OPSAWG
Internet-Draft
Intended status: Informational
Expires: July 6, 2021

B. Claise
Cisco Systems, Inc.
J. Quilbeuf
Independent
D. Lopez
Telefonica I+D
D. Voyer
Bell Canada
T. Arumugam
Cisco Systems, Inc.

January 2, 2021

Service Assurance for Intent-based Networking Architecture
draft-claise-opsawg-service-assurance-architecture-04

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

Copyright Notice

Table of Contents

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+ [I-D.ietf-opsawg-tacacs], 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 as 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.

```
          +-----------------+
          | Service         |
          | Configuration   |<--------------------+
          | Orchestrator    |                     |
          +-----------------+                     |
             |            |                       |
             |            | Network               |
             |            | Service               | Feedback
             |            | Instance              | Loop
             |            | Configuration         |
             |            |                       |
             |            V                       |
             |        +-----------------+       +-------------------+
             |        | SAIN            |       | SAIN              |
             |        | Orchestrator    |       | Collector         |
             |        +-----------------+       +-------------------+
             |            |                        ^
             |            | Configuration          | Health Status
             |            | (assurance graph)      | (Score + Symptoms)
             |            V                        | Streamed
             |     +-------------------+           | via Telemetry
             |     |+-------------------+          |
             |     ||+-------------------+         |
             |     +|| SAIN              |---------+
             |      +| agent             |
             |       +-------------------+
             |               ^ ^ ^
             |               | | |
             |               | | |  Metric Collection
             V               V V V
         +-------------------------------------------------------------+
         | Monitored Entities                                          |
         |                                                             |
         +-------------------------------------------------------------+
```

Figure 1: SAIN Architecture

In order to produce the score assigned to a service instance, the architecture performs the following tasks:

- o 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.

- o Stream (via telemetry [RFC8641]) operational and config metric values when possible, else continuously poll.

- o 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.

- o TThe 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.

- o 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.

- o 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.

```
                      +------------------+
                      | Tunnel           |
                      | Service Instance |
                      +-----------------+
                                |
            +-------------------+-------------------+
            |                   |                   |
     +-------------+     +-------------+     +--------------+
     | Peer1       |     | Peer2       |     | IP           |
     | Tunnel      |     | Tunnel      |     | Connectivity |
     | Interface   |     | Interface   |     |              |
     +-------------+     +-------------+     +--------------}
            |                   |                 |
     +-------------+     +-------------+     +-------------+
     | Peer1       |     | Peer2       |     | IS-IS       |
     | Physical    |     | Physical    |     | Routing     |
     | Interface   |     | Interface   |     | Protocol    |
     +-------------+     +-------------+     +-------------+
            |                   |
     +-------------+     +-------------+
     |             |     |             |
     | Peer1       |     | Peer2       |
     | Device      |     | Device      |
     +-------------+     +-------------+
```

Figure 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:

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

- o 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:

- o 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.

- o 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 want 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:

- o 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.

- o 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 service 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 Orchesttrator, the SAIN Collector, as well as the SAIN Orchestrator. This garantees 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:

- o Keep the previous assurance-graph-last-change value (let's call it time T)
- o Run through all subservice instance and process the subservice instances for which the last-change is newer that the time T
- o 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

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

[RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <https://www.rfc-editor.org/info/rfc5905>.

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

8.2. Informative References

[I-D.claise-opsawg-service-assurance-yang]
Claise, B. and J. Quilbeuf, "Service Assurance for Intent-based Networking Architecture", February 2020.

[I-D.ietf-opsawg-tacacs]
Dahm, T., Ota, A., dcmgash@cisco.com, d., Carrel, D., and L. Grant, "The TACACS+ Protocol", draft-ietf-opsawg-tacacs-18 (work in progress), March 2020.

[RFC2865] Rigney, C., Willens, S., Rubens, A., and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, DOI 10.17487/RFC2865, June 2000, <https://www.rfc-editor.org/info/rfc2865>.

[RFC3164] Lonvick, C., "The BSD Syslog Protocol", RFC 3164, DOI 10.17487/RFC3164, August 2001, <https://www.rfc-editor.org/info/rfc3164>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <https://www.rfc-editor.org/info/rfc6241>.

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <https://www.rfc-editor.org/info/rfc6242>.

[RFC7011] Claise, B., Ed., Trammell, B., Ed., and P. Aitken, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information", STD 77, RFC 7011, DOI 10.17487/RFC7011, September 2013, <https://www.rfc-editor.org/info/rfc7011>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

[RFC8199] Bogdanovic, D., Claise, B., and C. Moberg, "YANG Module Classification", RFC 8199, DOI 10.17487/RFC8199, July 2017, <https://www.rfc-editor.org/info/rfc8199>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[RFC8641] Clemm, A. and E. Voit, "Subscription to YANG Notifications for Datastore Updates", RFC 8641, DOI 10.17487/RFC8641, September 2019, <https://www.rfc-editor.org/info/rfc8641>.

Appendix A. Changes between revisions

v02 - v03

- Timing Concepts
- New Assurance Graph Generation

v01 - v02

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

v00 - v01

- Terminology clarifications
- 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
Cisco Systems, Inc.
De Kleetlaan 6a b1
1831 Diegem
Belgium

Email: bclaise@cisco.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

Email: tarumuga@cisco.com

OPSAWG
Internet-Draft
Intended status: Standards Track
Expires: July 6, 2021

B. Claise
Cisco Systems, Inc.
J. Quilbeuf
Independent
P. Lucente
NTT
P. Fasano
TIM S.p.A
T. Arumugam
Cisco Systems, Inc.
January 2, 2021

YANG Modules for Service Assurance
draft-claise-opsawg-service-assurance-yang-06

Abstract

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

Status of This Memo

Copyright Notice

Table of Contents

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:

- o machine readable
- o vendor independent
- o 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:

- o 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.
- o 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 ce 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
    (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.
```

```
    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[type=current()/../type]/id";
      }
      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). T
hese 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 no
t
            necessary per se, this would get the entries sorted by start-date-tim
e
```

```
            for easy consumption.";
          uses symptom;
        }
      }
      container dependencies {
        description
          "configure the dependencies between the subservices, along with their t
ypes.";
        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-ass
urance: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()/
../type]/id
              +--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).
```

```
    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-ass
urance: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. Subservice Extension: ietf-service-assurance-interface YANG module

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-ass
urance: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-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 InterafaceHealthy 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;
  namespace "urn:ietf:params:xml:ns:yang:ietf-service-assurance-interface";
  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-ass
urance: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-ass
urance: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                                                      ide
ntityref
        +--rw id                                                        str
ing
        +--ro last-change?                                              yan
g:date-and-time
        +--ro label?                                                    str
ing
        +--rw under-maintenance?                                        boo
lean
        +--rw maintenance-contact                                       str
ing
        +--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?                 str
ing
        |  +--:(service-assurance-interface:interface)
        |  |  +--rw service-assurance-interface:interface?              str
ing
        |  +--:(example-service-assurance-device-acme:acme-device-idty)
        |     +--rw example-service-assurance-device-acme:acme-device-idty
        |        +--rw example-service-assurance-device-acme:device?
       string
        |        +--rw example-service-assurance-device-acme:acme-specific-parame
ter?   string
        +--ro health-score?                                             uin
t8
        +--ro symptoms-history-start?                                   yan
g: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
```

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-ass
urance: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-idt
y')";
           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 sepcific 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:

```
URI: urn:ietf:params:xml:ns:yang:ietf-service-assurance
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-service-assurance-device
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-service-assurance-interface
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 the following registrations are requested:

```
name:       ietf-service-assurance
namespace:  urn:ietf:params:xml:ns:yang:ietf-service-assurance
prefix:     inc
reference:  RFC XXXX

name:       ietf-service-assurance-device
namespace:  urn:ietf:params:xml:ns:yang:ietf-service-assurance-device
prefix:     inc
reference:  RFC XXXX

name:       ietf-service-assurance-interface
namespace:  urn:ietf:params:xml:ns:yang: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, B. and J. Quilbeuf, "draft-claise-opsawg-service-assurance-architecture", 2020.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <https://www.rfc-editor.org/info/rfc6241>.

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <https://www.rfc-editor.org/info/rfc6242>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

[RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <https://www.rfc-editor.org/info/rfc8341>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

11.2. Informative References

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

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/info/rfc3688>.

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

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <https://www.rfc-editor.org/info/rfc8340>.

Appendix A. Changes between revisions

v04 - v05

- Added the concept of symptoms-history-start
- Changed label to description, under symptoms. This was confusing as there was two labels in the models

v03 - v04

- Add the interface subservice, with two parameters

v02 - v03

- Added the maintenace 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
Cisco Systems, Inc.
De Kleetlaan 6a b1
1831 Diegem
Belgium

Email: bclaise@cisco.com

Jean Quilbeuf
Independent

Email: jean@quilbeuf.net

Paolo Lucente
NTT
Siriusdreef 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

NMRG LM. Contreras
Internet-Draft Telefonica
Intended status: Informational July 13, 2020
Expires: January 14, 2021

Interconnection Intents
draft-contreras-nmrg-interconnection-intents-00

Abstract

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

Table of Contents

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.

New capabilities based on network programmability and virtualization are producing service situations where a connectivity only approach is not sufficient. The availability of computing capabilities internal to the networks, or attached to them, enable new scenarios where those capabilities can be consumed through the advertising or exposing of these execution environments from the perspective of the raw infrastructure (i.e., compute, storage and associated networking). In addition or complementary to that, even service function could be advertised in order to make them available for interconnection.

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 advance 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 multi providers or domains will become the norm.

Complex network services leveraging on virtualization solutions and leveraging on different infrastructure environments pertaining to distinct administrative domains, that is, 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.

3. Interconnection intents

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 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 enable IP traffic interchange as in traditional peering

- to facilitate whatever combination of all of them

4. Interconnection intent structure

To be done.

5. Security Considerations

To be done.

6. IANA Considerations

This draft does not include any IANA considerations

Acknowledgments

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

Author's Address

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/

NMRG
Internet-Draft
Intended status: Informational
Expires: May 6, 2021

LM. Contreras
Telefonica
P. Demestichas
WINGS
J. Tantsura
Apstra, Inc.

November 2, 2020

IETF Network Slice Intent
draft-contreras-nmrg-transport-slice-intent-04

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

Copyright Notice

Table of Contents

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.nsdt-teas-ietf-network-slice-definition]. The following paragraph is directly taken from it: "An IETF Network Slice is a logical network topology connecting a number of endpoints with a set of shared or dedicated network resources, that are used to satisfy specific Service Level Objectives (SLOs)."

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 should meet and outcomes that the network is supposed to deliver, without specifying how to achieve them. Those goals and outcomes are defined in a manner that is purely declarative - they specify what to accomplish, not how to achieve it."

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 intent 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.

3. 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].

4. Mechanisms for translating IETF network slice intents

This section describes approaches for implementing mechanisms to translate IETF network slice intents.

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.

- o 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).

- o 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

[GSMA] "Generic Network Slice Template, version 3.0", NG.116 , May 2020.

[I-D.contreras-teas-slice-nbi]
Contreras, L., Homma, S., and J. Ordonez-Lucena, "IETF Network Slice use cases and attributes for Northbound Interface of controller", draft-contreras-teas-slice-nbi-03 (work in progress), October 2020.

[I-D.irtf-nmrg-ibn-concepts-definitions]
Clemm, A., Ciavaglia, L., Granville, L., and J. Tantsura, "Intent-Based Networking - Concepts and Definitions", draft-irtf-nmrg-ibn-concepts-definitions-02 (work in progress), September 2020.

[I-D.nsdt-teas-ietf-network-slice-definition]
Rokui, R., Homma, S., Makhijani, K., Contreras, L., and J. Tantsura, "Definition of IETF Network Slices", draft-nsdt-teas-ietf-network-slice-definition-00 (work in progress), October 2020.

[TS28.541]
"TS 28.541 Management and orchestration; 5G Network Resource Model (NRM); Stage 2 and stage 3 (Release 16) V16.2.0.", 3GPP TS 28.541 V16.2.0 , September 2019.

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 Gkiolias (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
Apstra, Inc.

Email: jefftant.ietf@gmail.com

Network Working Group
Internet Draft
Intended status: Informational
Expires: July 2021

C. Li
China Telecom
O. Havel
W. Liu
A. Olariu
Huawei Technologies
P. Martinez-Julia
NICT
J. Nobre
UFRGS
D. Lopez
Telefonica, I+D
January 22, 2021

Intent Classification
draft-irtf-nmrg-ibn-intent-classification-02

Status of this Memo

Copyright Notice

Abstract

RFC7575 defines Intent as an abstract high-level policy used to operate the network. Intent 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. Up to now, there is no commonly agreed definition, interface or model of intent.

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 facilitate solution development.

Table of Contents

1. Introduction

The vision of intent-driven 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 telecom companies to begin adopting this new vision, and many Standards Development Organization (SDOs) to propose different intent framework.

Several SDOs and open source projects, such as Internet Engineering Task Force (IETF) (by the Autonomic Networking Integrated Model and Approach Working Group [ANIMA]), Open Networking Foundation (ONF) [ONF], Open Network Operating System (ONOS) [ONOS], have proposed intents for defining a set of network operations to execute in a declarative manner.

IETF [ANIMA] defines intent as a declarative policy, but still lacks a more complete definition, a tentative format, and a life-cycle. Within ONOS [ONOS], intent is represented as a list of Command-Line Interface (CLI) commands that allows users to bypass low-level details on the network, such as flows or host addresses. ONF through its Boulder and Aspen projects focuses on Northbound Interface (NBI) semantics and intent models.

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 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 a 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 user, and it can be ported across different platforms.

- It must provide an easy-to-use interface, which simplifies the 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.

Currently, work is underway on unifying a common understanding of intent concepts and terminology. Concerning NMRG, [CLEMM] is a document to present a definition for intent as higher-level declarative policy that operates at the level of network and services it provides. In addition, this document captures the differences between intent, policy and service.

The present document, together with [CLEMM], aims to become the foundation for future intent-related topic discussions regarding the NMRG.

1.1. Scope

This document mostly addresses intents in the context of network intents, however other types of intents are not excluded, as presented in Section 5.4. and Section 7.2. .

It is impossible to fully differentiate intents only by the common characteristics followed by concepts, terms and intentions. This document clarify what an intent represents for different stakeholders through a classification on various dimensions, such as solutions, 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, 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, users, or intent types to cater for future scenarios, applications or domains.

2. Key Words

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

3. Acronyms

AI: Artificial Intelligence

API: Application Programming Interface

CE: Customer Equipment

CFS: Customer Facing Service

CLI: Command Line Interface

DB: Data Base

DC: Data Center

ECA: Event-Condition-Action

GBP: Group-Based Policy

GPU: Graphics Processing Unit

IETF: Internet Engineering Task Force

IP: Internet Protocol

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

QoS: Quality of Service

RFS: Resource Facing Service

SDO: Standards Development Organization

SD-WAN: Software-Defined Wide-Area Network

SLA: Service-Level Agreement

SUPA: Simplified Use of Policy Abstractions

VLAN: Virtual Local Area Network

VM: Virtual Machine

VPN: Virtual Private Network

VNF: Virtual Network Function

4. 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.

Other definitions relevant to this draft, such as intent user, intent scope, intent network scope, intent abstraction, intent abstraction, and intent lifecycle are available in Section 6.

5. 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.

5.1. 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.

Different stakeholders consider an intent to be an ECA policy, a GBP policy, a business policy, a network service, a customer service, a network configuration, application/application group policy, any operator/administrator task, network troubleshooting/diagnostics/ test, a new app, a marketing term for existing management/orchestration capabilities, etc. Their intent is sometimes technical, non-technical, abstract or technology specific. For some stakeholders, intent is a subset of these and for other stakeholders intent is all of these. It has in some cases become a term to replace a very generic 'service' or 'policy' terminology.

Concerning this, [CLEMM] draft brings clarification with relation to what an intent is and how it differentiates from policies and services. Future versions of this draft will be kept aligned with [CLEMM].

While it is easier for those familiar with different standards to understand what service, CFS, RFS, resource, policy continuum, ECA policy, declarative policy, abstract policy or intent policy is, it may be more difficult for the wider audience.

An intent is mistaken by many to be just a synonym for policy. While it is easier for those familiar with different standards to understand what service, CFS, RFS, resource, policy continuum, ECA policy, declarative policy, abstract policy or intent policy is, it may be more difficult for the wider audience. Furthermore, those familiar with policies understand the difference between a business, intent, declarative, imperative, and ECA policy.

Therefore, it is important to start a discussion in the industry 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, DSLs and models.

5.2. 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 Actor. They can be considered as the main keys to classify intents, as we can easily group requirements by Solution and Actor. On the one hand, different Solutions and Actors have different requirements, expectations and priorities for intent-driven networking. Therefore, Actors require different intent types, depending on their context, the so they participate in different use cases. For instance, some users are more technical and require intents that expose more technical information. Other users do not understand networks and require intents that shield them from different networking concepts and technologies. The following are the solutions and intent users that intent-driven networking needs to support:

Solutions	Intent Users
Carrier Networks	Network Operator Service Designers Service Operators Customers/Subscribers
DC Networks	Cloud Administrator Underlay Network Administrator App Developers End-Users
Enterprise Networks	Enterprise Administrator App Developers End-Users

These intent solutions and intent users represent a starting point for the classification and are expendable through the methodology presented in Section 7.1. .

o For carrier networks scenario, for example, if the end-users wants to watch high-definition video, then the intent is to convert the video image to 1080p rate for the users.

o 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%.

o For Enterprise Networks scenario, enterprise administrators express their intent from an external client (application service provider). For example, when hosting a video conference, multiple remote accesses are required. An example of the intent expressed to the network operator is: For any 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.

5.3. Benefits of Intents to Respond to Network Requirements

Current network APIs and CLIs are too complex because they are highly integrated with the low level concepts exposed by networks. More specifically, network solutions must determine which low level communication technologies (e.g. protocol) they will use and, even more specifically, they must deal with the network topology that supports such communication (e.g. structure of networks and sub-networks). Application developers and end-users must not be required to set IP Addresses, VLANs, subnets, ports, etc. Therefore, all network stakeholders, including developers, end-users, operators, and administrators would benefit from the simpler interfaces, like:

o Allow Customer Site A to be connected to Internet via Network B

o Allow User A to access all internal resources, except the Server B

o Allow User B to access Internet via Corporate Network A

o Move all Users from Corporate Network A to the Corporate Network B

o Request Gold VPN service between my sites A, B and C

o Provide CE Redundancy for all Customer Sites

o Add Access Rules to my Service

Networks are complex, with many different protocols and encapsulations. Some basic questions are not easy to answer:

o Can User A talk to User B?

o Can Host A talk to Host B?

o Are there any loops in my network?

- o Are Network A and Network B connected?
- o Can User A listen to communications between Users B and C?

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

- o a reliable network that is self-configured and self-assured based on the intent
- o to be notified about the problem before the user is aware
- o automation of network/service recovery based on intent (self-healing, self-optimization)
- o to get suggestions about correction/optimization steps based on experience (historical data and behaviour)

Therefore, Operators and Administrators want to:

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

End-users cannot build their own services and policies without becoming technical experts and they must perform manual maintenance actions. Application developers and end-users/subscribers want to be able to:

- o build their own network services with their own policies via simple interfaces, without becoming networking experts
- o have their network services up and running based on intent and automation only, without any manual actions or maintenance

5.4. Intent Types that need to be supported

The following intent types 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 or add a service
  - 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.
  - For configuring and monitoring policies, alarms generation for non-compliance, auto-recovery
  - For design models and policies for network and network service design

- For design workflows, models and policies for operational task intents

- Operational task intents

    - For network migration
    - For server replacements
    - For device replacements
    - For network software upgrades
    - To automate any tasks that operators/administrator often perform

- Intents that affect other intents

    - It may be task-based intent that modifies 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.

6. 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.

6.1. 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.

- 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.

- 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.

6.2. Intent User Types

Intent user types, or intent actors as they are known in the area of declarative policy, represent the users that define and issue the intent request. Depending on the Intent Solutions, there are specific intent actors. Examples of intent actors are customers, network operators, service operators, enterprise administrators, cloud administrators, and underlay network administrators, or application developers.

- 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 users. This class of actor 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 actors.

- o Management personnel, such as network operators, may have the knowledge of the underlying network. However, they may not understand the details of the applications and services of Customers and End-Users.

6.3. Intent Scope

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

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

These intent scopes are expendable through the methodology presented in Section 7.1. .

6.4. 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.

6.5. Intent Abstraction

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

- For ordinary 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.

6.6. Intent Life-cycle

Intents can be classified into transient and persistent intents:

- If 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.

6.7. Hierarchy

In different phases of the autonomous driving network [TMF-auto], the intents are different. 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 o 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

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

o Level 3 - Self-optimization network: Deep awareness of network status and automatic network control, meeting users' network intentions. - Intent based on network status cognition

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

o 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

7. Intent Classification

This chapter proposes an intent classification approach that may help to classify mainstream intent related demos/tools.

The three classifications in this draft have been proposed from scratch, following the methodology presented, through three iterations: one for carrier 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 the 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 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.

7.1. 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, users or intent types in order to cater for future scenarios, applications or domains.

The intent classification workflow starts from the Solution Knowledge, which can provide information on requirements, use cases, technologies used, network properties, actors 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 provided as input to identifying the intent solution (e.g. Carrier, Enterprise, and Data Center). This intent solution is reviewed against the existing classification and it can either be used or add/remove the intent solution identified from the solution knowledge (R1-U1).

2. The next step is identifying the intent user types (e.g. customer, network operators, service operators, etc.) and then review existing classification and use it or add/remove the intent user type identified from the solution knowledge (R2-U2).

3. The next phase is to identify the type of intent (e.g. Network Intent, Customer Service Intent) and then review existing classification and use/add/remove the intent type (R3-U3).

4. The forth step is identifying the intent scope (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. The next step is to identify the network scope (e.g. Campus, Radio Access) and then then review existing classification and either use it or add/remove the identified network scope (R5-U5).

6. The next phase is to identify the abstractions (e.g. technical, non-technical) and then review existing classification and use/add/remove the abstractions (R6-U6).

7. The seventh step is to 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. The last step is to 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.

```
             +------------------------------------------+
             |Solution Knowledge (requirements,         |
             |use cases, technologies, network, actors, |
             |intent requirements)                      |
             +----------------+-------------------------+
                              |
                              | Input
                              v
                     +--------+--------+
                     |1.Identify Intent|
                     |  Solution       +------------+
                     |                 |            |
                     +---------^-+-----+            |
                            R1 | | U1               |
                               | |                  |
   +---------------+ U8        | |    R2         +--v----------------+
   |8.Identify New +---------+ | |    +----------> 2.Identify        |
   |  Categories   | R8      | | |    | U2       |   Intent          |
   |               <-------- | | |    +---------+   User Types       |
   +--------^------+       | | | |    | |         +-------|-----------+
            |              | | | |    | |                 |
            |             ++-+-v-v---+-v-+                |
   +--------+------+ U7   |              | R3     +------v------------+
   |7.Identify     +------>   Intent     +--------> 3.Identify        |
   |  Life-cycle   | R7   |Classification| U3     |    Type            |
   |  Requirements <------+              <--------+    of Intent       |
   +--------^------+      +^--^-+--^-+---+        +------|------------+
            |              || | |  | |                   |
            |              || | |  | |                   |
   +--------+-----+        || | |  | | R4         +-------v-----------+
   |6.Identify    | U6     || | |  | +----------> 4.Identify        |
   |  Abstractions+--------| | |  |   U4        |   Intent          |
   |              <---------+ | |  +-------------+   Scope           |
   +-------^------+ R6        | |                +-------+-----------+
           |                  | |                        |
           |               U5 | |R5                      |
           |          +-------+-v--------+               |
           |          |5.Identify Network|               |
           +----------+  Scope           <---------------+
                      +------------------+
```

7.2. 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 Solution, Data Center, and Enterprise).

```
                                   +--------------------------------+
                                   |Carrier           Enterprise    |
                               +-->|Data Center                     |
                               |   +--------------------------------+
                               |   +--------------------------------+
                               |   |Customer                        |
                 +----------+  |   |Network or Service Operator     |
               +>+Solutions +--+   |Application Developer           |
               | +----------+   +->|Enterprise Administrator        |
               |                |  |Cloud Administrator             |
               | +----------+   |  |Underlay Network Administrator  |
               +>+Intent    +---+  +--------------------------------+
               | |User      |      +--------------------------------+
               | |Types     |      |Customer Service Intent         |
               | +----------+      |Strategy Intent                 |
               | +----------+      |Network Service Intent          |
               +>+Intent    +----->|Underlay Network Service Intent |
      +------+ | |Type      |      |Network Intent                  |
      |Intent+-+ +----------+      |Underlay Network Intent         |
      +------+ |                   |Operational Task Intent         |
               | +----------+      |Cloud Management Intent         |
               +>+Intent    +---+  |Cloud Resource Management Intent|
               | |Scope     |   |  +--------------------------------+
               | +----------+   |  +--------------------------------+
               |                +->|Connectivity      Application   |
               | +----------+      |Security          QoS           |
               +>+Network   +---+  +--------------------------------+
               | |Scope     |   |  +--------------------------------+
               | +----------+   |  |Radio Access      Branch        |
               |                +->|Transport Access  SD-WAN        |
               | +----------+      |Transport Aggr.   VNF      PNF  |
               +>+Abstrac   +----+ |Transport Core    Physical      |
               | |tion      |    | |Cloud Edge        Logical       |
               | +----------+    | |Cloud Core        Campus        |
               | +----------+    | +--------------------------------+
               +>+Life      |    | +--------------------------------+
                 |cycle     +--+ +>|Technical         Non-Technical |
                 +----------+  |   +--------------------------------+
                               |   +--------------------------------+
                               +-->|Persistent        Transient     |
                                   +--------------------------------+
```

7.3. Intent Classification for Carrier Solution

7.3.1. Intent Users and Intent Types

The following table describes the Intent Users in Carrier Solutions and Intent Types with their descriptions for different intent users.

Intent User	Intent Type	Intent Type Description
Customer/ Subscriber	Customer Service Intent	Customer Self-Service with SLA and Value Added Service Example: Always maintain high quality of service and high bandwidth for gold level users. Operational statement: Measure the network congestion status, give different adaptive parameters to stations of different priority, thus in heavy load situation, makes the bandwidth of the high-priority users guaranteed. At the same time ensure the overall utilization of system, improve the overall throughput of the system.
	Strategy Intent	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.
Network Operator	Network Service Intent	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.
	Network Intent	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%.

Service Operator	Customer Service Intent	Service Operator’s Customer Orders, Customer Service / SLA Example: Provide service S with guaranteed bandwidth for customer A.
	Network Service Intent	Service Operator’s Network Orders / Network SLA Example: Provide network guarantees in terms of security, low latency and high bandwidth
	Operational Task Intent	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.
	Strategy Intent	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.
Application Developer	Customer Service Intent	Customer Service Intent API provided to the Application Developers Example: API to request network to watch HD video 4K/8K.

	Network Service Intent	Network Service Intent API provided to the Application Developers Example: API to request network and monitoring and traffic grooming.
	Network Intent	Network Intent API provided to the Application Developers Example: API to request network resources configuration.
	Operational Task Intent	Operational Task Intent API provided to the Application Developers. This is for the trusted internal Operator / Service Providers / Customer DevOps Example: API to request server migrations.
	Strategy Intent	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

7.3.2. Intent Categories

The following are the proposed categories:

Intent Scope: C1=Connectivity, C2=Security, C3=Application, C4=QoS

Network Scope:

oNetwork Domain: C1=Radio Access, C2=Transport Access, C3=Transport Aggregation, C4=Transport Core, C5=Cloud Edge, C6=Cloud Core)

oNetwork Function (NF) Scope: C1=VNFs, C2=PNFs

Abstraction(ABS): C1=Technical(with technical feedback), C2=Non-technical (without technical feedback) see Section 6.2. .

Life-cycle (L-C): C1=Persistent (Full life-cycle), C2=Transient (Short Lived)

7.3.3. Intent Classification Example

This section depicts an example on how the methodology described in Section 7.1. can be used in order to classify intents introduced in the 'A Multi-Level Approach to IBN' PoC demonstration [POC-IBN]. The PoC considered two 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 7.1. , we identify the following:

1.The Intent Solution is for the Carrier

2.The Intent User Type is the Network Operator for the slice intent, and the 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.

Intent User	Intent Type	Intent Scope				NF Scope		Network Scope						ABS		L-C	
		C1	C2	C3	C4	C1	C2	C1	C2	C3	C4	C5	C6	C1	C2	C1	C2
Customer / Sub-scriber	Customer Service Intent																
	Strategy Intent																
Network Operator	Network Service Intent	X		X		X						X	X	X		X	
	Network Intent																
	Operatio-nal Task Intent																
	Strategy Intent																
Service Operator	Customer Service Intent	Y		Y		Y							Y	Y	Y	Y	
	Network Service Intent																
	Op Task Intent																
	Strategy Intent																

```
|App      |Customer |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|Developer|Intent   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|         +---------+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|         |Network  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|         |Service  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|         |Intent   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|         +---------+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|         |Network  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|         |Intent   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|         +---------+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|         |Op Task  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|         |Intent   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|         +---------+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|         |Strategy |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|         |Intent   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
+---------+---------+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

7.4. Intent Classification for Data Center Solutions

7.4.1. Intent Users and Intent Types

The following table describes the Intent Users in DCN Solutions and Intent Types with their descriptions for different intent users.

Intent User	Intent Type	Intent Type Description
Customer / Tenants	Customer Service Intent	Customer Self-Service via Tenant Portal, Customers may have multiple type of end-users. Example: Request GPU computing and storage resources to meet 10k video surveillance services.
	Strategy Intent	This includes models and policy intents designed by Customers/ Tenants to be used by Customer and End-User Intents. Example: Request dynamic computing and storage resources of the service in special and daily times.
Cloud Administrator	Cloud Management Intent	Configuration of VMs, DB Servers, App Servers, Connectivity, Communication between VMs. Example: Request connectivity between VMs A,B,and C in Network N1.
	Cloud Resource Management Intent	Policy-driven self-configuration and and recovery / optimization Example: Request automatic life -cycle management of VM cloud resources.
	Operational Task Intent	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.

		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	Underlay Network Service Intent	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

		Intent. Example: Request automatic rapid detection of device failures and pre-alarm correlation.
	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%.
Application Developer	Cloud Management Intent	Cloud Management Intent API provided to the Application Developers. Example: API to request configuration of VMs,or DB Servers
	Cloud Resource Management Intent	Cloud Resource Management Intent API provided to the Application Developers. Example: API to request automatic life-cycle management of cloud resources.
	Underlay Network Service Intent	Underlay Network Service API provided to the Application Developers. Example: API to request real-time monitoring of device condition.
	Underlay Network Intent	Underlay Network Resource API provided to the Application Developers. Example: API to request dynamic management of IPv4 address pool resources.

	Operational Task Intent	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
	Strategy Intent	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.

7.4.2. Intent Categories

The following are the proposed categories:

Intent Scope: C1=Connectivity, C2=Security, C3=Application, C4=QoS C5=Storage C6=Compute

Network Scope

- oNetwork Domain: DC Network
- oDCN Network (DCN Net) Scope: C1=Logical, C2=Physical
- oDCN Resource (DCN Res) Scope: C1=Virtual, C2=Physical

Abstraction(ABS): C1=Technical(with technical feedback), C2=Non-technical (without technical feedback), see Section 6.2.

Life-cycle (L-C): C1=Persistent (Full life-cycle), C2=Transient (Short Lived)

7.4.3. Intent Classification Example

This section depicts an example on how the methodology described in Section 7.1. can be used in order to classify intents introduced in the 'A Multi-Level Approach to IBN' PoC demonstration [POC-IBN]. The PoC considered two 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 7.1. , 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 and service chain intent.

4.The Intent Scopes are connectivity and application.

5.The Network Scope is a logical, and the resource scope is virtual.

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, where the 'X' in the table refers to the slice and service chain intent.

Intent User	Intent Type	Intent Scope						DCN Res		DCN Net		ABS		L-C	
		C1	C2	C3	C4	C5	C6	C1	C2	C1	C2	C1	C2	C1	C2
Customer /Tenants	Customer Intent														
	Strategy Intent														
Cloud Admin	Cloud Management Intent	X		X				X		X		X	X	X	
	Cloud Resource Management Intent														
	Operational Task Intent														
	Strategy Intent														
Underlay Network Admin	Underlay Network Service Intent														
	Underlay Network Resource Intent														
	Operational Task Intent														
	Strategy														

	Intent														
App Developer	Cloud Management Intent														
	Cloud Resource Management Intent														
	Underlay Network Service Intent														
	Underlay Network Resource Intent														
	Operational Task Intent														
	Strategy Intent														

7.5. Intent Classification for Enterprise Solution

7.5.1. Intent Users and Intent Types

The following table describes the Intent Users in Enterprise Solutions and their Intent Types.

Intent User	Intent Type	Intent Type Description
End-User	Customer Service Intent	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 user A and B.
	Strategy Intent	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.
Administrator (internal or MSP)	Network Service Intent	Service provided by the Administrator to the End-Users and their Applications. Example: For any user of application X, 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 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 the routing algorithm. At the same time,the node resources need to meet the bandwidth requirements of 4K

		video conferences.
	Network Intent	Administrator requires network wide configuration (e.g. underlay, campus) or resource configuration (switches, routers, policies). Example: Configure switches in campus network 1 to prioritise traffic of type A. Configure Youtube as business non-relevant.
	Operational Task Intent	Administrator requests execution of any automated task other than Network Service Intents and Network Intents. Example: Request network security automated tasks such as Web filtering and DDOS cloud protection.
	Strategy Intent	Administrator designs models, policy intents and workflows to be used by other intents. Automate any tasks that Administrator often performs. Example: In case of emergency, automatically shift all traffic of type A through network N.
Application Developer	End-User Intent	End-User Service / Application Intent API provided to the Application Developers. Example: API for request to open a VPN service.
	Network Service Intent	Network Service API Provided to Application Developers. Example: API for request network 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.

7.5.2. Intent Categories

The following are the proposed categories:

- Intent Scope: C1=Connectivity, C2=Security, 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 6.2.
- Life-cycle (L-C): C1=Persistent (Full life-cycle), C2=Transient (Short Lived)

The following is the Intent Classification Table Example for Enterprise Solutions.

Intent User	Intent Type	Intent Scope				Net			ABS		L-C	
		C1	C2	C3	C4	C1	C2	C3	C1	C2	C1	C2
End-User	End-User Intent											
	Strategy Intent											
Enterprise Administrator	Network Intent											
	Strategy Intent											
Application Developer	End-User Intent											
	Network Service Intent											
	Network Intent											
	Operational Task Intent											
	Strategy Intent											

8. Security Considerations

This document does not have any Security Considerations.

9. IANA Considerations

This document has no actions for IANA.

10. Contributors

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

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

11. Acknowledgments

This document has benefited from reviews, suggestions, comments and proposed text provided by the following members, listed in alphabetical order: Brian E Carpenter, Juergen Schoenwaelder, Laurent Ciavaglia, Xiaolin Song, Alexander Clemm, Daniel King, Mehdi Bezahaf, Yehia Elkhatib, Pedro Andres Aranda Gutierrez.

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. References

12.1. Normative References

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

[RFC7575] Behringer, M., Pritikin, M., Bjarnason, S., Clemm, A., Carpenter, B., Jiang, S., and L. Ciavaglia, "Autonomic Networking: Definitions and Design Goals", RFC 7575, June 2015.

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

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

12.2. Informative References

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

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

[ANIMA] Du, Z., "ANIMA Intent Policy and Format", 2017, <https://datatracker.ietf.org/doc/draft-du-anima-an-intent/>.

[ONF] ONF, "Intent Definition Principles", 2017, <https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR-523_Intent_Definition_Principles.pdf>.

[ONOS] ONOS, "ONOS Intent Framework", 2017, <https://wiki.onosproject.org/display/ONOS/Intent+Framework/>.

[SUPA] Strassner, J., "Simplified Use of Policy Abstractions", 2017, <https://datatracker.ietf.org/doc/draft-ietf-supa-generic-policy-info-model/?include_text=1>.

[ANIMA-Prefix] Jiang, S., Du, Z., Carpenter, B., and Q. Sun, "Autonomic IPv6 Edge Prefix Management in Large-scale Networks", draft-ietf-anima-prefix-management-07 (work in progress), December 2017.

[TMF-auto] Aaron Richard Earl Boasman-Patel,et, A whitepaper of Autonomous Networks: Empowering Digital Transformation For the Telecoms Industry, inform.tmforum.org, 15 May, 2019.

[CLEMM] A. Clemm, L. Ciavaglia, L. Granville, J. Tantsura, "Intent-Based Networking - Concepts and Overview", Work in Progress, draft-irtf-nmrg-ibn-concepts-definitions-02, September 2020, https://tools.ietf.org/html/draft-irtf-nmrg-ibn-concepts-definitions-02

[POC-IBN] Barbara Martini, Walter Cerroni, Molka Gharbaoui, Davide Borsatti, "A multi-level approach to IBN", July 2020, https://www.ietf.org/proceedings/108/slides/slides-108-nmrg-ietf-108-hackathon-report-a-multi-level-approach-to-ibn-02

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

Will(Shucheng) Liu
Huawei Technologies
P.R. China
Email: liushucheng@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

Internet Research Task Force
Internet-Draft
Intended status: Informational
Expires: May 20, 2021

C. Zhou
H. Yang
X. Duan
China Mobile
D. Lopez
A. Pastor
Telefonica I+D

November 16, 2020

Concepts of Digital Twin Network
draft-zhou-nmrg-digitaltwin-network-concepts-02

Abstract

Digital twin technology is becoming a hot technology in industry 4.0. The application of digital twin technology in network field helps to realize efficient and intelligent management and network innovation. This document presents an overview of the concepts of Digital Twin Network (DTN), provides the definition and DTN, and then describes the benefits and key challenges of DTN.

Requirements Language

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

Status of This Memo

Table of Contents

1. Introduction

With the advent of 5G, Internet of Things and Cloud Computing, the scale of network is expanding constantly. Accordingly, the network operation and maintenance are becoming more complex due to higher complexity of network; and innovations on network will be more and more difficult due to the higher risk of network failure and higher trial cost.

Digital twin is the real-time representation of physical entities in the digital world. It has the characteristics of virtual-reality integration and real-time interaction, iterative operation and optimization, as well as full life-cycle, and full business data-driven. At present, it has been successfully applied in the fields of intelligent manufacturing, smart city, complex system operation and maintenance [Tao2019].

A digital twin network platform can be built by applying digital twin technology to network and creating virtual image of physical network facilities. Through the real-time data interaction between physical network and twin network, the digital twin network platform can help the network to achieve more intelligent, efficient, safe and full life-cycle operation and maintenance.

2. Definition of Digital Twin Network

So far, there is no standard definition of digital twin network in networking industry or SDOs. This document attempts to define Digital Twin Network (DTN) as a virtual representation of the physical network, analyzing, diagnosing, simulating and controlling the physical network based on data, model and interface, so as to achieve the real-time interactive mapping between physical network and virtual twin network. According to the definition, DTN contains five key elements: data, mapping, model, interface and orchestration stack, as shown in Figure 1.

```
                                 +------------+
                                 |            |
                                 | Interface  |
             +------------+      |            |      +------------+
             |            |------+------------+------|            |
             |   Models   |                          |    Data    |
             |            |    Analyze, Diagnose     |            |
             +------------+                          +------------+
                          | +----------------------+ |
                          | | NETWORK DIGITAL TWIN | |
                          | +----------------------+ |
                          |                          |
                          |    Simulate, Control     |
                          |                          |
                  +-------------+--------------+--------------+
                  |             |              |              |
                  |  Mapping    |              |Orchestration |
                  |             |              |              |
                  +-------------+              +--------------+
```

Figure 1: Key Elements of Digital Twin Network

- o Data is cornerstone for constructing a DTN system, in which unified data repository can be the single source of the truth and provide timely and accurate data support.

- o Real-time interactive mapping between physical network and virtual twin network is the most typical feature that DTN is different from network simulation system.

- o Data model is the ability source of DTN. Various data models can be designed and flexibly combined to serve various network applications.

- o Standardized interface is the key technique enabler, which can effectively ensure the compatibility and scalability of DTN system.

- o The orchestration stack controls the flows of data and control actions. It relies on the dynamic lifecycle management of network models and elements to provide repeatablity (the capacity to replicate network conditions on demand) and reproducibility (the ability to replay successions of events, possibly under controlled variations).

3. Benefits of Digital Twin Network

DTN can help enable closed-loop network management across the entire lifecycle, from digital deployment and simulation, to visualized assessment, physical deployment, and continuous verification. In doing so, customers are able to achieve network-wide insights, precise planning, and rapid deployment in multiple areas, including networks, services, users, and applications. All the benefits of DTN can be categorized into three major types: low cost of network optimization, intelligent network decision making, and high efficient network innovation. The following sections describe the three types of benefits respectively.

3.1. Lower the cost of network optimization

With extremely large scale, network is becoming more and more complex and difficult to operate. Since there is no effective platform for simulation, traditional network optimization has to be tried on real network directly with long time cost and high service impact running on real network. This also greatly increases network operator’s OpEX.

With DTN platform, network operators can well simulate the candidate optimization solutions before finally deploy them to real network. Compared with traditional methods, this is of quite low risk and will bring much less impact on real network. In addition, the operator’s OpEX will be greatly decreased accordingly.

3.2. More intelligent for network decision making

Traditional network operation and management mainly focus on deploying and managing current services, while lacking of handling past data and predicting future status. This kind of passive and protective maintenance is difficult to adapt to large-scale network scenarios.

DTN can combine data acquisition, big data processing and AI modeling to achieve the assessment of current status, diagnosis of past problems, as well as prediction of future trends, then give the results of analysis, simulate various possibilities, and provide more comprehensive decision support. This will help network achieve predictive maintenance from current protective maintenance. The network behavioral repeatability and reproducibility properties in the DTN allow to evaluate different conditions and controlled variations of them, exploring choice as many times as needed to apply the better emulation and decision procedures.

3.3. High efficient for network innovation

Due to higher trial risk, real network environment is normally unavailable to network researcher when they explore innovation techniques. Instead, researchers have to use some offline simulation platforms. This greatly impacts the real effectiveness of the innovation, and greatly slow down the speed of network innovation. Moreover, risk-averse network operators naturally reluctant to try new technologies due to higher failure risk as well as the higher failure cost.

DTN can generate virtual twin entity of the real network. This helps researches explore network innovation (e.g. new network protocols, network AI/ML applications, etc.) efficiently, and helps network operators 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%) of network and the slow learning speed or phase-in learning steps of AI/ML algorithms. With DTN platform, AI/ML can fully complete the leaning and training with the sufficient data before deploy the model to the real network. This will greatly encourage more network AI innovations in future network.

Implementing Intent-Based Networking (IBN) via DTN can be another example to show how DTN improves the efficiency of deploying network innovation. IBN is an innovative technology for life-cycle network management. Future network 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 character of an IBN system is that user's intent can be assured automatically via continuously adjusting the policies and validating the real-time situation. To lower the impact on real network, several rounds of adjustment and validation can be simulated on the DTN platform instead of directly on physical netowrk. Therefore, DTN can be an important enabler platform to implement IBN system and speed up the deployment of IBN in customer's network.

3.4. Privacy and Regulatory Compliance

The requirements on data confidentiality and privacy on network service providers increase the complexity of network management, as intelligent decision engines depend on data flows. As a result, the improvement of data-enabled management requires complementary techniques providing strict control and security mechanisms to guarantee data privacy protection and regulatory compliance in these aspects. Some examples of these techniques can include payload inspection, including de-encryption user explicit consents, or data anonymization mechanisms.

Given DTN works with mapped traffic or services from real networks, but using traffic simulations, including automated tools for synthetic user activity. The lack of personal data permits to lower the privacy requirements and simplify privacy-preserving techniques, as the data is not coming from real users. As a result, DTN allows to focus on management improvements, without other concerns. Additionally, logging and auditing the DTN experiments and synthetic user activities provide additional information for further design and planning, without the need of traffic inspection.

3.5. Customize Network Operation Training

Networks architectures can be complex, and their operation and management require expert personnel and the learning curve can be steep in most cases. DTN offers an opportunity to train staff for customized networks and specific user needs. Several areas can benefit with the use of it. 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.

4. Reference Architecture of Digital Twin Network

So far, there is no reference or standard architecture for Digital Twin Network in network domain. Based on the definition of key elements of DTN described in section 2, reference architecture with three layers of Digital Twin Network can be designed as below, shown in Figure 2.

```
     +---------------------------------------------------------+
     |   +-------+   +-------+          +-------+       Network|
     |   | App 1 |   | App 2 |   ...    | App n |   Application|
     |   +-------+   +-------+          +-------+              |
     +-------------^-------------------+-----------------------+
                   | ability supply    |intent input
                   |                   |
     +---------------------------------v-----------------------+
     |                                     Network Digital Twin|
     |  +--------+   +------------------------+   +--------+   |
     |  |        |   | Service Mapping Models |   |        |   |
     |  |        |   |  +------------------+  |   |        |   |
     |  | Data   +--->  |Functional Models |  +---> Digital|   |
     |  | Sharing|   |  +-----+-----^------+  |   | Twin   |   |
     |  | Repo-  |   |        |     |         |   | Entity |   |
     |  | sitory |   |  +-----v-----+------+  |   | Mngmt  |   |
     |  |        <---+  |  Basic Models    |  <---+        |   |
     |  |        |   |  +------------------+  |   |        |   |
     |  +--------+   +------------------------+   +--------+   |
     +--------^------------------------------------------------+
              |                            |
              | data collection            | control
     +-------------------------------------v-------------------+
     |                                         Physical Network|
     |              Network infrastructures                    |
     +---------------------------------------------------------+
```

Figure 2: Reference Architecutre of Digital Twin Network

1. Bottom layer is Physical Network. All network elements in physical network exchange massive network data and control with network digital twin entity, via southbound interfaces. Physical network can be either telecommunication operator network, or data center network, campus network, industrial Internet of things or other network types.

2. Middle layer is Network Digital Twin Entity, which is the core of DTN system. This layer includes three key subsystems: Data Sharing Repository, Service Mapping Models and Digital Twin Entity Management.

    * Data Sharing Repository provides accurate and complete information for building various service models by collecting and updating the real-time operational data of various network elements through the southbound interface. In addition to

data storage, Data Sharing Repository is also responsible to provide data services for the Service Mapping Models sub-system, including fast retrieval, concurrent conflict, batch service, unified interface, etc.

* Service Mapping Models completes data-based modelling, provides data model instances for various network applications, and maximizes the agility and programmability of network services. The data models include two major types: basic models and functional models.

  + Basic Model refers to the network element model and network topology model of the network digital twin entity based on the basic configuration, environment information, operational state, link topology and other information of the network element, to complete the real-time accurate description of the physical network.

  + Functional model refers to various data models such as network analysis, simulation, diagnosis, prediction, assurance, etc. The functional models can be constructed and expanded by multiple dimensions: by network type, there can be models serving for single network domain or multi network domain; by function type, it can be divided into state monitoring, traffic analysis, security drill, fault diagnosis, quality assurance and other models; by generality, it can be divided into general model and special-purpose model. Specifically, multiple dimensions can be combined to create a data model for more specific application scenario.

* Digital Twin Entity Management completes the management function of digital twin network, records the life-cycle of the entity, visualizes and controls various elements of network digital twin, including topology management, model management and security management.

3. Top layer is Network Application. Various applications (e.g. Network intelligent O&M, IBN, etc.) can effectively run against Digital Twin Network platform to implement either conventional or innovative network operations, with low cost and less service impact on real network. Network application provide requirements to network digital twin entity via northbound interface; then the service is simulated by various service model instances; after fully verified, the change control can be deployed safely to physical network.

5. Challenges to build Digital Twin Network

As mentioned in above section, DTN can bring many benefits to network management as well as network innovation. However, it is still challenging to build an effective and efficient DTN system. The following are the major challenges and problems.

- Large scale challenge: The digital twin entity of large-scale network will significantly increase the complexity of data acquisition and storage, the design and implementation of model. And the requirements of software and hardware of the system will be very high.

- Compatibility issue: It is difficult to establish a unified digital twin platform with unified data model in the whole network domain due to the inconsistency of technical implementation and supporting functionalities of different manufacturers' devices in the network.

- Data modeling difficulties: Based on large-scale network data, data modeling should not only focus on ensuring the richness of model functions, but also need to consider the flexibility and scalability of the model. These requirements further increase the difficulty of building efficient and hierarchical functional data models.

- Real-time requirement: For services with high real-time requirements, the processing of model simulation and verification through DTN system will increase the service delay, so the function and process of the data model need to increase the processing mechanism under various network application scenarios; at the same time, the real-time requirements will further increase the system software and hardware performance requirements.

- Security risks: Network digital twin entity synchronizes all the data of physical network in real time, which will increase the security risk of user data, such as information leakage or more vulnerable to attack.

To solve the above problems and challenges, Digital Twin Network needs continuous optimization and breakthrough on key enabling technologies including data acquisition, data storage, data modeling, network visualization, interface standardization, and security assurance, so as to meet the requirements of compatibility, reliability, real-time and security under large-scale network.

6. Summary

The research and application of Digital Twin Network is just beginning. This document presents an overview of the concepts and definition of DTN. Looking forward, further researches on DTN usage scenarios, requirements, architecture and key enabling technologies should be promoted by the industry, so as to accelerate the implementation and deployment of DTN in real network.

7. Security Considerations

TBD.

8. IANA Considerations

This document has no requests to IANA.

9. References

9.1. Normative References

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

9.2. Informative References

[I-D.irtf-nmrg-ibn-concepts-definitions]
Clemm, A., Ciavaglia, L., Granville, L., and J. Tantsura, "Intent-Based Networking - Concepts and Definitions", draft-irtf-nmrg-ibn-concepts-definitions-02 (work in progress), September 2020.

[Tao2019] Tao, F., Zhang, H., Liu, A., and A. Nee, "Digital Twin in Industry: State-of-the-Art. IEEE Transactions on Industrial Informatics, vol. 15, no. 4.", April 2019.

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