
  o Timing Concepts
  o New Assurance Graph Generation

v01 - v02
  o Handling maintenance windows
  o Flexible architecture better explained
  o Improved the terminology
  o Notion of mapping information model to data model, while waiting for YANG to be everywhere
  o Started a security considerations section

v00 - v01
  o Terminology clarifications
  o Figure 1 improved

Acknowledgements
The authors would like to thank Stephane Litkowski, Charles Eckel, Rob Wilton, Vladimir Vassiliev, Gustavo Alburquerque, Stefan Vallin, and Eric Vyncke for their reviews and feedback.

Authors’ Addresses

Benoit Claise
Huawei
Email: benoit.claise@huawei.com

Jean Quilbeuf
Independent
Email: jean@quilbeuf.net

Diego R. Lopez
Telefonica I+D
Don Ramon de la Cruz, 82
Madrid 28006
Spain
Email: diego.r.lopez@telefonica.com

Dan Voyer
Bell Canada
Canada
Email: daniel.voyer@bell.ca

Thangam Arumugam
Cisco Systems, Inc.
Milpitas (California)
United States of America
Email: tarumuga@cisco.com
YANG Modules for Service Assurance
draft-claise-opsawg-service-assurance-yang-07

Abstract

This document proposes YANG modules for the Service Assurance for Intent-based Networking Architecture.

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 October 25, 2021.

Copyright Notice

Copyright (c) 2021 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
Table of Contents

1. Terminology .................................................. 3
2. Introduction .................................................. 3
3. YANG Models Overview ......................................... 3
4. Base ietf-service-assurance YANG module ................. 4
   4.1. Tree View .............................................. 4
   4.2. Concepts ............................................... 5
   4.3. YANG Module ............................................ 6
5. Subservice Extension: ietf-service-assurance-device YANG
   module ......................................................... 13
   5.1. Tree View .............................................. 13
   5.2. Complete Tree View .................................... 13
   5.3. Concepts ............................................... 14
   5.4. YANG Module ............................................ 15
   module ......................................................... 16
   6.1. Tree View .............................................. 16
   6.2. Complete Tree View .................................... 17
   6.3. Concepts ............................................... 18
   6.4. YANG Module ............................................ 18
7. Vendor-specific Subservice Extension: example-service-
   assurance-device-acme YANG module ......................... 19
   7.1. Tree View .............................................. 19
   7.2. Complete Tree View .................................... 20
   7.3. Concepts ............................................... 21
   7.4. YANG Module ............................................ 22
8. Security Considerations ...................................... 23
9. IANA Considerations .......................................... 24
   9.1. The IETF XML Registry ................................ 24
   9.2. The YANG Module Names Registry ....................... 25
10. Open Issues .................................................. 25
11. References .................................................. 25
   11.1. Normative References ................................ 25
   11.2. Informative References ................................ 26
Appendix A. Changes between revisions .......................... 26
Acknowledgements ................................................ 27
Authors’ Addresses ............................................... 27
1. Terminology

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.

The terms used in this document are defined in draft-claise-opsawg-service-assurance-architecture IETF draft.

2. Introduction

The "Service Assurance for Intent-based Networking Architecture" draft-claise-opsawg-service-assurance-architecture, specifies the framework and all of its components for service assurance. This document complements the architecture by providing open interfaces between components. More specifically, the goal is to provide YANG modules for the purpose of service assurance in a format that is:

- machine readable
- vendor independent
- augmentable

3. YANG Models Overview

The main YANG module, ietf-service-assurance, defines objects for assuring network services based on their decomposition into so-called subservices. The subservices are hierarchically organised by dependencies. The subservices, along with the dependencies, constitute an assurance graph. This module should be supported by an agent, able to interact with the devices in order to produce a health status and symptoms for each subservice in the assurance graph. This module is intended for the following use cases:

- Assurance graph configuration:
  - Subservices: configure a set of subservices to assure, by specifying their types and parameters.
  - Dependencies: configure the dependencies between the subservices, along with their type.

- Assurance telemetry: export the health status of the subservices, along with the observed symptoms.
The second YANG module, ietf-service-assurance-device, extends the ietf-service-assurance module to add support for the subservice DeviceHealthy. Additional subservice types might be added the same way.

The third YANG module, example-service-assurance-device-acme, extends the ietf-service-assurance-device module as an example to add support for the subservice DeviceHealthy, with specifics for the fictional ACME Corporation. Additional vendor-specific parameters might be added the same way.

4. Base ietf-service-assurance YANG module

4.1. Tree View

The following tree diagram [RFC8340] provides an overview of the ietf-service-assurance data model.

```
module: ietf-service-assurance
   +--ro assurance-graph-version        yang:counter32
   +--ro assurance-graph-last-change    yang:date-and-time
   +--rw subservices
       +--rw subservice* [type id]
           +--rw type                                identityref
           +--rw id                                  string
           +--ro last-change?                        yang:date-and-time
           +--ro label?                              string
           +--rw under-maintenance?                  boolean
           +--rw maintenance-contact                 string
           +--rw (parameter)?
               +--:(service-instance-parameter)
                   +--rw service-instance-parameter
                       +--rw service?         string
                       +--rw instance-name?   string
           +--ro health-score?                       uint8
           +--ro symptoms-history-start?             yang:date-and-time
           +--rw symptoms
               +--ro symptom* [start-date-time id]
                   +--ro id                               string
                   +--ro health-score-weight?               uint8
                   +--ro description?                      string
                   +--ro start-date-time                   yang:date-and-time
                   +--ro stop-date-time                    yang:date-and-time
               +--rw dependencies
                   +--rw dependency* [type id]
                       +--rw type                               -> /subservices/subservice/type
                       +--rw id                                 -> /subservices/subservice[type=current()]/../type/id
               +--rw dependency-type?                   identityref
```
4.2. Concepts

The ietf-service-assurance YANG model assumes an identified number of subservices, to be assured independently. A subservice is a feature or a subpart of the network system that a given service instance might depend on. Example of subservices include:

- DeviceHealthy: whether a device is healthy, and if not, what are the symptoms. Potential symptoms are "CPU overloaded", "Out of RAM", or "Out of TCAM".

- ConnectivityHealthy: given two IP addresses owned by two devices, what is the quality of the connection between them. Potential symptoms are "No route available" or "ECMP Imbalance".

The first example is a subservice representing a subpart of the network system, while the second is a subservice representing a feature of the network. In both cases, these subservices might depend on other subservices, for instance, the connectivity might depend on a subservice representing the routing mechanism and on a subservice representing ECMP.

The symptoms are listed for each subservice. Each symptom is specified by a unique id and contains a health-score-weight (the impact to the health score incurred by this symptom), a label (text describing what the symptom is), and dates and times at which the symptom was detected and stopped being detected. While the unique id is sufficient as an unique key list, the start-date-time second key help sorting and retrieving relevant symptoms.

The assurance of a given service instance can be obtained by composing the assurance of the subservices that it depends on, via the dependency relations.

In order to declare a subservice MUST provide:

- A type: identity inheriting of the base identity for subservice,
- An id: string uniquely identifying the subservice among those with the same identity,
- Some parameters, which should be specified in an augmenting model, as described in the next sections.

The type and id uniquely identify a given subservice. They are used to indicate the dependencies. Dependencies have types as well. Two types are specified in the model:
o Impacting: such a dependency indicates an impact on the health of the dependent,

o Informational: such a dependency might explain why the dependent has issues but does not impact its health.

To illustrate the difference between "impacting" and "informational", consider the subservice InterfaceHealthy, representing a network interface. If the device to which the network interface belongs goes down, the network interface will transition to a down state as well. Therefore, the dependency of InterfaceHealthy towards DeviceHealthy is "impacting". On the other hand, as a the dependency towards the ECMPLoad subservice, which checks that the load between ECMP remains stable throughout time, is only "informational". Indeed, services might be perfectly healthy even if the load distribution between ECMP changed. However, such an instability might be a relevant symptom for diagnosing the root cause of a problem.

Service instances MUST be modeled as a particular type of subservice with two parameters, a type and an instance name. The type is the name of the service defined in the network orchestrator, for instance "point-to-point-l2vpn". The instance name is the name assigned to the particular instance that we are assuring, for instance the name of the customer using that instance.

The "under-maintenance" and "maintenance-contact" flags inhibit the emission of symptoms for that subservice and subservices that depend on them. See Section 3.7 of [draft-claise-opsawg-service-assurance-architecture] for a more detailed discussion.

By specifying service instances and their dependencies in terms of subservices, one defines the whole assurance to apply for them. An assurance agent supporting this model should then produce telemetry in return with, for each subservice: a health-status indicating how healthy the subservice is and when the subservice is not healthy, a list of symptoms explaining why the subservice is not healthy.

4.3. YANG Module

<CODE BEGINS> file "ietf-service-assurance@2020-01-13.yang"

module ietf-service-assurance {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-service-assurance";
  prefix service-assurance;

  import ietf-yang-types {


prefix yang;

organization
  "IETF NETCONF (Network Configuration) Working Group";
contact
  "WG Web: <https://datatracker.ietf.org/wg/netconf/>
  WG List: <mailto:netconf@ietf.org>
  Author: Benoit Claise <mailto:bclaise@cisco.com>
  Author: Jean Quilbeuf <mailto:jquilbeu@cisco.com>";
description
  "This module defines objects for assuring network services based on
  their decomposition into so-called subservices, according to the SAIN
  (Service Assurance for Intent-based Networking) architecture.

  The subservices hierarchically organised by dependencies constitute an
  assurance graph. This module should be supported by an assurance agent,
  able to interact with the devices in order to produce a health status
  and symptoms for each subservice in the assurance graph.

  This module is intended for the following use cases:
  * Assurance graph configuration:
    * subservices: configure a set of subservices to assure, by specifying
      their types and parameters.
    * dependencies: configure the dependencies between the subservices,
      along with their type.
  * Assurance telemetry: export the health status of the subservices, along
    with the observed symptoms.

  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 (RFC 2119)
  (RFC 8174) when, and only when, they appear in all
  capitals, as shown here.

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

  Redistribution and use in source and binary forms, with or
  without modification, is permitted pursuant to, and subject
  to the license terms contained in, the Simplified BSD License
  set forth in Section 4.c of the IETF Trust’s Legal Provisions
  Relating to IETF Documents

  This version of this YANG module is part of RFC XXXX; see the
  RFC itself for full legal notices.

TO DO:
- Better type (IETF or OC) for device-id, interface-id, etc.
- Have a YANG module for IETF and one for OC?;

revision 2020-01-13 {
  description
    "Added the maintenance window concept.";
  reference
    "RFC xxxx: Title to be completed";
}

revision 2019-11-16 {
  description
    "Initial revision.";
  reference
    "RFC xxxx: Title to be completed";
}

identity subservice-idty {
  description
    "Root identity for all subservice types.";
}

identity service-instance-idty {
  base subservice-idty;
  description
    "Identity representing a service instance.";
}

identity dependency-type {
  description
    "Base identity for representing dependency types.";
}

identity informational-dependency {
  base dependency-type;
  description
    "Indicates that symptoms of the dependency might be of interest for the dependent, but the status of the dependency should not have any impact on the dependent.";
}

identity impacting-dependency {
  base dependency-type;
  description
    "Indicates that the status of the dependency directly impacts the status of the dependent.";
}
grouping symptom {
    description
        "Contains the list of symptoms for a specific subservice.";
    leaf id {
        type string;
        description
            "A unique identifier for the symptom.";
    }
    leaf health-score-weight {
        type uint8 {
            range "0 .. 100";
        }
        description
            "The weight to the health score incurred by this symptom. The higher the
            value, the more of an impact this symptom has. If a subservice health
            score is not 100, there must be at least one symptom with a health
            score weight larger than 0.";
    }
    leaf description {
        type string;
        description
            "Description of the symptom, i.e. text describing what the symptom is, to
            be computer-consumable and be displayed on a human interface. ";
    }
    leaf start-date-time {
        type yang:date-and-time;
        description
            "Date and time at which the symptom was detected.";
    }
    leaf stop-date-time {
        type yang:date-and-time;
        description
            "Date and time at which the symptom stopped being detected.";
    }
}

grouping subservice-dependency {
    description
        "Represent a dependency to another subservice.";
    leaf type {
        type leafref {
            path "/subservices/subservice/type";
        }
        description
            "The type of the subservice to refer to (e.g. DeviceHealthy).";
    }
    leaf id {
        type leafref {
            path "/subservices/subservice/id";
        }
        description
            "The ID of the subservice to refer to;";
    }
}
leaf id {
  type string;
  mandatory true;
  description
    "The identifier of the subservice to refer to.";
}

leaf dependency-type {
  type identityref {
    base dependency-type;
  }
  description
    "Represents the type of dependency (i.e. informational, impacting).";
}

// augment here if more info are needed (i.e. a percentage) depending on the
dependency type.

leaf assurance-graph-version {
  type yang:counter32;
  mandatory true;
  config false;
  description
    "The assurance graph version, which increases by 1 for each new version, af
    ter the changes
    (dependencies and/or maintenance windows parameters) are applied to the su
    bservice(s).";
}

leaf assurance-graph-last-change {
  type yang:date-and-time;
  mandatory true;
  config false;
  description
    "Date and time at which the assurance graph last changed after the changes
    (dependencies
    and/or maintenance windows parameters) are applied to the subservice(s). Th
    ese date and time
    must be more recent or equal compared to the more recent value of any chan
    ged subservices
    last-change";
}

container subservices {
  description
    "Root container for the subservices.";
  list subservice {
    key "type id";
    description
      "List of subservice configured.";
    leaf type {
      type identityref {
        base subservice-idty;
      }
      description
        "Name of the subservice, e.g. DeviceHealthy.";
    }
    leaf id {
type string;
  description
    "Unique identifier of the subservice instance, for each type.";
}

leaf last-change {
  type yang:date-and-time;
  config false;
  description
    "Date and time at which the assurance graph for this subservice
     instance last changed, i.e. dependencies and/or maintenance windows pa
     rameters.";
}

leaf label {
  type string;
  config false;
  description
    "Label of the subservice, i.e. text describing what the subservice is t
    o be displayed on a human interface.";
}

leaf under-maintenance {
  type boolean;
  default false;
  description
    "An optional flag indicating whether this particular subservice is unde
    r maintenance. Under this circumstance, the subservice symptoms and the
    symptoms of its dependencies in the assurance graph should not be taken
    into account. Instead, the subservice should send a 'Under Maintenance'
    single symptom.

    The operator changing the under-maintenance value must set the
    maintenance-contact variable.

    When the subservice is not under maintenance any longer, the
    under-maintenance flag must return to its default value and
    the under-maintenance-owner variable deleted.";
}

leaf maintenance-contact {
  when "../under-maintenance = 'true'";
  type string;
  mandatory true;
  description
    "A string used to model an administratively assigned name of the
    resource that changed the under-maintenance value to 'true.

    It is suggested that this name contain one or more of the following:
    IP address, management station name, network manager’s name, location,
    or phone number. In some cases the agent itself will be the owner of
    an entry. In these cases, this string shall be set to a string
    starting with 'monitor'.";
choice parameter {
    description "Specify the required parameters per subservice type."
    container service-instance-parameter {
        when "derived-from-or-self(../type, 'service-assurance:service-instance-idty')"
        description "Specify the parameters of a service instance."
        leaf service {
            type string;
            description "Name of the service."
        }
        leaf instance-name{
            type string;
            description "Name of the instance for that service."
        }
    }
    // Other modules can augment their own cases into here
}
leaf health-score {
    type uint8 {
        range "0 .. 100";
    }
    config false;
    description "Score value of the subservice health. A value of 100 means that subservice is healthy. A value of 0 means that the subservice is broken. A value between 0 and 100 means that the subservice is degraded.";
}
leaf symptoms-history-start {
    type yang:date-and-time;
    config false;
    description "Date and time at which the symptoms history starts for this subservice instance, either because the subservice instance started at that date and time or because the symptoms before that were removed due to a garbage collection process."
}
container symptoms {
    description "Symptoms for the subservice.";
    list symptom {
        key "start-date-time id";
        config false;
        description "List of symptoms the subservice. While the start-date-time key is not necessary per se, this would get the entries sorted by start-date-tim"
for easy consumption."
uses symptom;
}
}
container dependencies {
    description
    "configure the dependencies between the subservices, along with their
types.";
    list dependency {
        key "type id";
        description
        "List of soft dependencies of the subservice.";
        uses subservice-dependency;
    }
}
}

<CODE ENDS>

5. Subservice Extension: ietf-service-assurance-device YANG module

5.1. Tree View

The following tree diagram [RFC8340] provides an overview of the
ietf-service-assurance-device data model.

module: ietf-service-assurance-device
    augment /service-assurance:subservices/service-assurance:subservice/service-assurance:parameter:
        +++-rw device-idty
        +++-rw device?   string

5.2. Complete Tree View

The following tree diagram [RFC8340] provides an overview of the
ietf-service-assurance and ietf-service-assurance-device data models.
module: ietf-service-assurance
  +--ro assurance-graph-version         yang:counter32
  +--ro assurance-graph-last-change    yang:date-and-time
  +--rw subservices
    +--rw subservice* [type id]
      +--rw type                                          identityref
      +--rw id                                            string
      +--ro last-change?                                   yang:date-and-time
      +--ro label?                                        string
      +--rw under-maintenance?                            boolean
      +--rw maintenance-contact                           string
      +--rw (parameter)?
        +--:(service-instance-parameter)
          +--rw service-instance-parameter
            +--rw service?         string
            +--rw instance-name?   string
          +--:(service-assurance-device:device-idty)
            +--rw service-assurance-device:device-idty
              +--rw service-assurance-device:device?   string
            +--ro health-score?                                 uint8
            +--ro symptoms-history-start?                       yang:date-and-time
      +--rw symptoms
        +--ro symptom* [start-date-time id]
          +--ro id                                            string
          +--ro health-score-weight?  uint8
          +--ro description?           string
          +--ro start-date-time        yang:date-and-time
          +--ro stop-date-time?        yang:date-and-time
      +--rw dependencies
        +--rw dependency* [type id]
          +--rw type                           -> /subservices/subservice/type
          +--rw id                             -> /subservices/subservice[type=current()]/.
          +--rw dependency-type?               identityref

5.3. Concepts

As the number of subservices will grow over time, the YANG module
is designed to be extensible. A new subservice type requires the
precise specifications of its type and expected parameters. Let us
illustrate the example of the new DeviceHealthy subservice type. As
the name implies, it monitors and reports the device health, along
with some symptoms in case of degradation.

For our DeviceHealthy subservice definition, the new device-idty is
specified, as an inheritance from the base identity for subservices.
This indicates to the assurance agent that we are now assuring the
health of a device.
The typical parameter for the configuration of the DeviceHealthy subservice is the name of the device that we want to assure. By augmenting the parameter choice from ietf-service-assurance YANG module for the case of the device-idty subservice type, this new parameter is specified.

5.4. YANG Module

<CODE BEGINS> file "ietf-service-assurance-device@2020-01-13.yang"

module ietf-service-assurance-device {
    yang-version 1.1;
    namespace "urn:ietf:params:xml:ns:yang:ietf-service-assurance-device";
    prefix service-assurance-device;

    import ietf-service-assurance {
        prefix "service-assurance";
    }
}

organization
    "IETF NETCONF (Network Configuration) Working Group";
contact
    "WG Web: <https://datatracker.ietf.org/wg/netconf>"
    "WG List: <mailto:netconf@ietf.org>"
    "Author: Benoit Claise <mailto:bclaise@cisco.com>"
    "Author: Jean Quilbeuf <mailto:jquilbeu@cisco.com>"

description
    "This module extends the ietf-service-assurance module to add support for the subservice DeviceHealthy."

    Checks whether a network device is healthy.

    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 (RFC 2119) (RFC 8174) when, and only when, they appear in all capitals, as shown here.

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

    Redistribution and use in source and binary forms, with or without modification, is permitted pursuant to, and subject to the license terms contained in, the Simplified BSD License set forth in Section 4.c of the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info).

Internet-Draft YANG Modules for Service Assurance April 2021

This version of this YANG module is part of RFC XXXX; see the RFC itself for full legal notices.

revision 2020-01-13 {
  description
  "Added the maintenance window concept."
  reference
  "RFC xxxx: Title to be completed"
}

revision 2019-11-16 {
  description
  "Initial revision."
  reference
  "RFC xxxx: Title to be completed"
}

identity device-idty {
  base service-assurance:subservice-idty;
  description "Network Device is healthy."
}

augment /service-assurance:subservices/service-assurance:subservice/service-assurance:parameter {
  description
  "Specify the required parameters for a new subservice type"
  container device-idty{
    when "derived-from-or-self(../service-assurance:type, 'device-idty')"
    description
      "Specify the required parameters for the device-idty subservice type"
    leaf device {
      type string;
      description "The device to monitor."
    }
  }
}

<CODE ENDS>


6.1. Tree View

The following tree diagram [RFC8340] provides an overview of the ietf-service-assurance-interface data model.
module: ietf-service-assurance-interface
  augment /service-assurance:subservices/service-assurance:subservice/service-assurance:parameter:
    +--rw device?      string
    +--rw interface?   string

6.2. Complete Tree View

The following tree diagram [RFC8340] provides an overview of the ietf-service-assurance, ietf-service-assurance-device, and ietf-service-assurance-interface data models.

module: ietf-service-assurance
  +--ro assurance-graph-version        yang:counter32
  +--ro assurance-graph-last-change    yang:date-and-time
  +--rw subservices
    +--rw subservice* [type id]
      +--rw type                                           identityref
      +--rw id                                             string
      +--ro last-change?                                   yang:date-and-time
      +--ro label?                                         string
      +--rw under-maintenance?                             boolean
      +--rw maintenance-contact                            string
      +--rw (parameter)?
        +--:(service-instance-parameter)
          +--rw service-instance-parameter
            +--rw service?         string
            +--rw instance-name?   string
        +--:(service-assurance-device:device-idty)
          +--rw service-assurance-device:device?   string
        +--:(service-assurance-interface:device)
          +--rw service-assurance-interface:device?      string
        +--:(service-assurance-interface:interface)
          +--rw service-assurance-interface:interface?   string
          +--ro health-score?                                  uint8
          +--ro symptoms-history-start?                        yang:date-and-time
          +--rw symptoms
            +--ro symptom* [start-date-time id]
              +--ro id                                     string
              +--ro health-score-weight?  uint8
              +--ro description?             string
              +--ro start-date-time         yang:date-and-time
              +--ro stop-date-time?         yang:date-and-time
          +--rw dependencies
            +--rw dependency* [type id]
              +--rw type                   -> /subservices/subservice/type
              +--rw id                    -> /subservices/subservice[type=current()]/
                ../type]/id
              +--rw dependency-type?     identityref
6.3. Concepts

For our InterfaceHealthy subservice definition, the new interface-idty is specified, as an inheritance from the base identity for subservices. This indicates to the assurance agent that we are now assuring the health of an interface.

The typical parameters for the configuration of the InterfaceHealthy subservice are the name of the device and, on that specific device, a specific interface. By augmenting the parameter choice from ietf-service-assurance YANG module for the case of the interface-idty subservice type, those two new parameter are specified.

6.4. YANG Module

<CODE BEGINS> file "ietf-service-assurance-interface@2020-01-13.yang"

module ietf-service-assurance-interface {
  yang-version 1.1;
  prefix service-assurance-interface;

  import ietf-service-assurance {
    prefix "service-assurance";
  }

  organization
    "IETF OPSAWG Working Group";
  contact
    "WG Web: <https://datatracker.ietf.org/wg/opsawg/>
    WG List: <mailto:opsawg@ietf.org>
    Author: Benoit Claise <mailto:bclaise@cisco.com>
    Author: Jean Quilbeuf <mailto:jquilbeu@cisco.com>
  
  description
    "This module extends the ietf-service-assurance module to add support for the subservice InterfaceHealthy.

    Checks whether an interface is healthy.

    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 (RFC 2119) (RFC 8174) when, and only when, they appear in all capitals, as shown here.

    Copyright (c) 2020 IETF Trust and the persons identified as authors of the code. All rights reserved."
Redistribution and use in source and binary forms, with or without modification, is permitted pursuant to, and subject to the license terms contained in, the Simplified BSD License set forth in Section 4.c of the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info).

This version of this YANG module is part of RFC XXXX; see the RFC itself for full legal notices. 

revision 2020-01-13 {  
description  
"Initial revision.";  
reference  
"RFC xxxx: Title to be completed";  
}  

identity interface-idty {  
base service-assurance:subservice-idty;  
description "Checks whether an interface is healthy.";  
}  

augment /service-assurance:subservices/service-assurance:subservice/service-assurance:parameter {  
when "derived-from-or-self(service-assurance:type, 'interface-idty')";  
description  
"Specify the required parameters for the interface-idty subservice type" ;  
}

leaf device {  
type string;  
description "Device supporting the interface.";  
}  

leaf interface {  
type string;  
description "Name of the interface.";  
}  

}  

<CODE ENDS>

7. Vendor-specific Subservice Extension: example-service-assurance-device-acme YANG module

7.1. Tree View

The following tree diagram [RFC8340] provides an overview of the example-service-assurance-device-acme data model.
module: example-service-assurance-device-acme
  augment /service-assurance:subservices/service-assurance:subservice/service-assurance:parameter:
    +++rw acme-device-idty
    +++rw device?      string
    +++rw acme-specific-parameter?  string

7.2. Complete Tree View

The following tree diagram [RFC8340] provides an overview of the
ietf-service-assurance, ietf-service-assurance-device, and example-
service-assurance-device-acme data models.
module: ietf-service-assurance
  +--ro assurance-graph-version yang:counter32
  +--ro assurance-graph-last-change yang:date-and-time
  +--rw subservices
    +--rw subservice* [type id]
    +--rw type identityref
      +--rw id str
    +--ro last-change? yang:date-and-time
    +--ro label? str
    +--rw under-maintenance? bool
    +--rw maintenance-contact str
  +--rw (parameter)?
    | +--:(service-instance-parameter)
    |   | +--rw service-instance-parameter
    |   |   +--rw service? string
    |   |   +--rw instance-name? string
    | +--:(service-assurance-device:device-idty)
    |   | +--rw service-assurance-device:device-idty
    |   |   +--rw service-assurance-device:device? string
    | +--:(service-assurance-interface:device)
    |   | +--rw service-assurance-interface:device? string
  +--rw symptoms
    +--ro symptom* [start-date-time id]
      +--ro id string
      +--ro health-score-weight? uint8
      +--ro description? string
      +--ro start-date-time yang:date-and-time
      +--ro stop-date-time? yang:date-and-time
    +--rw dependencies
      +--rw dependency* [type id]
        | +--rw type -> /subservices/subservice/type
        | +--rw id -> /subservices/subservice[type=current()]/../type]/id
        +--rw dependency-type? identityref

7.3. Concepts

Under some circumstances, vendor-specific subservice types might be
required. As an example of this vendor-specific implementation, this section shows how to augment the ietf-service-assurance-device module
to add support for the subservice DeviceHealthy, specific to the ACME Corporation. The new parameter is acme-specific-parameter.

7.4. YANG Module

module example-service-assurance-device-acme {
  yang-version 1.1;
  namespace "urn:example:example-service-assurance-device-acme";
  prefix example-service-assurance-device-acme;

  import ietf-service-assurance {
    prefix "service-assurance";
  }

  import ietf-service-assurance-device {
    prefix "service-assurance-device";
  }

  organization
    "IETF NETCONF (Network Configuration) Working Group";
  contact
    "WG Web:  <https://datatracker.ietf.org/wg/netconf/>
    WG List:  <mailto:netconf@ietf.org>
    Author:   Benoit Claise <mailto:bclaise@cisco.com>
    Author:   Jean Quilbeuf <mailto:jquilbeu@cisco.com>";
  description
    "This module extends the ietf-service-assurance-device module to add support for the subservice DeviceHealthy, specific to the ACME Corporation.

ACME Network Device is healthy.

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 (RFC 2119) (RFC 8174) when, and only when, they appear in all capitals, as shown here.

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

Redistribution and use in source and binary forms, with or without modification, is permitted pursuant to, and subject to the license terms contained in, the Simplified BSD License set forth in Section 4.c of the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info).
This version of this YANG module is part of RFC XXXX; see the RFC itself for full legal notices.

revision 2020-01-13 {
  description
  "Added the maintenance window concept.";
  reference
  "RFC xxxx: Title to be completed";
}

revision 2019-11-16 {
  description
  "Initial revision.";
  reference
  "RFC xxxx: Title to be completed";
}

identity device-acme-idty {
  base service-assurance-device:device-idty;
  description "Network Device is healthy.";
}

augment /service-assurance:subservices/service-assurance:subservice/service-assurance:parameter {
  description
  "Specify the required parameters for a new subservice type";
  container acme-device-idty{
    when "derived-from-or-self(../service-assurance:type, 'device-acme-idty')";
    description
      "Specify the required parameters for the device-acme-idty subservice type";
    leaf device {
      type string;
      description "The device to monitor.";
    }
    leaf acme-specific-parameter {
      type string;
      description "The ACME Corporation specific parameter.";
    }
  }
}

8. Security Considerations

The YANG module specified in this document defines a schema for data that is designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer
is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC8446].

The Network Configuration Access Control Model (NACM) [RFC8341] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.

There are a number of data nodes defined in this YANG module that are writable/creatable/deletable (i.e., config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., edit-config) to these data nodes without proper protection can have a negative effect on network operations. These are the subtrees and data nodes and their sensitivity/vulnerability:

- /subservices/subservice/type
- /subservices/subservice/id
- /subservices/subservice/under-maintenance
- /subservices/subservice/maintenance-contact

9. IANA Considerations

9.1. The IETF XML Registry

This document registers two URIs in the IETF XML registry [RFC3688]. Following the format in [RFC3688], the following registrations are requested:

Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.
9.2. The YANG Module Names Registry

This document registers three YANG modules in the YANG Module Names registry [RFC7950]. Following the format in [RFC7950], the following registrations are requested:

name:       ietf-service-assurance
prefix:     inc
reference:  RFC XXXX

name:       ietf-service-assurance-device
prefix:     inc
reference:  RFC XXXX

name:       ietf-service-assurance-interface
prefix:     inc
reference:  RFC XXXX

10. Open Issues

-None

11. References

11.1. Normative References

[draft-claise-opsawg-service-assurance-architecture]


Claise, et al. Expires October 25, 2021
11.2. Informative References


Appendix A. Changes between revisions

v04 – v05

o Added the concept of symptoms-history-start

o Changed label to description, under symptoms. This was confusing as there was two labels in the models

v03 – v04

o Add the interface subservice, with two parameters

v02 – v03

o Added the maintenance window concepts
v01 - v02
  o Improved leaf naming
  o Clarified some concepts: symptoms, dependency

v00 - v01
  o Terminology clarifications
  o Provide example of impacting versus impacted dependencies

Acknowledgements

The authors would like to thank Jan Lindblad for his help during the design of these YANG modules. The authors would like to thank Stephane Litkowski and Charles Eckel for their reviews.

Authors’ Addresses

Benoit Claise
Huawei
Email: benoit.claise@huawei.com

Jean Quilbeuf
Independent
Email: jean@quilbeuf.net

Paolo Lucente
NTT
Sirusdreef 70-72
Hoofddorp, WT 2132
Netherlands
Email: paolo@ntt.net

Paolo Fasano
TIM S.p.A
via G. Reiss Romoli, 274
10148 Torino
Italy
Email: paolo2.fasano@telecomitalia.it
Thangam Arumugam
Cisco Systems, Inc.
Milpitas (California)
United States

Email: tarumuga@cisco.com
Abstract

This memo introduces the use case of the usage of intents for expressing advance interconnection features, further than traditional IP peering.

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 July 24, 2022.

Copyright Notice

Copyright (c) 2022 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.
1. Introduction

The success of Internet-based services has been built on top of the global reachability of content accessed by the end-users, which is facilitated by the interconnection of individual networks owned by distinct service providers constituting independent administrative domains.

Such interconnection services have been initially based simply on delivery of IP traffic between the interconnected parties leveraging on BGP. This peer model enables full connectivity. However, the traditional interconnection model shows some limitations when additional information to that related to routing is needed.

New network capabilities based on programmability and virtualization are producing service situations where a connectivity-only approach is not sufficient. The increasing availability of computing capabilities internal to the networks, or attached to them, enable new scenarios where those capabilities can be consumed through the advertisement or exposure of these execution environments (i.e., in terms of compute, storage and associated networking resources). Such information from an interconnected provider can be obtained from e.g. [I-D.1lc-teas-dc-aware-topo-model].

In addition or complementary to that, even services or network functions could be advertised in order to make them available for interconnection. For example, as service we could consider the advertisement of CDN capabilities as in CDNi approach [RFC7336], while as network function we could consider functions like firewall, CGNAT, etc, present in the network [I-D.ietf-teas-sf-aware-topo-model].

All these scenarios present clear evolutions of the interconnection model which can not be simply expressed through existing mechanisms,
or at least, cannot be expressed in a simple (and comprehensive) way with such existing mechanisms. Here is where an advanced interconnection intent can assist on declaring the goal of the interconnection transcending pure IP traffic exchange and including more advanced capabilities as the ones mentioned before.

2. Evolution of Network Interconnection

It becomes clear the trend to increasingly rely on multi-domain scenarios for the provision of services. For instance, the access today to an on-demand OTT video on Internet implies the interaction of more than one single administrative domain. Thus, end-to-end service delivery over multiple providers or domains is becoming the norm.

Complex network services leveraging on virtualization solutions and different infrastructure environments pertaining to distinct administrative domains (i.e., operated and managed by distinct providers) can be easily foreseen.

It is then necessary to explore mechanisms for interconnecting that multiple domain environments in a common, portable way independently of the owner of such infrastructure.

2.1. Potential Interconnection Intent Types

The interconnection intent should provide enough abstractions to express a variety of interconnection options.

The purpose of the interconnection intent can be multiple:

- to enable multi-domain network service programming, by soliciting interconnection of service / network functions in different domains
- to enable multi-domain deployment of virtualized network functions, by advertising the availability of compute and storage resources in different domains
- to facilitate multi-domain network function or service charging, by advertising (cumulative) costs in the different domains
- to enable traffic interchange, i.e. IP as in traditional peering or optical
- to put in place the right collection of policies to implement and operate the interconnection
o to facilitate whatever combination of all of them

2.2. Interconnection intent lifecycle

[I-D.irtf-nmrg-ibn-concepts-definitions] defines an intent lifecycle composed of two phases, namely fulfillment and assurance. Figure 1 captures the intent procedure for the fulfillment phase (assurance phase will be detailed in future versions of this draft).

User Space : Translation / IBS : Network Ops
Space : Space :

Fulfill : recognize/translate/learn/configure:
<table>
<thead>
<tr>
<th>generate</th>
<th>refine</th>
<th>plan</th>
<th>render</th>
<th>provision</th>
</tr>
</thead>
</table>

- Select interconn. : - Mapping of intent types to : - Establishment of
type protocols / APIs for : protocol sessions
resources (i.e., : - Parametrization of that : for configure or
routes, compute : protocols / APIs, e.g. : provisioning
quotes, service : leveraging on data models : targeted resources
functions, etc.) :

Figure 1: Fulfillment phase of the Interconnection Intent

2.3. Protocol aspects

Ultimately the ideas and notions elaborated in this document will need to find room in a framework made of one or multiple protocols (i.e. BGP, LISP, etc.) and/or API definitions. While the exact definition of such framework is left as future work, in this document we intend to perform some seminal work in this sense (i.e. identify existing protocols that could fit, determine gaps of such protocols, etc.).
3. Interconnection intent structure

To be done.

4. Security Considerations

To be done.

5. IANA Considerations

This draft does not include any IANA considerations

6. References

[I-D.ietf-teas-sf-aware-topo-model]

[I-D.irtf-nmrg-ibn-concepts-definitions]

[I-D.llc-teas-dc-aware-topo-model]


Acknowledgments

This work has been partly funded by the European Commission through the H2020 project 5GROWTH (Grant Agreement no. 856709).

Authors’ Addresses
Luis M. Contreras
Telefonica
Ronda de la Comunicacion, s/n
Sur-3 building, 3rd floor
Madrid 28050
Spain

Email: luismiguel.contrerasmurillo@telefonica.com
URI: http://lmcontreras.com/

Paolo Lucente
NTT
Siriusdreef 70-72
Hoofddorp, WT 2132
Netherlands

Email: paolo@ntt.net
Abstract

Slicing at the transport network is expected to be offered as part of end-to-end network slices, fostered by the introduction of new services such as 5G. This document explores the usage of intent technologies for requesting IETF network slices.

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 July 15, 2022.

Copyright Notice

Copyright (c) 2022 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
1. Introduction

Network slicing is emerging as the future model for service offering in telecom operator networks. Conceptually, network slicing provides a customer with an apparent dedicated network built on top of logical (i.e., virtual) and/or physical functions and resources supported by a shared infrastructure, provided by one or more telecom operators.

The concept of network slicing has been largely fostered by the advent of 5G services that are expected to be deployed on top of different kind of slices, each built to support specific characteristics (extreme low latency, high bandwidth, etc).

As part of an end-to-end network slice it is expected to have a number of network slices at transport level (referred as IETF network slices) providing the necessary connectivity to the rest of components of the end-to-end slice, e.g., mobile packet core slice.

For a definition of an IETF network slice refer to [I-D.ietf-teas-ietf-network-slices]. The following paragraph is directly taken from it: "An IETF Network Slice Service enables connectivity between a set of CEs with specific Service Level Objectives (SLOs) and Service Level Expectations (SLEs) over a common underlay network."

Intent is a high-level, declarative goal 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.
In consequence, it seems very convenient to apply the intent-based mechanisms for the provision of IETF network slices, providing the adequate level of abstraction towards the transport network control and management planes.

This document leverages current industry trends in the definition of end-to-end network slices. The final objective is to describe intents that can be used to flexibly declare the operational aspects and goals of an IETF network slice, meaning that the customer could declare what kind of IETF network slice is needed (the outcome) and not how to achieve the goals of the IETF network slice.

2. IETF network slice intent

As stated in [I-D.irtf-nmrg-ibn-concepts-definitions], "Intent is a declaration of operational goals that a network is supposed to meet and outcomes that the network is supposed to deliver, without specifying how to achieve or how to implement them. Those goals and outcomes are defined in a manner that is purely declarative - they specify what to accomplish, not how to achieve it."

When applied to transport networks, this implies that an intent for IETF network slices should provide the necessary abstraction with respect to implementation details, including the final devices (or resources) involved, and be focused on the characteristics and performance expectations related to it.

With that aim it can be expected that the intent based system can fulfill and assure the requested IETF network slice, triggering initial configurations at the time of initial provisioning and corrective actions during the IETF network slice lifetime.

Regarding the corrective actions it is possible to differentiate two levels. First, corrective actions that could be performed by the management and control capabilities of the network (i.e., by the IETF Network Slice Controller) to maintain the Service level Objectives (SLOs) as originally declared in the slice intent, so being these internal actions to the management and control elements of the network. Second, corrective actions that could be necessary to perform due to incongruences between the SLOs expressed in the intent and the observed monitoring information, then requiring some adaptation to the intent itself in order to perform the corrective action.
Foundation of IETF network slice intents

The industrial interest around 5G is accelerating network deployments and operational changes.

With this respect, the GSMA has been developing a universal blueprint that can be used by any vertical customer to request the deployment of a network slice instance (NSI) based on a specific set of service requirements. Such a blueprint is a network slice descriptor called Generic Slice Template (GST) [GSMA]. The GST contains multiple attributes that can be used to characterize a network slice. A particular template filled with values generates a specific Network Slice Type (NEST).

Such templates refer to the end-to-end network slice, including the transport part. Despite the fact that some of the values would not have applicability for the transport network, others do. An analysis of the relevant attributes is performed in [I-D.contreras-teas-slice-nbi].

According to 3GPP propositions [TS28.541], an upper 3GPP Management System interacts with the transport network for establishing the necessary slices at the transport level. Such interaction can be expected to happen using the IETF network slice intent, described to an intent-based system (IBS) in the transport network part. Then, according to the intent lifecycle in [I-D.irtf-nmrg-ibn-concepts-definitions], the IBS, after recognizing the intent, will proceed to translate it in order to interact with a IETF network slice controller by using a NBI as proposed in [I-D.contreras-teas-slice-nbi].

Figure 1 captures the intent procedure for the fulfillment phase (assurance phase will be detailed in future versions of this draft).
Figure 1: Fulfillment phase of the Transport Slicing Intent

4. Mechanisms for translating IETF network slice intents

This section describes approaches for implementing mechanisms to translate IETF network slice intents. As part of such translation it could be necessary to translate the slice needs expressed by the customer in terms of service-specific SLOs (e.g., high-resolution real-time video quality) to network- or connectivity-specific SLOs (e.g., a correspondent throughput and/or latency) which are the SLOs an IETF Network Slice Controller understands. More on this can be found in [TMV].

4.1. Translation approaches and interaction with the upper systems

A suite of mechanisms will be required to allow instantiation of the user’s intent into a IETF network slice. In order to be able to deliver an end2end Intent driven slice - a well defined set of context aware attributes that allow unambiguous instantiation of the intent should be agreed upon. A combination of a structured set of attributes communicated between an IBN and an upper layer system with user input would allow an IBN to have intent modeled and reason about its completeness/validity. Translation approaches and interaction with the upper systems might benefit from Natural Language Processing (NLP) technics that are needed for enabling high level expression of
requirements found missing. The goal would be to identify and classify the answers for as many fields as possible from the Generic Slice Template (GST), based on the free text / speech provided by the user. As it is highly unlikely that the minimum set of fields to properly define an IETF network slice (geo-temporal characteristics, performance characteristics, SLO and SLA properties) will be fulfilled in this first step, a follow up two-step approach might need to be implemented.

- The minimum missing fields from the GST have to be identified and appropriate questions have to be generated (e.g. based on a pool of available questions correlated with each field, or based on AI approaches).

- An iterative interrogation phase will be initiated towards the user using the previously generated questions, until the user provides all the missing information, so the intent can be modeled accordingly.

Interaction with the user and higher-up systems can potentially be further improved by utilizing Machine Learning techniques.

4.2. Intent-based system suite

In order to consolidate on the set of devices, technologies and resources to be used, a combination of deterministic or stochastic computation approaches will be needed. Deterministic approaches will rely on mathematical models and respective algorithms. Stochastic approaches will rely on technologies like machine learning. Their goal will be to learn from experience, so as to optimize future decisions from the viewpoint of speed and reliability. The target of learning will be related to the service behavior and to the anticipated network status in the area and time period of the service provision.

5. Security Considerations

To be done.

6. IANA Considerations

This draft does not include any IANA considerations

7. References

Acknowledgments

This work has been partly funded by the European Commission through the H2020 project 5G-EVE (Grant Agreement no. 815074).

Contributors

Kostas Tsagkaris, Kostas Trichias, Vassilis Foteinos, and Thanasis Gkiollas (all from WINGS ICT Solutions) have also contributed to this work.

Authors’ Addresses
Luis M. Contreras
Telefonica
Ronda de la Comunicacion, s/n
Sur-3 building, 3rd floor
Madrid  28050
Spain

Email: luismiguel.contrerasmurillo@telefonica.com
URI:   http://lmcontreras.com/

Panagiotis Demestichas
WINGS ICT Solutions
Greece

Email: pdemest@wings-ict-solutions.eu

Jeff Tantsura
Microsoft

Email: jefftant.ietf@gmail.com
Intent Classification

draft-irtf-nmrg-ibn-intent-classification-08

Abstract

Intent is an abstract, high-level policy used to operate the network. Intent-based management system includes an interface for users to input requests and an engine to translate the intents into the network configuration and manage their life-cycle.

This document discusses mostly the concept of network intents, but other types of intents are also being considered. Specifically, it highlights stakeholder perspectives of intent, methods to classify and encode intent, the associated intent taxonomy, and defines relevant intent terms where necessary. This document provides a foundation for intent related research and facilitates solution development.

This document is a product of the IRTF Network Management Research Group (NMRG).

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 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 November 18, 2022.

Copyright Notice

Copyright (c) 2022 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.
Table of Contents

1. Introduction...................................................4
   1.1. Research activities......................................4
   1.2. Standards and open source activities....................5
   1.3. Scope..................................................................6
2. Acronyms..........................................................7
3. Definitions.......................................................8
4. Abstract Intent Requirements...................................8
   4.1. What is Intent?...............................................8
   4.2. Intent Solutions and Intent Users.........................9
   4.3. Benefits of Intents for Different Stakeholders..........11
   4.4. Intent Types that need to be supported..................12
5. Functional Characteristics and Behaviour....................13
   5.1. Abstracting Intent Operation..............................13
   5.2. Intent User Types...........................................14
   5.3. Intent Scope................................................15
   5.4. Intent Network Scope.......................................15
   5.5. Intent Abstraction..........................................16
   5.6. Intent Life-cycle...........................................16
   5.7. Autonomous Driving Levels................................16
6. Intent Classification........................................17
   6.1. Intent Classification Methodology........................18
   6.2. Intent Taxonomy.............................................21
   6.3. Intent Classification for Carrier Solution...............23
       6.3.1. Intent Users and Intent Types.........................23
       6.3.2. Intent Categories.....................................27
       6.3.3. Intent Classification Example........................27
   6.4. Intent Classification for Data Center Network Solutions...31
       6.4.1. Intent Users and Intent Types.........................31
       6.4.2. Intent Categories.....................................35
       6.4.3. Intent Classification Example........................35
   6.5. Intent Classification for Enterprise Solution..............39
       6.5.1. Intent Users and Intent Types.........................39
       6.5.2. Intent Categories.....................................41
6. Conclusions...................................................43
7. Security Considerations.......................................43
8. IANA Considerations...........................................43
9. Contributors...................................................44
10. Acknowledgments...............................................44
12. Informative References.......................................44
1. Introduction

The vision of intent-based networks has attracted a lot of attention, as it promises to simplify the management of networks by human operators. This is done by simply specifying what should happen on the network, without giving any instructions on how to do it. This promise led many researcher-led activities and telecom companies to start researching this new vision, and many Standards Development Organization (SDOs) to propose different intent frameworks.

This draft proposes an intent classification methodology and an intent taxonomy. The scope of these proposals is to ensure a common understanding in the research community in terms of what are the intent users, intent types, or intent solutions, etc. for specific scenarios that are being considered.

The document represents the consensus of the Network Management Research Group (NMRG). It has been reviewed extensively by the Research Group (RG) members who are actively involved in the research and development of the technology covered by this document. It is not an IETF product and is not a standard.

1.1. Research activities

Intent-based networking is an active research topic which spans different areas that could benefit from an intent classification and taxonomy.

One such area is intent expression and recognition ([Bezahaf21], [Bezahaf19]), NILE [Jacobs18]). The use of a common classification can provide consistency in the understanding of the various forms of intent expressions being proposed and investigated.

Another area where this intent classification could contribute is the orchestration of cognitive autonomous RANs [Banerjee21] where intents are classified based on their content.

The work carried in intent network verification [Tian19] where the authors are proposing new intent language is another candidate where intent classification could be used advantageously.

Furthermore, this draft is proving itself already extremely relevant to the research community as it has been used as the basis for proposing self-generated Intent-based systems [Bezahaf19], for advancing IBN-based VNF placement solutions that rely on defining user intent profiles corresponding to abstract network services [Leivadeas21], for improving existing solutions in provisioning...
intent-based networks, and proposing new approaches to service management [Davoli21], or even for defining grammars for users to specify the high-level requirements for blockchain selection in the form of intent [Padovan20]. As well, the draft has been mentioned in surveys addressing the topic of intelligent intent-based autonomous networks [Mehmood21], [Szilagyi21].

This document describes as well an example on how this proposal has been successfully applied in an academic environment [IBN-POC] by researchers in the area of SDN/NFV for defining the scope of their project. The specific problem addressed by researchers is how to apply intent concepts at different levels that correspond to different stakeholders.

IEEE Communications Society Technical Committee on Network Operation and Management (IEEE-CNOM), IRTF-NMRG and IFIP WG6.6 have developed a taxonomy for network and service management [IFIP-NSM] that is used by the research community in network management and operations to structure the research area through a well-defined set of keywords and to improve quality of reviews in submissions to journals, conferences and workshops. The proposed intent taxonomy may be contributed as an extension to this taxonomy for intent driven management.

1.2. Standards and open source activities

Several SDOs and open source projects, such as Internet Research Task Force (IRTF)/Network Management Research Group (NMRG), Open Networking Foundation (ONF) [ONF] / Open Network Operating System (ONOS) [ONOS], European Telecommunications Standards Institute (ETSI)/Experiential Networked Intelligence (ENI), TMF with its Autonomous Networks, have proposed intents for defining a set of network operations to execute in a declarative manner.

More recently, the IRTF NMRG is working on the Intent-based Networking - Concepts and Definitions document, [CLEMM]. This document clarifies the concept of "Intent" and provides an overview of the functionality that is associated with it. The goal is to contribute towards a common and shared understanding of terms, concepts, and functionality that can be used as the foundation to guide further definition of associated research and engineering problems and their solutions.

The present document, together with [CLEMM], aims to become the foundation for future intent-related topic discussions regarding the NMRG.
The SDOs usually came up with their own way of specifying an intent, and with their own understanding of what an intent is. Besides that, each SDO defines a set of terms and level of abstraction, its intended intent users, and the applications and usage scenarios.

However, most intent approaches proposed by SDOs share the same following features:

- It must be declarative in nature, meaning that an intent user specifies the goal on the network without specifying how to achieve that goal.
- It must be vendor agnostic, in the sense that it abstracts the network capabilities, or the network infrastructure from the intent user, and it can be ported across different platforms.
- It must provide an easy-to-use interface, which simplifies the intent users’ interaction with the intent system through the usage of familiar terminology or concepts.
  
  It should be able to detect and resolve intent conflicts, which include, for example, static (compile-time) conflicts and dynamic (run-time) conflicts.

1.3. Scope

The focus of this document is on the definition of criteria enabling to categorize intents from the stakeholders’ viewpoint. Concepts and definitions related to IBN are provided in [CLEMM].

This document mostly addresses intents in the context of network intents, however other types of intents are not excluded, as presented in section 4.4. and section 6.2.

It is impossible to fully differentiate intents only by the common characteristics followed by concepts, terms and intentions. This document clarifies what an intent represents for different stakeholders through a classification on various dimensions, such as solutions, intent users, and intent types. This classification ensures common understanding among all participants and is used to determine the scope and priority of individual projects, proof-of-concept (PoCs), research initiatives, or open source projects.

The scope of intent classification in this document includes solutions, intent users and intent types, and the initial
classification table is made according to this scope. The methodology presented can be used to update the classification tables by adding or removing different solutions, intent users, or intent types to cater for future scenarios, applications or domains.

2. Acronyms

AI: Artificial Intelligence
CE: Customer Equipment
CFS: Customer Facing Service
CLI: Command Line Interface
DB: Database
DC: Data Center
ECA: Event-Condition-Action
GBP: Group-Based Policy
GPU: Graphics Processing Unit
IBN: Intent Based Network
NFV: Network Function Virtualization
O&M: Operations & Maintenance
ONF: Open Networking Foundation
ONOS: Open Network Operating System
PNF: Physical Network Function
QoE: Quality of Experience
RFS: Resource Facing Service
3. Definitions

A common and shared understanding of terms and definitions related to IBN is provided in [CLEMM], as follows:

- **Intent**: A set of operational goals (that a network should meet) and outcomes (that a network is supposed to deliver), defined in a declarative manner without specifying how to achieve or implement them.

- **Intent-Based Network**: A network that can be managed using intent.

- **Policy**: A set of rules that governs the choices in behaviour of a system.

- **Intent User**: A user that defines and issues the intent request to the intent-based management system.

Other definitions relevant to this draft, such as intent scope, intent network scope, intent abstraction, intent abstraction, and intent lifecycle are available in section 5.

4. Abstract Intent Requirements

In order to understand the different intent requirements that would drive intent classification, we first need to understand what intent means for different intent users.

4.1. What is Intent?

The term Intent has become very widely used in the industry for different purposes, sometimes it is not even in agreement with SDO
shared principles mentioned in the Introduction section. [CLEMM] draft brings clarification with relation to what an intent is and how it differentiates from policies and services.

Different stakeholders have different perspective of the network and therefore have different intent requirements. Their intent is sometimes technical, non-technical, abstract or technology specific. Therefore, it is important to start a discussion in the industry and academia communities about what intent is for different solutions and intent users. It is also imperative to try to propose some intent categories/classifications that could be understood by a wider audience. This would help us define intent interfaces, domain-specific languages, and models.

4.2. Intent Solutions and Intent Users

Intent types are defined by all aspects that are required to profile different requirements to easily distinguish among them. However, in order to facilitate a clustered classification, we can focus on two aspects, the solution and intent user. They can be considered as the main keys to classify intents, as we can easily group requirements by solution and intent user.

On the one hand, different solutions and intent users have different requirements, expectations and priorities for intent-based networking. Therefore, intent users require different intent types, depending on their context, since they participate in different use cases. For instance, some intent users are more technical and require intents that expose more technical information. Other intent users do not have knowledge of the network infrastructure and require intents that shield them from different networking concepts and technologies.

The following are the solutions and intent users that intent-based networking needs to support:
These intent solutions and intent users represent a starting point for the classification and are expendable through the methodology presented in section 6.1.

- For carrier networks scenario, for example, if a customer/subscriber wants to watch high-definition video, then the intent is to convert the video image to 1080p rate.

- For DC networks scenario, administrators have their own clear network intent such as load balancing. For all traffic flows that need NFV service chaining, restrict the maximum load of any VNF node/container below 50% and the maximum load of any network link below 70%.

- For enterprise networks scenario, when hosting a video conference multiple remote accesses are required. An example of the intent from the network administrator is: for any end-user of this application, the arrival time of hologram objects of all the remote tele-presenters should be synchronised within 50ms to reach the destination viewer for each conversation session.
4.3. Benefits of Intents for Different Stakeholders

Current network APIs and CLIs are too complex because they are highly integrated with the low level concepts exposed by networks. Customers, application developers and end-users must not be required to set IP addresses, VLANs, subnets, ports, while operators may still want to have more technical and network visibility. All stakeholders would benefit from the simpler interfaces, like:

- Request gold VPN service between my sites A, B and C
- Provide CE redundancy for the customer sites
- Add access rules to the network service

Operators and administrators manually troubleshoot and fix their networks and services. They instead want to:

- simplify and automate network operations
- simplify definitions of network services
- provide simple customer APIs for value added services (operators)
- be informed if the network or service is not behaving as requested
- enable automatic optimization and correction for selected scenarios
- have systems that learn from historic information and behaviour

Currently, intent users cannot build their own services and policies without becoming technical experts and performing manual maintenance actions. They instead want to be able to:

- build their own network services with their own policies via simple interfaces, without becoming networking experts
- have their network services up and running based on intent and automation only, without any manual actions or maintenance
4.4. Intent Types that need to be supported

Next to the intent solutions and intent users, another way to categorize the intent is through the intent types. The following intent types and subtypes need to be supported, in order to address the requirements from different solutions and intent users:

- **Customer service intent**
  - for customer self-service with SLA
  - for service operator orders

- **Network and underlay network service intent**
  - for service operator orders
  - for intent driven network configuration, verification, correction and optimization
  - for intent created and provided by the underlay network administrator

- **Network and underlay network intent**
  - for network configuration
  - for automated lifecycle management of network configurations
  - for network resources (switches, routers, routing, policies, underlay)

- **Cloud management intent**
  - for DC configuration, VMs, DB servers, APP servers
  - for communication between VMs

- **Cloud resource management intent**
  - for cloud resource life-cycle management (policy driven self-configuration and auto-scaling and recovery/optimization)

- **Strategy intent**
  - for security, QoS, application policies, traffic steering, etc.
o for configuring and monitoring policies, alarms generation for non-compliance, auto-recovery
o for design models and policies for network and network service design
o for design workflows, models and policies for operational task intents

- Operational task intents
  - for network migration
  - for device replacements
  - for network software upgrades
  - for automating any other tasks that operators/administrator often perform

It is important to mention there all of the previously mentioned types and subtypes may affect other intents. For example, operational task intent can modify many other intents. The task itself is short-lived, but the modification of other intents has an impact on their life-cycle, so those changes must continue to be continuously monitored and self-corrected/self-optimized.

5. Functional Characteristics and Behaviour

Intent can be used to operate immediately on a target (much like issuing a command), or whenever it is appropriate (e.g., in response to an event). In either case, intent has a number of behaviours that serve to further organize its purpose, as described by the following subsections.

5.1. Abstracting Intent Operation

The modelling of intents can be abstracted using the following three-tuple:

(Context, Capabilities, Constraints)

- Context grounds the intent, and determines if it is relevant or not for the current situation. Thus, context selects intents based on applicability.
o Capabilities describe the functionality that the intent can perform. Capabilities take different forms, depending on the expressivity of the intent as well as the programming paradigm(s) used.

o Constraints define any restrictions on the capabilities to be used for that particular context.

Metadata can be attached via strategy templates to each of the elements of the three-tuple, and may be used to describe how the intent should be used and how it operates, as well as prescribe any operational dependencies that must be taken into account.

Although different intent categories share the same abstracted intent model, each category will have its own specific context, capabilities and constraints.

5.2. Intent User Types

Expanding on the introduction in section 4.2., intent user types represent the intent users that define and issue the intent request. Depending on the intent solutions, there are specific intent users. Examples of intent users are customers, network operators, service operators, enterprise administrators, cloud administrators, and underlay network administrators, or application developers.

o Customers and end-users do not necessarily know the functional and operational details of the network that they are using. Furthermore, they lack skills to understand such details; in fact, such knowledge is typically not relevant to their job. In addition, the network may not expose these details to its intent users. This class of intent users focuses on the applications that they run, and uses services offered by the network. Hence, they want to specify policies that provide consistent behaviour according to their business needs. They do not have to worry about how the intents are deployed onto the underlying network, and especially, whether the intents need to be translated to different forms to enable network elements to understand them.
Application developers work in a set of abstractions defined by their application and programming environment(s). For example, many application developers think in terms of objects (e.g., a VPN). While this makes sense to the application developer, most network devices do not have a VPN object per se; rather, the VPN is formed through a set of configuration statements for that device in concert with configuration statements for the other devices that together make up the VPN. Hence, the view of application developers matches the services provided by the network, but may not directly correspond to other views of other intent users.

Network operators may have the knowledge of the underlying network. However, they may not understand the details of the applications and services of customers.

5.3. Intent Scope

Intents are used to manage the behaviour of the networks they are applied to and all intents are applied within a specific scope, such as:

- Connectivity scope, if the intent creates or modifies a connection.
- Security/privacy scope, if the intent specifies the security characteristics of the network, customers, or end-users.
- Application scope, when the intent specifies the applications to be affected by the intent request.
- QoS scope, when the intent specifies the QoS characteristics of the network.

These intent scopes are expendable through the methodology presented in section 6.1.

5.4. Intent Network Scope

Regardless on the intent user type, their intent request is affecting the network, or network components, which are representing the intent targets.

Thus, intent network scope, or policy target as known in the area of declarative policy, can represent VNFs or PNFs, physical network elements, campus networks, SD-WAN networks, radio access networks, cloud edge, cloud core, branch, etc.
5.5. Intent Abstraction

Intent can be classified by whether it is necessary to feedback technical network information or non-technical information to the intent user after the intent is executed. As well, intent abstraction covers the level of technical details in the intent itself.

- For non-technical intent users, they do not care how the intent is executed, or the details of the network. As a result, they do not need to know the configuration information of the underlying network. They only focus on whether the intent execution result achieves the goal, and the execution effect such as the quality of completion and the length of execution. In this scenario, we refer to an abstraction without technical feedback.

- For administrators, such as network administrators, they perform intents, such as allocating network resources, selecting transmission paths, handling network failures, etc. They require multiple feedback indicators for network resource conditions, congestion conditions, fault conditions, etc. after execution. In this case, we refer to an abstraction with technical feedback.

As per intent definition provided in [CLEMM], lower-level intents are not considered to qualify as intents. However, we kept this classification to identify any PoCs/Demos/Use Cases that still either require or implement lower level of abstraction for intents.

5.6. Intent Life-cycle

Intents can be classified into transient and persistent intents:

- If the intent is transient, it has no life-cycle management. As soon as the specified operation is successfully carried out, the intent is finished, and can no longer affect the target object.

- If the intent is persistent, it has life-cycle management. Once the intent is successfully activated and deployed, the system will keep all relevant intents active until they are deactivated or removed.

5.7. Autonomous Driving Levels

In different phases of the autonomous driving network [TMF-auto], the intents are different. Depending on the Autonomous Network Level of the overall solution, we may have different intent requirements and
types. For example, at lower level the customer intent is automatically converted to configuration policies only, while at the higher levels the customer intent is covering the full life cycle, it is converted to both configuration and monitoring policies and is self-assured using AI.

A typical example of autonomous driving network level 0 to 5 are listed as below.

- Level 0 - Traditional manual network: O&M personnel manually control the network and obtain network alarms and logs. - No intent

- Level 1 - Partially automated network: Automated scripts are used to automate service provisioning, network deployment, and maintenance. Shallow perception of network status and decision making suggestions of machine; - No intent

- Level 2 - Automated network: Automation of most service provisioning, network deployment, and maintenance of a comprehensive perception of network status and local machine decision making; - simple intent on service provisioning

- Level 3 - Self-optimization network: Deep awareness of network status and automatic network control, meeting requirements of intent users of the network. - Intent based on network status cognition

- Level 4 - Partial autonomous network: In a limited environment, people do not need to participate in decision-making and networks can adjust itself. - Intent based on limited AI

- Level 5 - Autonomous network: In different network environments and network conditions, the network can automatically adapt to and adjust to meet people’s intentions. - Intent based on AI

6. Intent Classification

This section proposes an intent classification approach that may help to classify mainstream intent related demos/tools.
The three classifications in this document have been proposed from scratch, following the methodology presented, through three iterations: one for carrier network intent solution, one for DC intent solution, and one for enterprise intent solution. For each intent solution, we identified the specific intent users and intent types. Then, we further identified intent scope, network scope, abstractions, and life-cycle requirements.

These classifications and the generated tables can be easily extended. For example, for the DC intent solution, a new category is identified, i.e. resource scope, and the classification table has been extended accordingly.

In the future, as new scenarios, applications, and domains are emerging, new classifications and taxonomies can be identified, following the proposed methodology.

The intent classifications have been documented to the best of our knowledge at this point in time. Additional classifications will most probably see the light in the future.

The output of the intent classification is the intent taxonomy introduced in the next sections.

Thus, this section first introduces the proposed intent classification methodology, followed by consolidated intent taxonomy for three intent solutions, and then by concrete examples of intent classifications for three different intent solutions (e.g. carrier network, data center, and enterprise) that were derived using the proposed methodology and then can be filled in for PoCs, demos, research projects or future drafts.

6.1. Intent Classification Methodology

This section describes the methodology used to derive the initial classification proposed in the draft. The proposed methodology can be used to create new intent classifications from scratch, by analysing the solution knowledge. As well, the methodology can be used to update existing classification tables by adding or removing different solutions, intent users or intent types in order to cater for future scenarios, applications or domains.
Solution Knowledge (requirements, use cases, technologies, network, intent users, intent requirements)

Input

Rx=Read
Ux=Update (Add/Remove)

1. Identify Intent
Solution

8. Identify New Categories
R1 | U1
R2

7. Identify Life-cycle Requirements
R3

6. Identify Abstractions
R4

5. Identify Network Scope
R5

4. Identify Intent Scope
U4

3. Identify Intent Classification
U3

2. Identify Intent User Types
U2

Figure 1 - Intent Classification Methodology
The intent classification workflow starts from the solution knowledge, which can provide information on requirements, use cases, technologies used, network properties, intent users that define and issue the intent request, and requirements. The following, defines the steps to classify an intent:

1. The information provided in the solution knowledge is given as input for identifying the intent solution (e.g., carrier, enterprise, and data center). Intent solutions are reviewed against the existing classification and they can either be used if present or added if not there or removed if not needed, from the classification. (R1-U1).

2. Identify the intent user types (e.g., customer, network operators, service operators, etc.), review existing intent classification and use the intent user type if present, add if it is not there or remove it if not needed (R2-U2).

3. Identify the types of intent (e.g., network intent, customer service intent) and then review existing classification and use/add/remove the intent type (R3-U3).

4. Identify the intent scopes (e.g., connectivity, application) based on the solution knowledge and then review existing classification and use/add/remove the identified intent scope (R4-U4).

5. Identify the network scopes (e.g., campus, radio access) and then review existing classification and either use it or add/remove the identified network scope (R5-U5).

6. Identify the abstractions (e.g., technical, non-technical) and then review existing classification and use/add/remove the abstractions (R6-U6).

7. Identify the life-cycle requirements (e.g., persistent, transient) and then review existing classification and use/add/remove the life-cycle requirements (R7-U7).

8. Identify any new categories and use/add the newly identified categories. New categories can be identified as new domains or applications are emerging, or new areas of concern (e.g., privacy, compliance) might arise, which are not listed in the current methodology.
6.2. Intent Taxonomy

The following taxonomy describes the various intent solutions, intent user types, intent types, intent scopes, network scopes, abstractions and life-cycle and represents the output of the intent classification tables for each of the solutions addressed (i.e. carrier, data center, and enterprise solutions).

The intent scope categories in Figure 2 are shared among the carrier, DC, and enterprise solutions. The abbreviations (Cx) in sections 6.3.2. 6.4.2. are introduced with the scope of fitting as column title in the following tables.
Figure 2 - Intent Taxonomy
6.3. Intent Classification for Carrier Solution

6.3.1. Intent Users and Intent Types

This section addresses step 1, 2, and 3 from Figure 1 and the following table describes the intent users in carrier solutions and intent types with their descriptions for different intent users.

<table>
<thead>
<tr>
<th>Intent User</th>
<th>Intent Type</th>
<th>Intent Type Description</th>
</tr>
</thead>
</table>
| Customer/Subscriber  | Customer    | Customer self-service with SLA and value added service  
| Service Intent       | Service     | Example: Always maintain high quality of service and high bandwidth for gold level subscribers. Operational statement: Measure the network congestion status, give different adaptive parameters to stations of different priority, thus in heavy load situation, make the bandwidth of the high-priority customers guaranteed.  
|                      | Intent       | At the same time ensure the overall utilization of system, improve the overall throughput of the system.                                                                                                               |
| Strategy Intent      | Strategy     | Customer designs models and policy intents to be used by customer service intents. Example: Request reliable service during peak traffic periods for apps of type video.                                             |
|                      | Intent       |                                                                                                                                                                                                                        |
| Network Operator     | Network      | Service provided by network service operator to the customer (e.g. the service operator) Example: Request network service with delay guarantee for access customer A.                                                      |
| Service Intent       | Intent       |                                                                                                                                                                                                                        |
|                      | Network      | Network operator requests network-wide (service underlay or other network-wide)                                                                                                                                                                   |
configuration) or network resource configurations (switches, routers, routing, policies). Includes connectivity, routing, QoS, security, application policies, traffic steering policies, configuration policies, monitoring policies, alarm generation for non-compliance, auto-recovery, etc. Example: Request high priority queueing for traffic of class A.

**Operational Task Intent**

Network operator requests execution of any automated task other than network service intent and network intent (e.g. network migration, server replacements, device replacements, network software upgrades). Example: Request migration of all services in network N to backup path P.

**Strategy Intent**

Network operator designs models, policy intents and workflows to be used by network service intents, network intents and operational task intents. Workflows can automate any tasks that network operator often performed in addition to network service intents and network intents. Example: Ensure the load on any link in the network is not higher than 50%.
<table>
<thead>
<tr>
<th>Service Operator</th>
<th>Customer Service Intent</th>
<th>Service operator’s customer orders, customer service / SLA Example: Provide service S with guaranteed bandwidth for customer A.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Network Service Intent</td>
<td>Service operator’s network orders / network SLA Example: Provide network guarantees in terms of security, low latency and high bandwidth</td>
</tr>
<tr>
<td></td>
<td>Operational Task Intent</td>
<td>Service operator requests execution of any automated task other than customer service intent and network service intent Example: Update service operator portal platforms and their software regularly. Move services from network operator 1 to network operator 2.</td>
</tr>
<tr>
<td></td>
<td>Strategy Intent</td>
<td>Service operator designs models, policy intents and workflows to be used by customer service intents, network service intents and operational task intents. Workflows can automate any tasks that service operator often performed in addition to network service intents and network intents. Example: Request network service guarantee to avoid network congestion during special periods such as black Friday, and Christmas.</td>
</tr>
<tr>
<td>Application Developer</td>
<td>Customer Service Intent</td>
<td>Customer service intent API provided to the application developers Example: API to request network to watch HD video 4K/8K.</td>
</tr>
<tr>
<td>Intent Type</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Network Service Intent</td>
<td>Network service intent API provided to the application developers Example: API to request network service, monitoring and traffic grooming.</td>
<td></td>
</tr>
<tr>
<td>Network Intent</td>
<td>Network intent API provided to the application developers Example: API to request network resources configuration.</td>
<td></td>
</tr>
<tr>
<td>Operational Task Intent</td>
<td>Operational task intent API provided to the application developers. This is for the trusted internal operator/service provider/customer DevOps Example: API to request server migrations.</td>
<td></td>
</tr>
<tr>
<td>Strategy Intent</td>
<td>Application developer designs models, policy and workflows to be used by customer service intents, network service intents and operational task intents. This is for the trusted internal operator/service provider/customer DevOps Example: API to design network load balancing strategies during peak times.</td>
<td></td>
</tr>
</tbody>
</table>

Table 2 - Intent Classification for Carrier Solution
6.3.2. Intent Categories

This subsection addresses step 4 to 7 from Figure 1, and the following are the proposed categories:

- **Intent Scope:** C1=Connectivity, C2=Security/Privacy, C3=Application, C4=QoS
- **Network Scope:**
  - **Network Domain:** C1=Radio Access, C2=Transport Access, C3=Transport Aggregation, C4=Transport Core, C5=Cloud Edge, C6=Cloud Core
  - **Network Function (NF) Scope:** C1=VNFs, C2=PNFs
- **Abstraction (ABS):** C1=Technical (with technical feedback), C2=Non-technical (without technical feedback) see section 5.2.
- **Life-cycle (L-C):** C1=Persistent (full life-cycle), C2=Transient (short lived)

6.3.3. Intent Classification Example

This section depicts an example on how the methodology described in section 6.1. can be used in order to classify intents introduced in the ‘A Multi-Level Approach to IBN’ PoC demonstration [POC-IBN]. This PoC is led by academics carrying research in the area of SDN/NFV and the specific problem they are addressing is to apply the intent concept at different levels that correspond to different stakeholders. For this research work, they considered two types of intents: slice intents and service chain intents.

In this PoC [POC-IBN], a slice intent expresses a request for a network slice with two types of components: a set of top layer virtual functions, and a set of virtual switches and/or routers of L2/L3 VNFs. A service chain intent expressed a request for a service operated through a chain of service components running in L4-L7 virtual functions.

Following the intent classification methodology described step-by-step in section 6.1., the following can be derived:

1. The intent solution for both intents is carrier network.
2. The intent user type is network operator for the slice intent, and service operator for the service chain intent.
3. The type of intent, is a network service intent for the slice intent, and a customer service intent for the service chain intent.
4. The intent scopes are connectivity and application.

5. The network scope is VNF, cloud edge, and cloud core.

6. The abstractions are with technical feedback for the slice intent, and without technical feedback for the service chain intent.

7. The life-cycle is persistent.

The following table shows how to represent this information in a tabular form. The ’X’ in the table refers to the slice intent, and the ’Y’ in the table refers to the service chain intent.
<table>
<thead>
<tr>
<th>Intent User</th>
<th>Intent Type</th>
<th>Intent Scope</th>
<th>NF Scope</th>
<th>Network Scope</th>
<th>ABS</th>
<th>L-C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer / Subscriber</td>
<td>Customer Service Intent</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Strategy Intent</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Network Operator</td>
<td>Network Service Intent</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Network Intent</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Operational Task Intent</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Strategy Intent</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Service Operator</td>
<td>Customer Service Intent</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>Network Service Intent</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Op Task Intent</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Strategy Intent</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 3 - Intent Classification Example for Carrier Solution

<table>
<thead>
<tr>
<th>App Developer</th>
<th>Customer Intent</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Network Service Intent</td>
</tr>
<tr>
<td></td>
<td>Network Intent</td>
</tr>
<tr>
<td></td>
<td>Op Task Intent</td>
</tr>
<tr>
<td></td>
<td>Strategy Intent</td>
</tr>
</tbody>
</table>

### 6.4. Intent Classification for Data Center Network Solutions

#### 6.4.1. Intent Users and Intent Types

The following table describes the intent users in DC network solutions and intent types with their descriptions for different intent users.

<table>
<thead>
<tr>
<th>Intent User</th>
<th>Intent Type</th>
<th>Intent Type Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer / Tenants</td>
<td>Customer Service</td>
<td>Customer self-service via tenant portal. Example: Request GPU computing and storage resources to meet 10k video surveillance services.</td>
</tr>
<tr>
<td></td>
<td>Strategy Intent</td>
<td>This includes models and policy intents designed by customers/tenants to be reused later during instantiation. Example: Request dynamic computing and storage resources of the service in special and daily times.</td>
</tr>
<tr>
<td></td>
<td>Cloud Administrator Intent</td>
<td>Configuration of VMs, DB Servers, app servers, connectivity, communication between VMs. Example: Request connectivity between VMs A, B, and C in network N1.</td>
</tr>
<tr>
<td></td>
<td>Operational Task Intent</td>
<td>Cloud administrator requests execution of any automated task other than cloud management intents and cloud resource management intents. Example: Request upgrade operating system to version X on all VMs in network N1.</td>
</tr>
</tbody>
</table>
Operational statement: an intent to update a system might reconfigure the system topology (connect to a service and to peers), exchange data (update the content), and uphold a certain QoE level (allocate sufficient network resources). The network, thus, carries out the necessary configuration to best serve such an intent; e.g. setting up direct connections between terminals, and allocating fair shares of router queues considering other network services.

**Strategy Intent**
Cloud administrator designs models, policy intents and workflows to be used by other intents. Automate any tasks that administrator often performs, in addition to life-cycle of cloud management intents and cloud management resource intents. Example: In case of emergency, automatically migrate all cloud resources to DC2.

**Underlay Network Administrator**
Service created and provided by the underlay network administrator. Example: Request underlay service between DC1 and DC2 with bandwidth B.

**Underlay Network Intent**
Underlay network administrator requests some DCN-wide underlay network configuration or network resource configurations. Example: Establish and allocate DHCP address pool.

**Operational Task Intent**
Underlay network administrator requests execution of the any automated task other than underlay network service and resource
### Strategy Intent

Underlay network administrator designs models, policy intents & workflows to be used by other intents. Automate any tasks that administrator often performs.

Example: For all traffic flows that need NFV service chaining, restrict the maximum load of any VNF node/container below 50% and the maximum load of any network link below 70%.

<table>
<thead>
<tr>
<th>Application Developer</th>
<th>Intent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cloud Management Intent</td>
<td>Cloud management intent API provided to the application developers. Example: API to request configuration of VMs, or DB Servers.</td>
<td></td>
</tr>
<tr>
<td>Cloud Resource Management Intent</td>
<td>Cloud resource management intent API provided to the application developers. Example: API to request automatic life-cycle management of cloud resources.</td>
<td></td>
</tr>
<tr>
<td>Underlay Network Service Intent</td>
<td>Underlay network service API provided to the application developers. Example: API to request real-time monitoring of device condition.</td>
<td></td>
</tr>
<tr>
<td>Underlay Network Intent</td>
<td>Underlay network resource API provided to the application developers. Example: API to request dynamic management of IPv4 address pool resources.</td>
<td></td>
</tr>
<tr>
<td>Operational Task Intent</td>
<td>Operational task intent API provided to the trusted application developer (internal DevOps). Example: API to request automatic rapid detection of device failures and pre-alarm correlation</td>
<td></td>
</tr>
<tr>
<td>-------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Strategy Intent</td>
<td>Application developer designs models, policy intents and building blocks to be used by other intents. This is for the trusted internal DCN DevOps. Example: API to request load balancing thresholds.</td>
<td></td>
</tr>
</tbody>
</table>

Table 4 - Intent Classification for Data Center Network Solutions
6.4.2. Intent Categories

The following are the proposed categories:

- **Intent Scope**: C1=Connectivity, C2=Security/Privacy, C3=Application, C4=QoS C5=Storage C6=Compute
- **Network Scope**
  - **Network Domain**: DC Network
  - **DCN Network (DCN Net) Scope**: C1=Logical, C2=Physical
  - **DCN Resource (DCN Res) Scope**: C1=Virtual, C2=Physical
- **Abstraction (ABS)**: C1=Technical (with technical feedback), C2=Non-technical (without technical feedback), see section 5.2.
- **Life-cycle (L-C)**: C1=Persistent (full life-cycle), C2=Transient (short lived)

6.4.3. Intent Classification Example

This section depicts an example on how the methodology described in section 6.1. can be used by the research community to classify intents. As mentioned in 6.3.3. a successful use of the classification proposed in this draft is introduced in the ‘A Multi-Level Approach to IBN’ PoC demonstration [POC-IBN]. The PoC is led by academics carrying research in the area of SDN/NFV and the specific problem they are addressing is to apply the intent concept at different levels that correspond to different stakeholders.

For their research work, they considered two types of intents: slice intents and service chain intents. For the data center solution, only the slice intent is relevant.

As already mentioned in section 6.3.3., a slice intent expresses a request for a network slice with two types of components: a set of top layer virtual functions, and a set of virtual switches and/or routers of L2/L3 VNFs.

Following the intent classification methodology described step-by-step in section 6.1., we identify the following:

1. The intent solution is for the data center.
2. The intent user type is the cloud administrator for the slice intent and service chain intent.
3. The type of intent, is a cloud management intent, for the slice intent.
4. The intent scopes are connectivity and application.

5. The network scope is logical, and the resource scope is virtual.

6. The abstractions are with technical feedback for the slice intent.

7. The life-cycle is persistent.

The following table shows how to represent this information in a tabular form, where the 'X' in the table refers to the slice intent.
<table>
<thead>
<tr>
<th>Intent User</th>
<th>Intent Type</th>
<th>Intent Scope</th>
<th>DCN Res</th>
<th>DCN Net</th>
<th>ABS</th>
<th>L-C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer / Tenants</td>
<td>Customer Service Intent</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Strategy Intent</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cloud Admin</td>
<td>Cloud Management Intent</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Cloud Resource Management Intent</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Operational Task Intent</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Strategy Intent</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Underlay Network Admin</td>
<td>Underlay Network Intent</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Underlay Network Resource Intent</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Operational Task Intent</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Strategy Intent</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 5 - Intent Classification Example for Data Center Network Solutions

<table>
<thead>
<tr>
<th>App Developer</th>
<th>Intent</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cloud Management Intent</td>
</tr>
<tr>
<td></td>
<td>Cloud Resource Management Intent</td>
</tr>
<tr>
<td></td>
<td>Underlay Network Intent</td>
</tr>
<tr>
<td></td>
<td>Underlay Network Resource Intent</td>
</tr>
<tr>
<td></td>
<td>Operational Task Intent</td>
</tr>
<tr>
<td></td>
<td>Strategy Intent</td>
</tr>
</tbody>
</table>
6.5. Intent Classification for Enterprise Solution

6.5.1. Intent Users and Intent Types

The following table describes the intent users in enterprise solutions and their intent types.

<table>
<thead>
<tr>
<th>Intent User</th>
<th>Intent Type</th>
<th>Intent Type Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>End-User</td>
<td>Customer Service Intent</td>
<td>Enterprise end-user self-service or applications, enterprise may have multiple types of end-users. Example: Request access to VPN service. Request video conference between end-user A and B.</td>
</tr>
<tr>
<td></td>
<td>Strategy Intent</td>
<td>This includes models and policy intents designed by end-users to be used by end-user intents and their applications. Example: Create a video conference type for a weekly meeting.</td>
</tr>
<tr>
<td>Enterprise Administrator (internal or MSP)</td>
<td>Network Service Intent</td>
<td>Service provided by the administrator to the end-users and their applications. Example: For any end-user of application X, the arrival of hologram objects of all the remote tele-presenters should be synchronised within 50ms to reach the destination viewer for each conversation session. Create management VPN connectivity for type of service A. Operational statement: The job of the network layer is to ensure that the delay is between 50-70ms through</td>
</tr>
<tr>
<td>Intent Classification</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>the routing algorithm. At the same</td>
<td></td>
<td></td>
</tr>
<tr>
<td>time, the node resources need to</td>
<td></td>
<td></td>
</tr>
<tr>
<td>meet the bandwidth requirements of</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4K video conferences.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Network Intent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Administrator requires network wide</td>
</tr>
<tr>
<td>configuration (e.g. underlay,</td>
</tr>
<tr>
<td>campus) or resource configuration</td>
</tr>
<tr>
<td>(switches, routers, policies).</td>
</tr>
<tr>
<td>Example: Configure switches in</td>
</tr>
<tr>
<td>campus network 1 to prioritise</td>
</tr>
<tr>
<td>traffic of type A.</td>
</tr>
<tr>
<td>Configure YouTube as business</td>
</tr>
<tr>
<td>non-relevant.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operational Task Intent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Administrator requests execution of</td>
</tr>
<tr>
<td>any automated task other than</td>
</tr>
<tr>
<td>network service intents and network</td>
</tr>
<tr>
<td>intents.</td>
</tr>
<tr>
<td>Example: Request network security</td>
</tr>
<tr>
<td>automated tasks such as web</td>
</tr>
<tr>
<td>filtering and DDoS cloud protection.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Strategy Intent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Administrator designs models, policy</td>
</tr>
<tr>
<td>intents and workflows to be used by</td>
</tr>
<tr>
<td>other intents. Automate any tasks</td>
</tr>
<tr>
<td>that administrator often performs.</td>
</tr>
<tr>
<td>Example: In case of emergency,</td>
</tr>
<tr>
<td>automatically shift all traffic of</td>
</tr>
<tr>
<td>type A through network N.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Application Developer Intent</th>
</tr>
</thead>
<tbody>
<tr>
<td>End-user service / application</td>
</tr>
<tr>
<td>intent API provided to the</td>
</tr>
<tr>
<td>application developers.</td>
</tr>
<tr>
<td>Example: API for request to open a</td>
</tr>
<tr>
<td>VPN service.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Network Service Intent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network service API provided to</td>
</tr>
<tr>
<td>application developers.</td>
</tr>
<tr>
<td>Example: API for request network</td>
</tr>
</tbody>
</table>

Li, et al. Expires November 18, 2022 [Page 40]
bandwidth and latency for hosting video conference.

Network Intent | Network API provided to application developers. Example: API for request of network devices configuration.

Operational Task Intent | Operational task intent API provided to the trusted application developer (internal DevOps). Example: API for requesting automatic monitoring and interception for network security

Strategy Intent | Application developer designs models, policy intents and building blocks to be used by other intents. This is for the trusted internal DevOps. Example: API for strategy intent in case of emergencies.

Table 6 - Intent Classification for Enterprise Solution

6.5.2. Intent Categories

The following are the proposed categories:
- **Intent Scope**: C1=Connectivity, C2=Security/Privacy, C3=Application, C4=QoS
- **Network (Net) Scope**: C1=Campus, C2=Branch, C3=SD-WAN
- **Abstraction (ABS)**: C1=Technical (with technical feedback), C2=Non-technical (without technical feedback), see section 5.2.
- **Life-cycle (L-C)**: C1=Persistent (full life-cycle), C2=Transient (short lived)
The following is the intent classification table example for enterprise solutions.

<table>
<thead>
<tr>
<th>Intent User</th>
<th>Intent Type</th>
<th>Intent Scope</th>
<th>Net</th>
<th>ABS</th>
<th>L-C</th>
</tr>
</thead>
<tbody>
<tr>
<td>End-User</td>
<td>Customer Service</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Strategy Intent</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enterprise Administrator</td>
<td>Network Service</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Operational Task</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Strategy Intent</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Application Developer</td>
<td>End-User Intent</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Network Service</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Network Intent</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
7. Conclusions

This document is aligned with the RG objectives and supports investigations into intent-based networking by proposing an intent categorization methodology and taxonomy. It brings clarification on what an intent represents for different stakeholders through the proposal of an Intent Classification approach, ensuring that a common understanding among all the participants exists. This, together with the proposed intent taxonomy provides a solid foundation for future intent-related topic discussions within NMRG.

The benefits of this intent classification draft in the research community have been demonstrated through a PoC implementation [POC-IBN] in which the draft’s concepts at different levels corresponding to different stakeholders have been applied to.

8. Security Considerations

This document identifies the security and privacy as categories of the intent scope. The intents could be solely security intents and privacy intents or security can be embedded in the intents that include also connectivity, application, and QoS scope.

Security and privacy scope, is when the intent specifies the security characteristics of the network, customers, or end-users, and privacy for customers and end-users.

More details of these security intents would be described in future documents that specify architecture, functionality, user intents and models. As well, an analysis of the security considerations of the overall intent-based system is provided in section 10 of [CLEMM].

9. IANA Considerations

This document has no actions for IANA.
10. Contributors

The following people all contributed to creating this document:

Contributed significant text:

Xueyuan Sun, China Telecom
Will (Shucheng) Liu, Huawei

Contributed text in early drafts:

Ying Chen, China Unicom
John Strassner, Huawei
Weiping Xu, Huawei
Richard Meade, Huawei

11. Acknowledgments

This document has benefited from reviews, suggestions, comments and proposed text provided by the following members, listed in alphabetical order: Mehdi Bezahaf, Brian E Carpenter, Laurent Ciavaglia, Benoit Claise, Alexander Clemm, Yehia Elkhadib, Jerome Francois, Pedro Andres Aranda Gutierrez, Daniel King, Branislav Meandzija, Bob Natale, Juergen Schoenwaelder, Xiaolin Song, Jeff Tantsura.

We thank to Barbara Martini, Walter Cerroni, Molka Gharbaoui, Davide Borsatti, for contributing with their 'A multi-level approach to IBN' PoC demonstration a first attempt to adopt the intent classification methodology.

12. Informative References


Authors’ Addresses

Chen Li
China Telecom
No.118 Xizhimennei street, Xicheng District
Beijing 100035
P.R. China
Email: lichen.bri@chinatelecom.cn

Olga Havel
Huawei Technologies
Ireland
Email: olga.havel@huawei.com

Adriana Olariu
Huawei Technologies
Ireland
Email: adriana.olariu@huawei.com

Pedro Martinez-Julia
NICT
Japan
Email: pedro@nict.go.jp

Jeferson Campos Nobre
Federal University of Rio Grande do Sul
Porto Alegre
Brazil
Email: jcnobre@inf.ufrgs.br

Diego R. Lopez
Telefonica I+D
Don Ramon de la Cruz, 82
Madrid 28006
Spain
Email: diego.r.lopez@telefonica.com
Abstract

Digital Twin technology has been seen as a rapid adoption technology in Industry 4.0. The application of Digital Twin technology in the networking field is meant to develop various rich network applications and realize efficient and cost effective data driven network management and accelerate network innovation.

This document presents an overview of the concepts of Digital Twin Network, provides the basic definitions and a reference architecture, lists a set of application scenarios, and discusses the benefits and key challenges of such technology.
1. Introduction

The fast growth of network scale and the increased demand placed on these networks require them to accommodate and adapt dynamically to customer needs, implying a significant challenge to network operators. Indeed, network operation and maintenance are becoming more complex due to higher complexity of the managed networks and the sophisticated services they are delivering. As such, providing innovations on network technologies, management and operation will be more and more challenging due to the high risk of interfering with existing services and the higher trial costs if no reliable emulation platforms are available.

A Digital Twin is the real-time representation of a physical entity in the digital world. It has the characteristics of virtual-reality interrelation and real-time interaction, iterative operation and process optimization, full life-cycle and comprehensive data-driven network infrastructure. Currently, digital twin has been widely acknowledged in academic publications. See more in Section 3.

A digital twin for networks platform can be built by applying Digital Twin technologies to networks and creating a virtual image of physical network facilities (called herein, emulation). Basically, the digital twin for networks is an expansion platform of network simulation. The main difference compared to traditional network management systems is the interactive virtual-real mapping and data driven approach to build closed-loop network automation. Therefore, a digital twin network platform is more than an emulation platform or network simulator.

Through the real-time data interaction between the physical network and its twin network(s), the digital twin network platform might help the network designers to achieve more simplification, automatic, resilient, and full life-cycle operation and maintenance. More specifically, the digital twin network can, thus, be used to develop various rich network applications and assess specific behaviors (including network transformation) before actual implementation in the physical network, tweak the network for better optimized behavior, run ‘what-if’ scenarios that cannot be tested and evaluated easily in the physical network. In addition, service impact analysis tasks can also be facilitated.

2. Terminology

2.1. Acronyms & Abbreviations

IBN: Intent-Based Networking
2.2. Definitions

This document makes use of the following terms:

Digital Twin: a virtual instance of a physical system (twin) that is continually updated with the latter’s performance, maintenance, and health status data throughout the physical system’s life cycle.

Digital twin network: a digital twin that is used in the context of networking. This is also called, digital twin for networks. See more in Section 3.3.

3. Introduction and Concepts of Digital Twin Network

3.1. Background of Digital Twin

The concept of the "twin" dates to the National Aeronautics and Space Administration (NASA) Apollo program in the 1970s, where a replica of space vehicles on Earth was built to mirror the condition of the equipment during the mission [Rosen2015].

In 2003, Digital Twin was attributed to John Vickers by Michael Grieves in his product lifecycle management (PLM) course as "virtual digital representation equivalent to physical products" [Grieves2014]. Digital twin can be defined as a virtual instance of a physical system (twin) that is continually updated with the latter’s performance, maintenance, and health status data throughout the physical system’s life cycle [Madni2019]. By providing a living copy of physical system, digital twins bring numerous advantages, such as accelerated business processes, enhanced productivity, and faster innovation with reduced costs. So far, digital twin has been successfully applied in the fields of intelligent manufacturing, smart city, or complex system operation and maintenance to help with not only object design and testing, but also management aspects [Tao2019].
Compared with ‘digital model’ and ‘digital shadow’, the key difference of ‘digital twin’ is the direction of data between the physical and virtual systems [Fuller2020]. Typically, when using a digital twin, the (twin) system is generated and then synchronized using data flows in both directions between physical and digital components, so that control data can be sent, and changes between the physical and digital objectives and systems are automatically represented. This behavior is unlike a ‘digital model’ or ‘digital shadow’, which are usually synchronized manually, lacking of control data, and might not have a full cycle of data integrated.

At present (2022), there is no unified definition of digital twin framework. The industry, scientific research institutions, and standards developing organizations are trying to define a general or domain-specific framework of digital twin. [Natis-Gartner2017] proposed that building a digital twin of a physical entity requires four key elements: model, data, monitoring, and uniqueness. [Tao2019] proposed a five-dimensional framework of digital twin {PE, VE, SS, DD, CN}, in which PE represents physical entity, VE represents virtual entity, SS represents service, DD represents twin data, and CN represents the connection between various components. [ISO-2021] issued a draft standard for digital twin manufacturing system, and proposed a reference framework including data collection domain, device control domain, digital twin domain, and user domain.

3.2. Digital Twin for Networks

Communication networks can provide a solid foundation for implementing various ‘digital twin’ applications. At the same time, in the face of increasing business types, scale and complexity, a network itself also needs to use digital twin technology to seek better solutions beyond physical network. Since 2017, the application of digital twin technology in the field of communication networks has gradually been researched. Some examples are listed below.

In academy, [Dong2019] established the digital twin of 5G mobile edge computing (MEC) network, used the twin offline to train the resource allocation optimization and normalized energy-saving algorithm based on reinforcement learning, and then updated the scheme to MEC network. [Dai2020] established a digital twin edge network for mobile edge computing system, in which a twin edge server is used to evaluate the state of entity server, and the twin mobile edge computing system provides data for training offloading strategy. [Nguyen2021] discusses how to deploy a digital twin for complex 5G networks. [Hong2021] presents a digital twin platform towards automatic and intelligent management for data center networks, and then proposes a simplified the workflows of network service.
management. In addition, international workshops dedicated to
digital twin in network field have already appeared, such as IEEE
DTPI 2021 - Digital Twin Network Online Session [DTPI2021], or are
being proposed such as IEEE NOMS 2022 - TNT workshop [TNT2022].

Although the application of digital twin technology in networking has
started, the research of digital twin for networks technology is
still in its infancy. Current applications focus on specific
scenarios (such as network optimization), where network digital twin
is just used as a network simulation tool to solve the problem of
network operation and maintenance. Combined with the characteristics
of digital twin technology and its application in other industries,
this document believes that digital twin network can be regarded as
an organic whole of the overall network system and become a general
architecture involving the whole life cycle of physical network in
the future, serving the application of network innovative
technologies such as network planning, construction, maintenance and
optimization, improving the automation and intelligence level of the
network.

3.3. Definition of Digital Twin Network

So far, there is no standard definition of "digital twin network"
within the networking industry. This document defines "digital twin
network" as a virtual representation of the physical network. Such
virtual representation of the network is meant to be used to analyze,
diagnose, emulate, and then control the physical network based on
data, models, and interfaces. To that aim, a real-time and
interactive mapping is required between the physical network and its
virtual twin network.

Referring the characteristics of digital twin in other industries and
the characteristics of the networking itself, the digital twin
network should involve four key elements: data, mapping, models and
interfaces as shown in Figure 1.
Figure 1: Key Elements of Digital Twin Network

Data: A digital twin network should maintain historical data and/or real-time data (configuration data, operational state data, topology data, trace data, metric data, process data, etc.) about its real-world twin (i.e., physical network) that are required by the models to represent and understand the states and behaviors of the real-world twin.

The data is characterized as the single source of "truth" and populated in the data repository, which provides timely and accurate data service support for building various models.

Models: Techniques that involve collecting data from one or more sources in the real-world twin and developing a comprehensive representation of the data (e.g., system, entity, process) using specific models. These models are used as emulation and diagnosis basis to provide dynamics and elements on how the live physical network operates and generates reasoning data utilized for decision-making.

Various models such as service models, data models, dataset models, or knowledge graph can be used to represent the physical network assets and, then, instantiated to serve various network applications.

Interfaces: Standardized interfaces can ensure the interoperability of digital twin network. There are two major types of interfaces:

* The interface between the digital twin network platform and the physical network infrastructure.
* The interface between digital twin network platform and applications.

The former provides real-time data collection and control on the physical network. The latter helps in delivering application requests to the digital twin network platform and exposing the various platform capabilities to applications.

Mapping: Used to identify the digital twin and the underlying entities and establish a real-time interactive relation between the physical network and the twin network or between two twin networks. The mapping can be:

* One to one (pairing, vertical): Synchronize between a physical network and its virtual twin network with continuous flows.

* One to many (coupling, horizontal): Synchronize among virtual twin networks with occasional data exchange.

Such mappings provide a good visibility of actual status, making the digital twin suitable to analyze and understand what is going on in the physical network. It also allows using the digital twin to optimize the performance and maintenance of the physical network.

The digital twin network constructed based on the four core technology elements can analyze, diagnose, emulate, and control the physical network in its whole life cycle with the help of optimization algorithms, management methods, and expert knowledge. One of the objectives of such control is to master the digital twin network environment and its elements to derive the required system behavior, e.g., provide:

* repeatability: that is the capacity to replicate network conditions on-demand.

* reproducibility: i.e., the ability to replay successions of events, possibly under controlled variations.
Note: Real-time interaction is not always mandatory for all twins. When testing some configuration changes or trying some innovative techniques, the digital twins can behave as a simulation platform without the need of real-time telemetry data. And even in this scenario, it is better to have interactive mapping capability so that the validated changes can be tested in real network whenever required by the testers. In most other cases (e.g., network optimization, network fault recovery), real-time interaction between virtual and real network is mandatory. This way, digital twin network can help achieve the goal of autonomous network or self-driven network.


Digital twin network can help enabling closed-loop network management across the entire lifecycle, from deployment and emulation, to visualized assessment, physical deployment, and continuous verification. By doing so, network operators and end-users to some extent, as allowed by specific application interfaces, can maintain a global, systemic, and consistent view of the network. Also, network operators and/or enterprise user can safely exercise the enforcement of network planning policies, deployment procedures, etc., without jeopardizing the daily operation of the physical network.

The main difference between digital twin network and simulation platform is the use of interactive virtual-real mapping to build closed-loop network automation. Simulation platforms are the predecessor of the digital twin network, one example of such a simulation platform is network simulator [NS-3], which can be seen as a variant of digital twin network but with low fidelity and lacking for interactive interfaces to the real network. Compared with those classical approaches, key benefits of digital twin network can be summarized as follows:

1) Using real-time data to establish high fidelity twins, the effectiveness of network simulation is higher; then the simulation cost will be relatively low.

2) The impact and risk on running networks is low when automatically applying configuration/policy changes after the full analysis and required verifications (e.g., service impact analysis) within the twin network.

3) The faults of the physical network can be automatically captured by analyzing real-time data, then the correction strategy can be distributed to the physical network elements after conducting adequate analysis within the twins to complete the closed-loop automatic fault repair.
The following subsections further elaborate such benefits in details.

4.1. Optimized Network Total Cost of Operation

Large scale networks are complex to operate. Since there is no effective platform for simulation, network optimization designs have to be tested on the physical network at the cost of jeopardizing its daily operation and possibly degrading the quality of the services supported by the network. Such assessment greatly increases network operator’s Operational Expenditure (OPEX) budgets too.

With a digital twin network platform, network operators can safely emulate candidate optimization solutions before deploying them in the physical network. In addition, operator’s OPEX on the real physical network deployment will be greatly decreased accordingly at the cost of the complexity of the assessment and the resources involved.

4.2. Optimized Decision Making

Traditional network operation and management mainly focus on deploying and managing running services, but hardly support predictive maintenance techniques.

Digital twin network can combine data acquisition, big data processing, and AI modeling to assess the status of the network, but also to predict future trends, and better organize predictive maintenance. The ability to reproduce network behaviors under various conditions facilitates the corresponding assessment of the various evolution options as often as required.

4.3. Safer Assessment of Innovative Network Capabilities

Testing a new feature in an operational network is not only complex, but also extremely risky. Service impact analysis is required to be adequately achieved prior to effective activation of a new feature.

Digital twin network can greatly help assessing innovative network capabilities without jeopardizing the daily operation of the physical network. In addition, it helps researchers to explore network innovation (e.g., new network protocols, network AI/ML applications) efficiently, and network operators to deploy new technologies quickly with lower risks. Take AI/ML application as example, it is a conflict between the continuous high reliability requirement (i.e., 99.999%) and the slow learning speed or phase-in learning steps of AI/ML algorithms. With digital twin network, AI/ML can complete the learning and training with the sufficient data before deploying the model in the real network. This would encourage more network AI innovations in future networks.
4.4. Privacy and Regulatory Compliance

The requirements on data confidentiality and privacy on network providers increase the complexity of network management, as decisions made by computation logics such as an SDN controller may rely upon the packet payloads. As a result, the improvement of data-driven management requires complementary techniques that can provide a strict control based upon security mechanisms to guarantee data privacy protection and regulatory compliance. This may range from flow identification (using the archetypal five-tuple of addresses, ports and protocol) to techniques requiring some degree of payload inspection, all of them considered suitable to be associated to an individual person, and hence requiring strong protection and/or data anonymization mechanisms.

With strong modeling capability provided by the digital twin network, very limited real data (if at all) will be needed to achieve similar or even higher level of data-driven intelligent analysis. This way, a lower demand of sensitive data will permit to satisfy privacy requirements and simplify the use of privacy-preserving techniques for data-driven operation.

4.5. Customized Network Operation Training

Network architectures can be complex, and their operation requires expert personnel. Digital twin network offers an opportunity to train staff for customized networks and specific user needs. Two salient examples are the application of new network architectures and protocols or the use of "cyber-ranges" to train security experts in threat detection and mitigation.

5. Challenges to Build Digital Twin Network

According to [Hu2021], the main challenges in building and maintaining digital twins can be summarized as the following five aspects:

* Data acquisition and processing
* High-fidelity modeling
* Real-time, two-way connection between the virtual and the real twins
* Unified development platform and tools
* Environmental coupling technologies
Compared with other industrial fields, digital twin in networking field has its unique characteristics. On one hand, network elements and system have higher level of digitalization, which implies that data acquisition and virtual-real connection are relatively easy to achieve. On the other hand, there are many kinds of network elements and topologies in the network field; and the complex giant system of network carries a variety of business services. So, the construction of a digital twin network system needs to consider the following major challenges:

Large scale challenge: A digital twin of large-scale networks will significantly increase the complexity of data acquisition and storage, the design and implementation of relevant models. The requirements of software and hardware of the digital twin network system will be even more constraining. Therefore, efficient and low cost tools in various fields should be required. Take data as an example, massive network data can help achieve more accurate models. However, to lower the cost of virtual-real communication and data storage, efficient tools on data collection and data compression methods must be used.

Interoperability: Due to the inconsistency of technical implementations and the heterogeneity of vendor technologies, it is difficult to establish a unified digital twin network system with a common technology in a network domain. Therefore, it is needed firstly to propose a unified architecture of digital twin network, in which all components and functionalities are clear to all stakeholders; then define standardized and unified interfaces to connect all network twins via ensuring necessary compatibility.

Data modeling difficulties: Based on large-scale network data, data modeling should not only focus on ensuring the accuracy of model functions, but also has to consider the flexibility and scalability to compose and extend as required to support large scale and multi-purpose applications. Balancing these requirements further increases the complexity of building efficient and hierarchical functional data models. As an optional solution, straightforwardly clone the real network using virtualized resources is feasible to build the twin network when the network scale is relatively small. However, it will be of unaffordable resource cost for larger scales network. In this case, network modeling using mathematical abstraction or leveraging the AI algorithms will be more suitable solutions.

Real-time requirements: Network services normally have real-time requirements, the processing of model simulation and verification through a digital twin network will increase the service latency. Meanwhile, the real-time requirements will further increase
performance requirements on the system software and hardware. Moreover, it is also challenge to keep network digital twins in sync given the nature of distributed systems and propagation delays. To address these requirements, the function and process of the data model need to be based on automated processing mechanism under various network application scenarios. On the one hand, it is needed to design a simplified process to reduce the time cost for tasks in network twin as much as possible; on the other hand, it is recommended to define the real-time requirements of different applications, and then match the corresponding computing resources and suitable solutions as needed to complete the task processing in the twin.

Security risks: A digital twin network has to synchronize all or subset of the data related to involved physical networks in real time, which inevitably augments the attack surface, with a higher risk of information leakage, in particular. On one hand, it is mandatory to design more secure data mechanism leveraging legacy data protection methods, as well as innovative technologies such as block chain. On the other hand, the system design can limit the data (especially raw data) requirement on building digital twin network, leveraging innovative modeling technologies such as federal learning.

In brief, to address the above listed challenges, it is important to firstly propose a unified architecture of digital twin network, which defines the main functional components and interfaces (Section 6). Then, relying upon such an architecture, it is required to continue researching on the key enabling technologies including data acquisition, data storage, data modeling, interface standardization, and security assurance.


Based on the definition of the key digital twin network technology elements introduced in Section 3.3, a digital twin network architecture is depicted in Figure 2. This digital twin network architecture is broken down into three layers: Application Layer, Digital Twin Layer, and Physical Network Layer.
Physical Network: All or subset of network elements in the physical network exchange network data and control messages with a network digital twin instance, through twin-physical control interfaces. The physical network can be a mobile access network, a transport network, a mobile core, a backbone, etc. The physical network can also be a data center network, a campus enterprise network, an industrial Internet of Things, etc.

The physical network can span across a single network administrative domain or multiple network administrative domains.

This document focuses on the IETF related physical network such as IP bearer network and datacenter network.

Digital Twin Layer: This layer includes three key subsystems: Data Repository subsystem, Service Mapping Models subsystem, and Digital Twin Network Management subsystem.
One or multiple digital twin network instances can be built and maintained:

* Data Repository subsystem is responsible for collecting and storing various network data for building various models by collecting and updating the real-time operational data of various network elements through the twin southbound interface, and providing data services (e.g., fast retrieval, concurrent conflict handling, batch service) and unified interfaces to Service Mapping Models subsystem.

* Service Mapping Models complete data modeling, provide data model instances for various network applications, and maximizes the agility and programmability of network services. The data models include two major types: basic and functional models.

- Basic models refer to the network element model(s) and network topology model(s) of the network digital twin based on the basic configuration, environment information, operational state, link topology and other information of the network element(s), to complete the real-time accurate characterization of the physical network.

- Functional models refer to various data models used for network analysis, emulation, diagnosis, prediction, assurance, etc. The functional models can be constructed and expanded by multiple dimensions: by network type, there can be models serving for a single or multiple network domains; by function type, it can be divided into state monitoring, traffic analysis, security exercise, fault diagnosis, quality assurance and other models; by network lifecycle management, it can be divided into planning, construction, maintenance, optimization and operation. Functional models can also be divided into general models and special-purpose models. Specifically, multiple dimensions can be combined to create a data model for more specific application scenarios.

New applications might need new functional models that do not exist yet. If a new model is needed, ‘Service Mapping Models’ subsystem will be triggered to help creating new models based on data retrieved from ‘Data Repository’.
* Digital Twin Network Management fulfills the management function of digital twin network, records the life-cycle transactions of the twin entity, monitors the performance and resource consumption of the twin entity or even of individual models, visualizes and controls various elements of the network digital twin, including topology management, model management and security management.

Notes: 'Data collection' and 'change control' are regarded as southbound interfaces between virtual and physical network. From implementation perspective, they can optionally form a sub-layer or sub-system to provide common functionalities of data collection and change control, enabled by a specific infrastructure supporting bi-directional flows and facilitating data aggregation, action translation, pre-processing and ontologies.

Application Layer: Various applications (e.g., Operations, Administration, and Maintenance (OAM)) can effectively run over a digital twin network platform to implement either conventional or innovative network operations, with low cost and less service impact on real networks. Network applications make requests that need to be addressed by the digital twin network. Such requests are exchanged through a northbound interface, so they are applied by service emulation at the appropriate twin instance(s).

7. Interaction with IBN

Implementing Intent-Based Networking (IBN) is an innovative technology for life-cycle network management. Future networks will be possibly Intent-based, which means that users can input their abstract ‘intent’ to the network, instead of detailed policies or configurations on the network devices. [I-D.irtf-nmrg-ibn-concepts-definitions] clarifies the concept of "Intent" and provides an overview of IBN functionalities. The key characteristic of an IBN system is that user intent can be assured automatically via continuously adjusting the policies and validating the real-time situation.

IBN can be envisaged in a digital twin network context to show how digital twin network improves the efficiency of deploying network innovation. To lower the impact on real networks, several rounds of adjustment and validation can be emulated on the digital twin network platform instead of directly on physical network. Therefore, digital twin network can be an important enabler platform to implement IBN systems and speed up their deployment.
8. Sample Application Scenarios

Digital twin network can be applied to solve different problems in network management and operation.

8.1. Human Training

The usual approach to network OAM with procedures applied by humans is open to errors in all these procedures, with impact in network availability and resilience. Response procedures and actions for most relevant operational requests and incidents are commonly defined to reduce errors to a minimum. The progressive automation of these procedures, such as predictive control or closed-loop management, reduce the faults and response time, but still there is the need of a human-in-the-loop for multiples actions. These processes are not intuitive and require training to learn how to respond.

The use of digital twin network for this purpose in different network management activities will improve the operators performance. One common example is cybersecurity incident handling, where "cyber-range" exercises are executed periodically to train security practitioners. Digital twin network will offer realistic environments, fitted to the real production networks.

8.2. Machine Learning Training

Machine Learning requires data and their context to be available in order to apply it. A common approach in the network management environment has been to simulate or import data in a specific environment (the ML developer lab), where they are used to train the selected model, while later, when the model is deployed in production, re-train or adjust to the production environment context. This demands a specific adaption period.

Digital twin network simplifies the complete ML lifecycle development by providing a realistic environment, including network topologies, to generate the data required in a well-aligned context. Dataset generated belongs to the digital twin network and not to the production network, allowing information access by third parties, without impacting data privacy.
8.3. DevOps-Oriented Certification

The potential application of CI/CD models network management operations increases the risk associated to deployment of non-validated updates, what conflicts with the goal of the certification requirements applied by network service providers. A solution for addressing these certification requirements is to verify the specific impacts of updates on service assurance and SLAs using a digital twin network environment replicating the network particularities, as a previous step to production release.

Digital twin network control functional block supports such dynamic mechanisms required by DevOps procedures.

8.4. Network Fuzzing

Network management dependency on programmability increases systems complexity. The behavior of new protocol stacks, API parameters, and interactions among complex software components are examples that imply higher risk to errors or vulnerabilities in software and configuration.

Digital twin network allows to apply fuzzing testing techniques on a twin network environment, with interactions and conditions similar to the production network, permitting to identify and solve vulnerabilities, bugs and zero-days attacks before production delivery.

9. Research Perspectives: A Summary

Research on digital twin network has just started. This document presents an overview of the digital twin network concepts and reference architecture. Looking forward, further elaboration on digital twin network scenarios, requirements, architecture, and key enabling technologies should be investigated by the industry, so as to accelerate the implementation and deployment of digital twin network.

10. Security Considerations

This document describes concepts and definitions of digital twin network. As such, the following security considerations remain high level, i.e., in the form of principles, guidelines or requirements.

Security considerations of the digital twin network include:

* Secure the digital twin system itself.
* Data privacy protection.

Securing the digital twin network system aims at making the digital twin system operationally secure by implementing security mechanisms and applying security best practices. In the context of digital twin network, such mechanisms and practices may consist in data verification and model validation, mapping operations between physical network and digital counterpart network by authenticated and authorized users only.

Synchronizing the data between the physical and the digital twin networks may increase the risk of sensitive data and information leakage. Strict control and security mechanisms must be provided and enabled to prevent data leaks.

11. Acknowledgements

Many thanks to the NMRG participants for their comments and reviews. Thanks to Daniel King, Quifang Ma, Laurent Ciavaglia, Jerome Francois, Jordi Paillisse, Luis Miguel Contreras Murillo, Alexander Clemm, Qiao Xiang, Ramin Sadre, Pedro Martinez-Julia, Wei Wang, Zongpeng Du, and Peng Liu.

Diego Lopez and Antonio Pastor were partly supported by the European Commission under Horizon 2020 grant agreement no. 833685 (SPIDER), and grant agreement no. 871808 (INSPIRE-5Gplus).

12. IANA Considerations

This document has no requests to IANA.

13. Open issues

* The draft focuses on concept and architecture of digital twin network, not including enabling technologies. Actually, each 'enabling technology' is worth of a separate draft to study in details in future. A decision is needed that whether to add a section to describe the enabling technologies in brief.

* Related to above issue, if section of enabling technologies is added, recent technologies (e.g. Network connectivity, Real-time data communication, Collaboration management, conflict detection and resolution, etc.) recently discussed in the IRTF/IETF should be described.

* In section of 'Sample Application Scenarios', to dig deeper into one or two use cases.
* On the research side, the idea behind digital twin networks is reminiscent of earlier work from the 1990s that should be referenced/acknowledged. Examples include the Shadow MIB concept, Inductive Modeling Technique, etc.

14. Informative References


[I-D.irtf-nmrg-ibn-concepts-definitions]


Appendix A. Change Logs

v06 - v07: Addressed reviewer’s comments from adoption call, including below major changes.

* Resequeced the sections via adding more subsections on concepts of digital twin network, removing the ‘Requirements Language’ section, and moving ahead the ‘Challenges’ section.

* Cited more papers, or industrial information on digital twin concepts and digital twin for networks.

* Added more information on describing the challenges and key characteristics digital twin network.

* Removed previous open issue on investigating related digital twin network work and identify the differences and commonalities, and added several new open issues for future studies.

* Other Editorial changes.

v05 - v06: Addressed comments from meeting and maillist, to request adoption call.

* Remove acronym DTN to avoid conflict with ‘Delay Tolerant Network’;

* Elaborate the description of Digital Twin Network architecture that supports multiple instances;

* Other Editorial changes.

04 - v05

* Clarify the difference between digital twin network platform and traditional network management system;

* Add more references of researches on applying digital twin to network field;

* Clarify the benefit of ’Privacy and Regulatory Compliance’;

* Refine the description of reference architecture;

* Other Editorial changes.

v03 - v04
* Update data definition and models definitions to clarify their difference.

* Remove the orchestration element and consolidated into control functionality building block in the digital twin network.

* Clarify the mapping relation (one to one, and one to many) in the mapping definition.

* Add explanation text for continuous verification.

v02 - v03

* Split interaction with IBN part as a separate section.

* Fill security section;

* Clarify the motivation in the introduction section;

* Use new boilerplate for requirements language section;

* Key elements definition update.

* Other editorial changes.

* Add open issues section.

* Add section on application scenarios.

Authors’ Addresses

Cheng Zhou
China Mobile
Beijing
100053
China
Email: zhouchengyjy@chinamobile.com

Hongwei Yang
China Mobile
Beijing
100053
China
Email: yanghongwei@chinamobile.com
Xiaodong Duan  
China Mobile  
Beijing  
100053  
China  
Email: duanxiaodong@chinamobile.com

Diego Lopez  
Telefonica I+D  
Seville  
Spain  
Email: diego.r.lopez@telefonica.com

Antonio Pastor  
Telefonica I+D  
Madrid  
Spain  
Email: antonio.pastorperales@telefonica.com

Qin Wu  
Huawei  
101 Software Avenue, Yuhua District  
Nanjing  
Jiangsu, 210012  
China  
Email: bill.wu@huawei.com

Mohamed Boucadair  
Orange  
Rennes 35000  
France  
Email: mohamed.boucadair@orange.com

Christian Jacquenet  
Orange  
Rennes 35000  
France  
Email: christian.jacquenet@orange.com