OPSAWG B. Claise

Internet-Draft Huawei

Intended status: Informational J. Quilbeuf

Expires: October 25, 2021 Independent

D. Lopez

Telefonica I+D

D. Voyer

Bell Canada

T. Arumugam

Cisco Systems, Inc.

April 23, 2021

Service Assurance for Intent-based Networking Architecture

draft-claise-opsawg-service-assurance-architecture-05

Abstract

This document describes an architecture for Service Assurance for

Intent-based Networking (SAIN). This architecture aims at assuring

that service instances are running as expected. As services rely upon

multiple sub-services that are provided by the underlying network devices, getting the

assurance of a healthy service is only possible with a holistic view

of network devices. This architecture not only helps to correlate

the service degradation with the network root cause but also the

impacted services when a network component fails or degrades.

Status of This Memo

This Internet-Draft is submitted in full conformance with the

provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering

Task Force (IETF). Note that other groups may also distribute

working documents as Internet-Drafts. The list of current Internet-

Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months

and may be updated, replaced, or obsoleted by other documents at any

time. It is inappropriate to use Internet-Drafts as reference

material or to cite them other than as "work in progress."

This Internet-Draft will expire on October 25, 2021.

Claise, et al. Expires October 25, 2021 [Page 1]

Internet-DraService Assurance for Intent-based Networking Ar April 2021

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the

document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal

Provisions Relating to IETF Documents

(https://trustee.ietf.org/license-info) in effect on the date of

publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must

include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as

described in the Simplified BSD License.

Table of Contents

1. Terminology . . . . . . . . . . . . . . . . . . . . . . . . . 2

2. Introduction . . . . . . . . . . . . . . . . . . . . . . . . 5

3. Architecture . . . . . . . . . . . . . . . . . . . . . . . . 6

3.1. Decomposing a Service Instance Configuration into an

Assurance Graph . . . . . . . . . . . . . . . . . . . . . 9

3.2. Intent and Assurance Graph . . . . . . . . . . . . . . . 10

3.3. Subservices . . . . . . . . . . . . . . . . . . . . . . . 11

3.4. Building the Expression Graph from the Assurance Graph . 11

3.5. Building the Expression from a Subservice . . . . . . . . 12

3.6. Open Interfaces with YANG Modules . . . . . . . . . . . . 12

3.7. Handling Maintenance Windows . . . . . . . . . . . . . . 13

3.8. Flexible Architecture . . . . . . . . . . . . . . . . . . 14

3.9. Timing . . . . . . . . . . . . . . . . . . . . . . . . . 15

3.10. New Assurance Graph Generation . . . . . . . . . . . . . 15

4. Security Considerations . . . . . . . . . . . . . . . . . . . 16

5. IANA Considerations . . . . . . . . . . . . . . . . . . . . . 16

6. Contributors . . . . . . . . . . . . . . . . . . . . . . . . 16

7. Open Issues . . . . . . . . . . . . . . . . . . . . . . . . . 16

8. References . . . . . . . . . . . . . . . . . . . . . . . . . 16

8.1. Normative References . . . . . . . . . . . . . . . . . . 16

8.2. Informative References . . . . . . . . . . . . . . . . . 17

Appendix A. Changes between revisions . . . . . . . . . . . . . 18

Acknowledgements . . . . . . . . . . . . . . . . . . . . . . . . 18

Authors' Addresses . . . . . . . . . . . . . . . . . . . . . . . 19

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

Claise, et al. Expires October 25, 2021 [Page 2]

Internet-DraService Assurance for Intent-based Networking Ar April 2021

14 [RFC2119] [RFC8174] when, and only when, they appear in all

capitals, as shown here.

SAIN agent: A functional 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: A Directed Acyclic 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: A functional component that fetches or receives the computer-

consumable output of the SAIN 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: A generic term for a DAG representing a computation

in SAIN. More specific terms are:

o Subservice expression: Is an expression graph representing all the

computations to execute for a subservice.

o Service expression: Is an expression graph representing all the

computations to execute for a service instance, i.e., including the

computations for all involved subservices.

o Global Computation Graph: Is an 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 service assurance graph.

Informational Dependency: Type of dependency whose health 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.

Claise, et al. Expires October 25, 2021 [Page 3]

Internet-DraService Assurance for Intent-based Networking Ar April 2021

Metric: An information retrieved from the network running the assured service.

Metric engine: An entity that 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

a 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: A functional component that is 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 measured subservice is operational as expected.

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.

Claise, et al. Expires October 25, 2021 [Page 4]

Internet-DraService Assurance for Intent-based Networking Ar April 2021

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

as expected (for example, the service might be degraded because of a

failure in the network), the network operator must monitor the

service operational data at the same time as the configuration. The

industry has been standardizing on telemetry to push network element

performance information.

A network administrator needs to monitor her network and services as

a whole, independently of the use cases or the management protocols.

With different protocols come different data models and different

ways to model the same type of information. When network

administrators deal with multiple protocols, the network management

must perform the difficult and time-consuming job of mapping data

models: the model used for configuration with the model used for

monitoring. This problem is compounded by a large, disparate set of

data sources (MIB modules, YANG models [RFC7950], IPFIX information

elements [RFC7011], syslog plain text [RFC3164], TACACS+ [RFC8907],

RADIUS [RFC2865], etc.). In order to avoid this data model mapping,

the industry converged on model-driven telemetry to stream the

service operational data, reusing the YANG models used for

configuration. Model-driven telemetry greatly facilitates the notion

of closed-loop automation whereby events/status from the network drive

remediation changes back into the network.

Claise, et al. Expires October 25, 2021 [Page 5]

Internet-DraService Assurance for Intent-based Networking Ar April 2021

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 services are impacted when this

specific optic dBM begins to degrade? Which applications are 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 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 document

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, SAIN 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

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.

Claise, et al. Expires October 25, 2021 [Page 6]

Internet-DraService Assurance for Intent-based Networking Ar April 2021

The SAIN architecture is a generic architecture, applicable to

multiple environments. Obviously wireline but also wireless,

including 5G, virtual infrastructure manager (VIM), and even virtual

functions. Thanks to the distributed graph design principle, graphs

from different environments/orchestrator can be combined together.

As an example of a service, let us consider a point-to-point L2VPN

connection (i.e., pseudowire). Such a service would take as

parameters the two ends of the connection (device, interface or

subinterface, and address of the other end) and configure both

devices (and maybe more) so that a L2VPN connection is established

between the two devices. Examples of symptoms might be "Interface

has high error rate" or "Interface flapping", or "Device almost out

of memory".

To compute the health status of such a service, the service is

decomposed into an assurance graph formed by subservices linked

through dependencies. Each subservice is then turned into an

expression graph that details how to fetch metrics from the devices

and compute the health status of the subservice. The subservice

expressions are combined according to the dependencies between the

subservices in order to obtain the expression graph which computes

the health status of the service.

The overall SAIN architecture 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/service orchestrator.

Claise, et al. Expires October 25, 2021 [Page 7]

Internet-DraService Assurance for Intent-based Networking Ar April 2021

+-----------------+

| 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.

Claise, et al. Expires October 25, 2021 [Page 8]

Internet-DraService Assurance for Intent-based Networking Ar April 2021

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. Inferring an Assurance Graph from a Service Instance Configuration

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 service.

The decomposition into subservices is an important function of this

architecture, for the following reasons.

o The result of this decomposition provides a relational picture of

a service instance, that can be represented as a graph (called

assurance graph) to the operator.

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.

Claise, et al. Expires October 25, 2021 [Page 9]

Internet-DraService Assurance for Intent-based Networking Ar April 2021

+------------------+

| 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.

Claise, et al. Expires October 25, 2021 [Page 10]

Internet-DraService Assurance for Intent-based Networking Ar April 2021

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 two devices and configure

on each device a virtual tunnel interface. Then:

o 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, the latency and

bandwidth requirements of this tunnel.

o 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.

Claise, et al. Expires October 25, 2021 [Page 11]

Internet-DraService Assurance for Intent-based Networking Ar April 2021

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.

Claise, et al. Expires October 25, 2021 [Page 12]

Internet-DraService Assurance for Intent-based Networking Ar April 2021

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 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

Claise, et al. Expires October 25, 2021 [Page 13]

Internet-DraService Assurance for Intent-based Networking Ar April 2021

symptoms from "Peer 1 Tunnel Interface" and "Tunnel Service

Instance". All other subservices are unaffected.

o 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 a 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.

Claise, et al. Expires October 25, 2021 [Page 14]

Internet-DraService Assurance for Intent-based Networking Ar April 2021

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 maintenance) 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

Claise, et al. Expires October 25, 2021 [Page 15]

Internet-DraService Assurance for Intent-based Networking Ar April 2021

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

o Youssef El Fathi

o 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>.

Claise, et al. Expires October 25, 2021 [Page 16]

Internet-DraService Assurance for Intent-based Networking Ar April 2021

[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.

[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>.

Claise, et al. Expires October 25, 2021 [Page 17]

Internet-DraService Assurance for Intent-based Networking Ar April 2021

[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>.

[RFC8907] Dahm, T., Ota, A., Medway Gash, D., Carrel, D., and L.

Grant, "The Terminal Access Controller Access-Control

System Plus (TACACS+) Protocol", RFC 8907,

DOI 10.17487/RFC8907, September 2020,

<https://www.rfc-editor.org/info/rfc8907>.

Appendix A. Changes between revisions

v02 - v03

o Timing Concepts

o New Assurance Graph Generation

v01 - v02

o Handling maintenance windows

o Flexible architecture better explained

o Improved the terminology

o Notion of mapping information model to data model, while waiting

for YANG to be everywhere

o Started a security considerations section

v00 - v01

o Terminology clarifications

o Figure 1 improved

Acknowledgements

Claise, et al. Expires October 25, 2021 [Page 18]

Internet-DraService Assurance for Intent-based Networking Ar April 2021

The authors would like to thank Stephane Litkowski, Charles Eckel,

Rob Wilton, Vladimir Vassiliev, Gustavo Alburquerque, Stefan Vallin,

and Eric Vyncke for their reviews and feedback.

Authors' Addresses

Benoit Claise

Huawei

Email: benoit.claise@huawei.com

Jean Quilbeuf

Independent

Email: jean@quilbeuf.net

Diego R. Lopez

Telefonica I+D

Don Ramon de la Cruz, 82

Madrid 28006

Spain

Email: diego.r.lopez@telefonica.com

Dan Voyer

Bell Canada

Canada

Email: daniel.voyer@bell.ca

Thangam Arumugam

Cisco Systems, Inc.

Milpitas (California)

United States of America

Email: tarumuga@cisco.com

Claise, et al. Expires October 25, 2021 [Page 19]