Development of Service Assurance Techniques for Intent Based Networking

Jovanovic, M.

https://doi.org/10.34737/w627v

University of Westminster
School of Computer Science and Engineering

Thesis

Doctoral Research Project Title:

Development of Service Assurance Techniques for Intent Based Networking

Submitted
by:

Mioljub Jovanovic

Director of studies: **Dr Djuradj Budimir**

September 2022
115 New Cavendish Street
London, W1W 6UW.

ABSTRACT

This project aims at developing a new methodology of Service Assurance Techniques for Intent Based Networking. The mismatch problem between a network service and monitoring of the network service is discussed and addressed. Objective is to reduce amount of telemetry data exported over Wide Area Network (WAN) by processing telemetry at the edge by designing and implementing novel assurance agents. Instead of telemetry compression or in-line telemetry techniques, in our research we then proposed the different approach how to increase relevancy of telemetry data by using Service Aware information/tagging which is exported by our monitoring and telemetry export agents. Current lab tests on NSO (Cisco Orchestrator) based setup prove reduction of data exported over WAN by using real-world service definitions created by Subject Matter Experts. Such service definitions are provisioned on Cisco devices (routers) automatically by service orchestrator (NSO). By applying the proposed solution, amount of telemetry data may be reduced by factor of 40, while retaining relevancy of the provided information. Such a large efficiency factor is achieved by avoiding sending general device telemetry data for processing at central location, but rather sending service specific status information from the edge. Instead of sending number raw data, we send only computed information relevant for the service status. Experimental lab setup used is leveraging real-time Model-Driven Telemetry, Physical Network Devices as well as Virtual Devices. Future work is to apply developed Service Assurance Techniques on Open-Source Platform, such as Open Network Automation Platform (ONAP) [1]

ACKNOWLEDGEMENT

First, I'd like to express my gratitude to my director of studies, Prof Dr D. Budimir, for all the support, encouragement, guidance, and supervision he has provided throughout the duration of the research work. In addition to all the above, I'd like to thank and acknowledge Professor Budimir for finding the time to help me, despite his busy schedule, by giving directions and advice which helped shape up many aspects of the research.

Also, I'd like to express my gratitude to my second supervisor, Dr. D. Bajramovic, for her help, assistance, and advice.

Next, I'd like to thank to Dr M. Cabarkapa, for his encouragement, support and help to keep the motivation at the high level and to persist towards pursuing the research activities.

Finally, I'd like to dedicate this thesis to my family and friends who believed in me and supported by on this endeavour.

LIST OF FIGURES

LIST OF TABLES

Table of contents

1 Introduction

1.1 Overview

Software Defined Networking (SDN) and Intent Based Networking (IBN) are concepts aimed at facilitating configuration management and ensuring agile capabilities for the network.

With rapid expansion of network services and data transported over the network, there is constant trend in increasing number of changes in the communication networks. Designing, deploying and verifying changes in legacy environment become even more challenging since it's becoming increasingly complex to obtain service-aware telemetry data from the network elements. Network operations are facing challenges extracting meaningful information from monitored data acquired from various data sources. Without complex analytics, telemetry data could almost be considered as simple heap of data. As a consequence, it becomes increasingly complex to monitor service assurance since telemetry data received is only with no particular correlation towards services which exist in the network. Such complexity should be addressed by network monitoring platform or utility which would offer holistic view of the services across the network with capability to correlate telemetry data with the service configuration usually deployed by the Orchestrator such as Open Network Automation Platform (ONAP) [1], Cisco Network Services Orchestrator (NSO) [2]

One of the recent innovative concepts to come to the networking industry is introduction of Intent Based Networking (IBN). The term Intent-Based Networking is not entirely new, since it has been around for more than a decade, yet only IBN concept has been leveraged by large cloud and network vendors such as Google and Cisco, along with new players in the area such as Apstra [2], Veriflow [3], Huawei [4] and other companies who entered IBN space. Objective of IBN based network is simple: Network that runs autonomously using given the initial intent – set of requirements/commands. Simple way to explain what IBN network would be to say it's a self-operating autonomous system, closed loop which doesn't require any human input. Analogy to some existing solutions could be drawn – such as self-driven cars for example and its automation features such as automatic parking. Generally, most recent cars would be able to park itself once driver decides it's safe to park and initiates automatic parking sequence – responsibility still lies with the driver. Such parallel approach can be drawn with the modern

automated network infrastructure, where the network administrator needs to decide when automation is required.

Comparatively, autonomous car a closed-loop system based on intent, where driver could issue high-level command such as “drive me to work” and such car would perform sequence of “intelligent” steps to conclude task by complying to all traffic and other set of rules which may have been imposed. Perhaps simpler, yet comparable example is parallel-parking of the car – where driver could issue command such as “parallel-park to the right” in order to activate autonomous parking sequence, but vehicle itself would perform all additional steps to interpret the command and start executing set of manoeuvres to park itself safely. It would be up to the vehicle to identify whether all the requirements for the intent have been met so that parking sequence could even commence sufficient space available to fit the vehicle, parking is allowed, Coming back to the actual topic at question – computer networking, in this context network operator could issue command such as: “establish connectivity between point A and B, with following criteria …” and IBN should be able to interpret this requirement by translating it to the appropriate network policy and configure networking devices to activate desired service. It’s also reasonable to expect some aspects of IBN may be leveraged separately, activating service, service assurance or similar.

We could consider Network Operations as a trade-off between three main Key Performance Metrics: Scale, Reliability and Efficiency. In order to focus on any of the mentioned metrics Network operators usually have to decide which metric to focus on at the expense of the remaining relevant metrics. For example: If Network Operator focuses on Scale, then Efficiency is usually the one that suffers. Similarly, should Efficiency be the main concern, then Reliability will probably be the impacted one. Of course, achieving all three metrics should be the final objective. Policy Based Network Management (PBNM) [5] is not a new concept, but its application has becoming really important in recent years because the number of network elements is rapidly increased. Therefore, automatic network management that considers network administrator's intent or policy is essential nowadays. Different policies are possible using Event-Condition-Action (ECA) rule ("if condition then action") [6], [7].

In the paper [8], administrator's intent reactive configuration with extended intent-based Network Modelling **(**NEMO) language has been described. The two use cases as well as two scenarios for these cases are shown in Figure and Figure respectively. The issues of implementation have been discussed in detail. Main problem is that the network traffic depends on software usage.

Therefore, the automatic network management with the extension of NEMO language has been introduced in order to solve this problem. However, this extension is not tested deeply and there is no guarantee this will work in more complex scenarios.

Cisco's approach to solve IBN is represented in the Figure 1

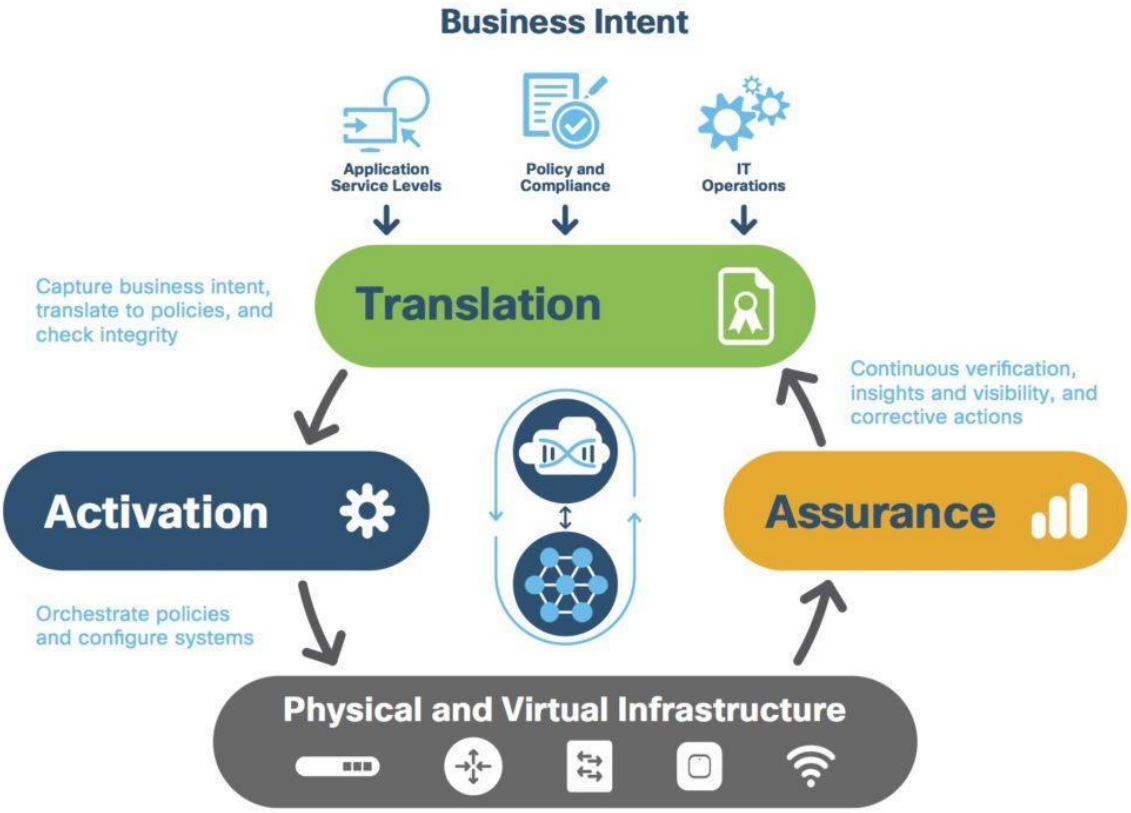

Figure 1 Cisco representation of Intent Based Networking (source: Cisco) [9]

From Figure 1 which represents Cisco's approach to IBN, it's visible that there are few transformations which need to happen in order to apply business intent to the actual network infrastructure. This essentially means that business logic needs to be translated to policies which will in turn be represented as series of configuration directives for both physical and virtual infrastructure network elements.

1.2 Research Aims and Objectives

The main challenges exhibited in the current network, telemetry and assurance designs are following:

- Too much data, but not enough information - Without complex analytics, telemetry is just a haystack of data. Operators are stuggling with large quantity of telemetry, while the most important information is missing – link between telemetry and the network service
- It's increasingly difficult to monitor service health and intent, as there is no service context or correlation associated to the data
- There is no simple solution for service assurance offering a holistic view of the state of the services across the network

The individual research aims, and objectives of this project can be summarised as follows:

- Defining mismatch between the network service sonfiguration and network service monitoring
- Clarify importance of the service assurance
- Investigation of current service assurance capabilities connection with Intent Based Networking
- Development of novel service assurance techniques for Intent based networking
- Verification of the proposed approaches and algorithms by simulations
- Verification of the proposed approaches and algorithms in experimental environment

In this research work, we are proposing solution based on Intent Based Networking. The proposed approach consists by:

- Extraction of configuration intent by analysing of the network service configuration.
- Discovery of the network elements along the network service data path.
- Leveraging existing network monitoring capabilities of network elements, along with probes and Model Driven Telemetry (MDT) to get more accurate information on the status of desired Network Service.

Research methodology used in understanding benefits of proposed monitoring approach involves qualitative approach, comparative analysis of existing - probe based monitoring and

proposed solution based on Intent Based Networking. By using the proposed approach, we have achieved higher than 40 for the telemetry efficiency ratio parameter.

1.3 Research questions

In this research, we are trying to answer the following fundamental questions, in order to achieve the main research objectives:

1. What is the network intent and how is it related to network service configuration?
2. How could we establish relation between network service intent, network device configuration and monitored data?
3. What methods should be used to compare techniques for monitoring and telemetry data?
4. What design methodologies can be offered to improve telemetry efficiency while retaining relevance of the information carried in telemetry data?
5. Which network assurance techniques exist?
6. How can we perform network service configuration decomposition in order to enable network assurance?
7. Could we develop improved network assurance technique?

While automation of network configuration and provisioning network services has advanced considerably, we're still observing lag in verification, monitoring and assurance of the mentioned configuration and automation tasks. To illustrate this observation, we can take examples of usual network service models, such as: Virtual Private Network (VPN), Routing Protocol infrastructure, Video Streaming service, Equal-path multipath etc.
In the Figure 12 we've outlined an example of the service configured using Yet Another Next Generation modelling language (YANG)
While the configuration part is well structured, monitoring and assurance of the provisioned service poses a significant challenge.

1.4 Original contribution to knowledge

Service Assurance, Model-Driven Telemetry, YANG are relatively new concepts, while service configuration and configuration monitoring are well known concepts present for decades. Despite the omni-present existence of mentioned concepts, there is very little that can use leveraged to tie telemetry data to the service intent.

In this research we're focusing on creating a bridge, providing the missing link between the intent, service definition, service configuration and service monitoring by means of advanced techniques aimed at:

- Reducing telemetry data, yet increasing amount of useful information encoded in the telemetry streams (Chapter 3)
- Service decomposition into atomic network service components (Chapter 4)
- Solving one service component at a time using service heuristic package definition (Chapter 4)
- Providing the health status per service component (Chapter 5)
- Closed loop automation architecture (Chapter 5)

By applying the above guiding principles, we're tying telemetry to the intent, based on the service KPI and therefore bridging the gap between Configuration and Telemetry and achieved following contributions:

1. We have introduced the notion of the "Metric Engine", which maps an abstracted metric such as "interface.mtu" to a concrete metric implementation. For instance with Model-Driven Telemetry (MDT), when using the exact sensor path such as Openconfig path to the mtu "openconfig-interfaces:interfaces/interface/state/mtu" or via the CLI command "show interface". The actual implementation has been selected in-alignment to the hardware and software platform from which the telemetry data/metrics need to be collected.

   Introduction of Metric Engine enabled decoupling the Service computation from the metrics collection and thus to write service components that can be applied to different devices from different vendors. Consequently, one could write the service components using abstract metrics, these metrics need to have a corresponding entry in the metric engine for the Service to be supported on a new device.

2. Introduced concept of a service component
   - definitions for the expressions related to service components

- ability to perform dynamic reconfiguration of the expressions.

  As for Service dynamic alternatives this essentially means that for the same expressions, several alternatives can be provided, in order of preference. The first one is usually the most detailed and the next ones are fallbacks to use when some metrics are missing. For instance, the first alternative of a Service monitoring a network interface can monitor the interface status, analyze the packet distribution, measure the temperature of the optics, while the last version might just check whether the interface is up. When a new metric is resolved, the Service computation can be reconfigured if the new metric allows to compute a better alternative. For instance, the Service checking the health of an interface might check at minimum whether the interface is up (last alternative) and at best check that the interface is up, that the number of errors is low, that the packet size distribution is stable and that the interface is not flapping (first alternative). The first alternative is more complete but needs less metrics to be resolved than the last one.
- Service components that involve a list of sub-elements to be checked. For instance, a Service that checks the Equal Cost Multi-Path (ECMP) traffic balance towards a given destination needs to be reconfigured when the list of egress interfaces to this destination is updated

3. We have introduced new language for writing service components, Domain-Specific Language (DSL), instead of writing Python code. The health score of a Service has been augmented with symptoms. A symptom is an issue detected by a service component. It is active as long as the Service detects that the issue is ongoing. Should the health score be less than 1 it would essentially mean that Service is not healthy. Such a Service is accompanied by accompanied with at least one symptom explaining what the cause of the issue.

4. We formalised definition of an assurance graph in the YANG model by defining dependencies for computation of the network service health status. Dependencies can be described by leveraging Service status of the relevant network service:

   - Impacting: the score of the dependant is at the most the score of the dependency. (If dependency is broken, the dependent is broken as well)

- Informational: The score of the dependent is not impacting, but the symptoms are propagated

1.5 Methodology

Our methodology is based on the sequence of transformations, a data pipeline: Analysis of the service intent analysis of the Service Definition using Natural Language Processing (NLP) techniques, service decomposition using Directed Acyclic Graphs (DAG) and Heuristics analysis of the resulting graphs in order to reduce amount of data send in the telemetry. Therefore, the methodology framework can be described in more details as follows:

1. Service definition is used as input and analysed by Natural Language Processing (NLP) principles in order to determine service components needed for further processing by the subsequent steps in our data pipeline analysis

2. Analysis and decomposing the configuration of each service instance into an assurance graph, represented by an Directed Acyclic Graph (DAG), which is built leveraging rules defined in the heuristic package. 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 Service instances. Each edge of this graph indicates a dependency between the two nodes at its extremities: the service or Service at the source of the edge depends on the service or Service at the destination of the edge.

3. From the network assurance graph, we derive a so-called expression graph which is a DAG whose sources are constants or metrics and other nodes are operators. The expression graph encodes all the operations needed to produce health statuses from the collected metrics Computing the expression graph from the assurance graph is a simple recursive operation:
    i. if the assurance graph contains a single service component, then build the corresponding expression graph containing
    ii. metrics
    iii. expressions

For instance, each of the service components “X Healthy” and “Y Healthy” independently produce a part of the expression graph

- otherwise take the root (i.e. top-level) nodes and recursively compute the expression graph of each of their dependencies. Then, for each root:
- build an expression graph by combining the expression graphs of its dependencies.
- if the root is a Service combine the expression graph from the previous step with the expression graph produced from the service component.

This graph can be seen as a standard dataflow, functional or streaming computation (inputs at the bottom, outputs at the top).

4. Finally resulting health calculation is exported as Telemetry, using gRPC export, which essentially makes Agent a source for streaming telemetry. Exported telemetry contains statuses of the services and service components, the current assurance graph and the annotated metrics. It also streams partial computation results.

The research is to conduct detailed development and investigation of techniques for Service Assurance for Intent Based Networking in order to notably improve service activation, performance and configuration verification to enable massive endpoint connectivity with reduced latency and cost compared to existing/legacy network technologies.

1.6 Report Structure

This report comprises six chapters:

A Literature Review

Model Driven Telemetry using YANG for Next Generation Network Applications

Mismatch between network service configuration and network service monitoring

Closed loop automation for Intent-Based Networking

Conclusion and Future work

2 A Literature Review

This literature overview chapter aims to provide a comprehensive analysis of the existing research on network assurance for Intent-Based Networking. Through an exploration of the literature, this chapter will examine the various approaches, methodologies, and technologies employed to ensure the assurance of network operations in an IBN environment, Gartner's reference model etc.

The chapter will begin by providing a brief overview of assurance in Intent-Based Networking, highlighting its fundamental principles and key characteristics. Subsequently, it will delve into the concept of network assurance, defining its scope and importance in the context of IBN. The chapter will then present a review of the literature, categorizing the existing research based on different aspects of network assurance.

Assurance of Intent Based Networking Systems

Telemetry collection, data processing and correlation with configuration intent has been topic of interest in various applications. Witnessing evolution in telemetry collection ranging from hardware implemented data collection devices in spacecraft [10] which required complex circuitry even to obtain level-0 telemetry data, focused moved towards the more modern use cases for telemetry.

Such recent paper [11] by J. Pérez-Romero et al. describes challenges in telemetry for 5G small cell deployments. Two main challenging points are outlined: distribution of small cells and high complexity – the large number of metrics and sheer data volume which needs to be collected and analysed. J. Pérez-Romero et al. suggest two key characteristics of the efficient telemetry platform: (i) the capability to produce derived and aggregated information and (ii) ability to acquire, store and make data and information available using a distributed approach. Bond between the intent of network configuration and data collection using telemetry is essentially the missing link that could transform the way network devices are configured and monitored.

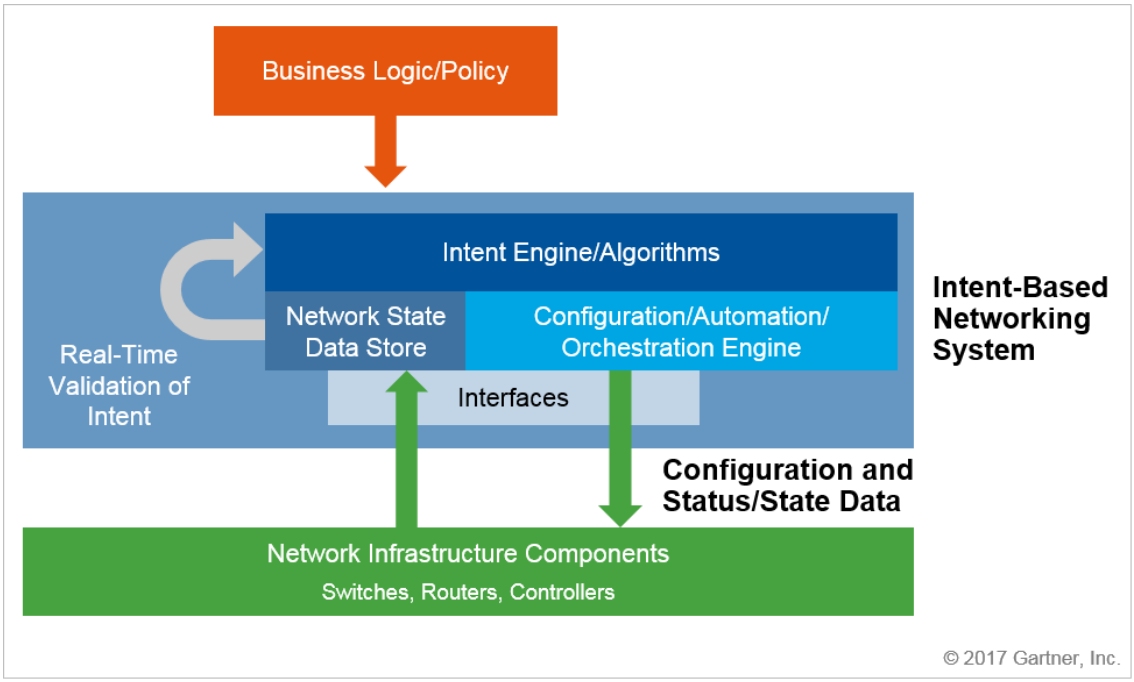

Figure 2 Assurance of Intent Based Networking System - Per Gartner [12]

Gartner [12] has issued a paper and represented their view on the IBN which has been displayed in Figure 2.

According to Gartner IBN should consist of the following four points:

1. **Translation of Intent to Configuration** – The system would take higher-level business policy (*what*) as input from end users and converts it to the necessary network configuration (*how*). The system then generates and validates the resulting design and configuration for correctness.
2. **Automated Implementation** – The system would be able to configure the appropriate network changes (*how*) over the whole existing network infrastructure. This task is usually performed using network automation and network orchestration.
3. **Awareness of Network State** – The system receives information on the network status in real time for all network devices under its control and is protocol- and transport-agnostic.
4. **Assurance and Remediation**– The IBN would validate and assure on continuous basis (in real time) whether the status of the system is matching the original intent by checking all given key performance indicators, If necessary, it would perform corrective actions (change network configuration, send notification, divert or even bock traffic as needed) when desired intent is not met.

Furthermore, Gartner's approach to Intent-based networking (IBN) involves leveraging automation and artificial intelligence (AI) to simplify network management and enhance network agility. The goal is to align the network's behaviour with the intent of the business or user, allowing for more dynamic and efficient network operations.

At the core of Gartner's IBN approach is the Intent Engine, which is responsible for interpreting the intent and translating it into network configurations and actions. The Intent Engine analyses high-level business or user intent statements and converts them into specific network policies or configurations. This abstraction layer allows for intent to be expressed in a more human-friendly manner, rather than requiring detailed technical configurations.

The Intent Engine utilizes various algorithms to understand and interpret intent statements. These algorithms may include natural language processing (NLP) techniques to parse and comprehend human-written intent, machine learning algorithms to analyse historical data and make predictions, and rule-based algorithms to enforce policies and constraints. The combination of these algorithms enables the Intent Engine to understand the context and intent behind the given statements and translate them into actionable network configurations.

Additionally, Gartner emphasizes the importance of closed-loop automation in IBN. This means that the intent-driven network continuously monitors its behaviour and compares it to the desired intent. If any deviations or issues are detected, the network automatically takes corrective actions to bring its behaviour back in line with the intended state.

Overall, Gartner's approach to IBN revolves around the Intent Engine, which acts as the central component for understanding and translating high-level intent into network configurations and actions. The use of various algorithms, including NLP and machine learning, helps in the interpretation and implementation of intent-driven networking.

Gartner's intent-based networking (IBN) focuses on translating high-level business intent into network configurations and actions. While Gartner does not provide specific algorithms for IBN, there are various algorithms used in intent engines to achieve intent-based networking. Here are a few examples:

1. Rule-based algorithms: These algorithms use a set of predefined rules to match intent statements with corresponding network configurations. The intent engine checks if the intent statement satisfies any of the predefined rules and applies the corresponding configuration.

2. Machine learning algorithms: These algorithms analyse large amounts of historical network data and intent statements to identify patterns and predict the appropriate network configurations. This approach enables intent engines to learn from past experiences and improve their accuracy over time.

3. Natural language processing (NLP) algorithms: NLP algorithms are used to interpret and understand intent statements written in natural language. They analyse the syntax, semantics, and context of the statements to extract the desired network outcomes and translate them into actionable configurations.

4. Optimization algorithms: Optimization algorithms are used to find the most efficient network configurations that align with the given intent. These algorithms consider various factors such as network topology, resource availability, performance requirements, and constraints to generate optimal or near-optimal solutions.

5. Policy-based algorithms: These algorithms utilize policies defined by network administrators to guide the intent engine's decision-making process. The intent engine evaluates the intent statement against the policies and applies the appropriate network configurations based on the matching policies.

It's important to note that Gartner's intent-based networking is a concept and framework, so the specific algorithms used may vary across vendors and implementations. The intent engine may incorporate a combination of these algorithms or even employ proprietary algorithms based on the vendor's approach and expertise.

In order to ensure transformation from business logic to the network configuration is as structured and seamless as possible it’s necessary to utilize standardized data models which will be discussed in the next chapter.

Since transformation from business logic into configuration is complex process, it is in turn prone to possible errors and discontinuity between configuration and operational, monitoring capabilities. Consequently, discovering novel service assurance techniques and algorithms using Intent Based Networking (IBN), network function virtualization (NFV) as well as software defined network paradigms are the main objectives of this PhD research project. [13], [14]

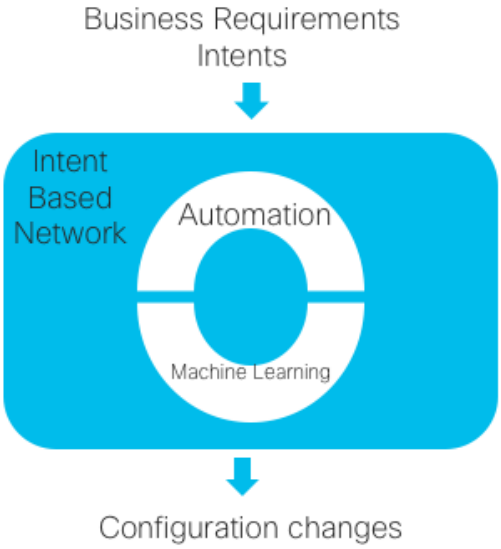

Figure 3 Intent Based Networking to Configuration Changes

Figure 3 illustrates high level and simplified view of transforming business requirements into configuration changes, by means of automation and additional transformation steps – such as machine learning.
NFV orchestrators (e.g., Tacker [13], ONAP [1], CISCO NSO [15]) are a crucial part for the dynamic and optimal management and orchestration of various virtualized network resources (e.g., VMs, Virtualized Network Functions). 5G technology, empowered by NFV and SDN, presents a new dimension of complexity that must be addressed by service assurance [16].

Using this orchestration software having higher level of abstraction, the rapid connectivity and provisioning could be achieved at lower prices while letting to operators possibility to build, arrange and preserve network service [17], [18]. Communication Service Provider (CSP) networks – such as Virtual Evolved Packet Core are subject to very dynamic configuration change. Provisioning, modification and termination of packet data services are being done in

rapid pace in order to keep up with dynamic environment needs and cater to main business drivers, such as IoT, Video etc. SDN technologies using Network Slicing approach are foundation for such a dynamic environment, allowing automated and programmatic configuration of network services [19].

Traditionally network services are being monitored by deployment of probes which generate traffic and provide feedback on the status of the service. Due to such rapid changes in network service configuration, there is open question in regard to monitoring and assuring provisioned services: What is the right approach to take in order to monitor the network which constantly changes? How to ensure network service is operational and carefully selecting probes to monitor network service? [18] [19],

Monitoring using active probes face challenges such as introduction of synthetized traffic within the data flow, end to end monitoring only with no understanding of the data path, lack of comprehension of the configuration intent etc [20] [21] [22]. Generated traffic using probes should resemble real traffic of the network service, however even with almost perfect synthetized traffic, there is substantial possibility that real network service traffic could be impacted, but probe does not detect such a problem since probe is not part of the actual real data flow [23] [24]. Therefore, there is a gap in regard to monitoring and assurance of the actual network service data flow, with all network elements data traverses on the path between endpoints.

Following chapter describes recent developments and methods in the field of model driven telemetry using YANG data modelling language for next generation networking. It also outlines advantages of using YANG models to configure network elements as compared to traditional approaches of using Command-Line Interface (CLI) and Simple Network Management Protocol (SNMP) in order to manage network elements. The explanation of shortcomings of SNMP when it is used for telemetry purposes with the recent scale of modern networks is also given. Finally, this paper provides outline on recent tool chain available to easily produce YANG based code as well as process model-driven telemetry data.

2.1 Recent state-of-the-art solutions

Having a look into state-of-the art solutions, there are few papers proposing in-band network telemetry solution direction [25] [26] [27] [28]. The in-band network telemetry approach uses programmable data planes. In other words, network providers should indirectly detect reasons for delayed and imprecise telemetry data through nodes at network edges. Using In-band telemetry approach, the data-plane packets have extended headers as telemetry instructions. Due to this, in-band telemetry approach could directly collect much more precise telemetry metrics on device level (for instance hop latency or queue length).

Paper [25] proposes a network telemetry platform named NetVision, simulates some aspects of in-band network telemetry, but does not propose a clear approach. The approaches have been proposed in journal papers' articles [26] [27] [29]. The paper [26] improves network orchestrating with in-band data plane telemetry using machine learning. This work is extended in the newest paper of the same research group [27]. The authors in paper [28] improve the in-band network telemetry approach in the direction that they raise run-time network programmability. The results demonstrate that the approach can not only adjust the sampling rate of in-band telemetry in run-time but also could program the corresponding data types dynamically. The authors also confirm that proper accuracy and timeliness for network monitoring could be achieved while greatly reducing the overheads of in-band telemetry. The article [30] presents an exhaustive survey on programmable data planes. Evidence indicates that the closed nature of today's networks will diminish in the future, and open-source and the deep programmability architecture will dominate. In paper [30], the idea of traffic monitoring through in-band telemetry is extended up to the user's end-devices, providing accurate end-to-end latency measurement most demanding in 5G networks. In this way, the end-device becomes aware of its experienced service performance, enabling autonomous operations for faster reactions between edge and cloud.

The main disadvantage of this approach is increasing network delay due to the injection of the telemetry data into data-plane packets. This is most demanding traffic from the network perspective and the approach proposed in this direction could not be scalable and sustainable to the challenges of the future networks. Also, compatibility on data paths is inherently lower if this telemetry approach is implemented.

We'll not review previously mentioned, very relevant solution is Netrounds [31], an Intent-Based Networking and Automated Closed Loop Assurance solution proposed and developer

by the vendor with the same name. Netrounds is a service assurance platform designed to ensure the quality and performance of network services. Its methodology and techniques are focused on end-to-end testing and monitoring, providing real-time insights into network performance, and enabling proactive troubleshooting and optimization. Here is short overview of the strong points that Netrounds provides:

1. End-to-End Testing: Netrounds conducts comprehensive end-to-end testing to validate the performance and functionality of network services. It verifies service availability, latency, throughput, and other key performance metrics across multiple network layers and devices.

2. Active Monitoring: Netrounds solution employs active monitoring techniques to continuously measure and monitor the performance of network services. It actively generates synthetic traffic to simulate real user traffic patterns and captures relevant performance data, providing insights into service quality in real-time.

3. Service-Level Monitoring: Netrounds measures the performance of network services from the perspective of end-users, focusing on service-level metrics such as availability, response time, and packet loss. It allows service providers to assess the customer experience and identify areas for improvement.

4. Real-Time Analytics: Netrounds utilizes real-time analytics to analyze network performance data as it is collected. It provides visualizations, reports, and alarms to highlight performance issues and trends, enabling operators to quickly identify and resolve problems.

However, Netrounds, like most of other network assurance solutions focuses on performing synthetic end-to-end testing on top of the network infrastructure. This essentially implies that synthetic tests are resembling the actual network service traffic, but due to its synthetic nature there is margin of error caused by dissimilarity of synthetic traffic versus the actual network service traffic.

As an example, we could say that performing “ICMP ping” test over the network is different from running the actual ICMP traffic over the same network. Why? Well simply put running synthetic “ICMP ping” is giving us indication of the network service when the specific test is

run, but it is not a formal proof that network performed in the same manner in the prior or next instance of time. Furthermore, synthetic test is introducing additional data to the in-band network traffic, which is different from the approach we've taken in our work.

2.2 Service Assurance techniques based on Open-Source orchestrators.

Developing any modern software solution almost inevitably involves some sort of open-source component. ONAP [1] community represents 70% of global subscribers [32], with large number of service providers participating in the project and testing or contributing. However, this still doesn't necessarily mean that mentioned service providers are using ONAP as orchestration platform in their production deployments.

ONAP project is launched in March 2017 as merger of two projects:

- OpenECOMP – project Lead by AT&T. It's open-source version of AT&T's ECOMP platform. ECOMP is software framework used by AT&T to manage lifecycle of virtual network function (VNFs) in their software defined network. ECOMP stands for Enhanced Control, Orchestration, Management & Policy and
- Open-O – Lead by ChinaMobile and contributed by large Asian vendors ZTE and Huawei. Open-O is also a platform for orchestrating VNFs and services over various platforms and equipment built by different vendors.

At this time, ONAP is managed by Networking Fund of the Linux Foundation. From ONAP's mission statement, whose excerpt we could find here:

"The mission of the Project is to create a comprehensive platform for real-time, policy-driven orchestration and automation of physical and virtual network functions that will enable software, network, IT and cloud providers and developers to rapidly automate new services and support complete lifecycle management. …" [32]

and taking into consideration that project is backed by most of the world leading vendors and service providers, one could conclude that ONAP is poised to become industry-wide framework for rapidly enabling network services and virtual functions. Additionally, having in mind that ONAP scope also covers defining analytics and policies which are expected to be used in real-time, including reference interfaces and requirements for telemetry on VNFs it's

clear that such a platform deserves serious analysis and consideration. It’s with no doubt clear that leveraging such a powerful platform, backed by many important global vendors and service providers could enable access to large number of test labs and environments to verify our work.

To further affirm mentioned point on relevance and global significance of ONAP one can use retrieve statistical data from Bitergia, which is used as metrics platform to track code contributions and evolution of the project. From what we could identify in the field of research and relevant articles, there are many different vendors and contributors participating in the development of the ONAP open-source project.

At the time of writing this article, ONAP grew into massive project with over 20 million lines in 273 repositories and 1100 contributors. AT&T’s contribution to the ONAP is still very substantial. However, presence and contributions by other organizations is also becoming significant.

Before diving into describing assurance techniques, it's necessary to review architecture of the whole ONAP as solution, so that it's better understood where does assurance fits in the whole picture.

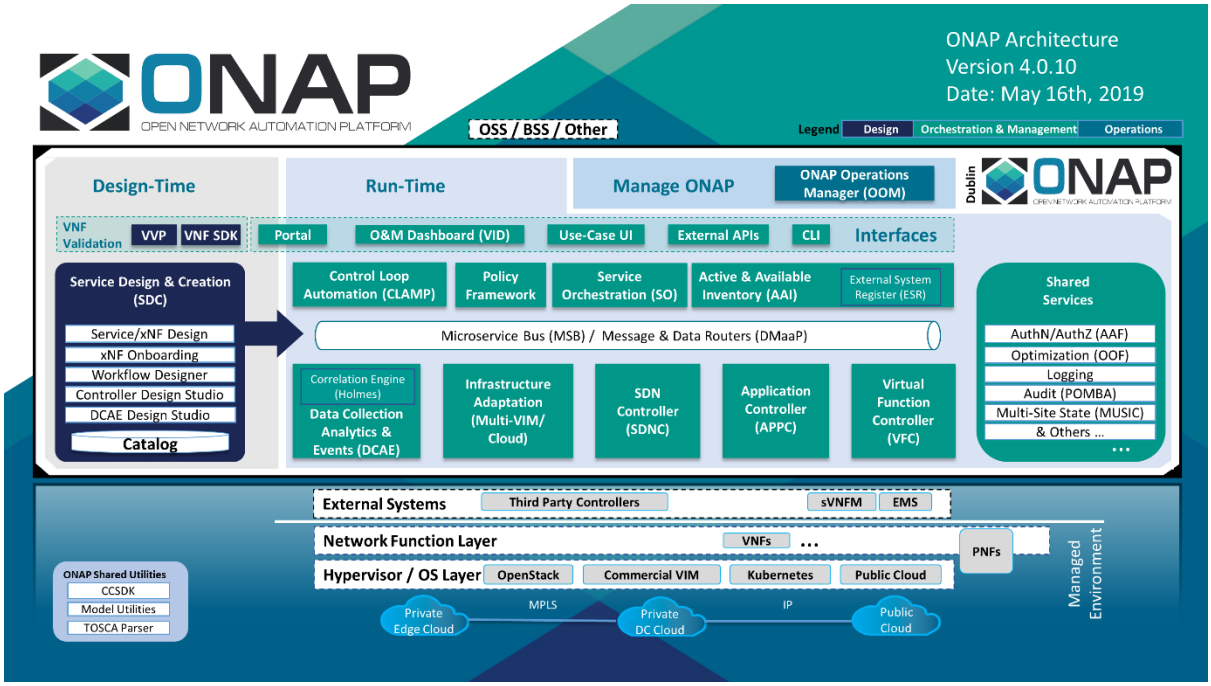

Figure 4 ONAP Platform architecture (4th release - Dublin) [1]

As depicted on Figure 4, ONAP is composed of following high-level components:

- Design Time Framework
- Runtime Framework
- ONAP Management
- Shared Services

Figure 5 provides simplified view to the ONAP functions and outlines role of the key components:

1. Northbound based - RESTAPIs for Standard based interoperability with external platforms
2. Integration with Cloud Providers using OOM which enables managing of the native cloud installation and deployment to Kubernetes-based cloud
3. Shared services for ONAP such as Optimization Framework (OOF) which enables declarative approach, policy-driven method for creating and optimizing applications for

VNF placement, change and scheduling management. Logging and audit as centralized capability is also provided.

4. Harmonization of industry models defined by Standards Development Organizations (SDOs) such as ESTI NFV MANO, VMF SID, ONF Core, TOSCA, IETF and MEF.

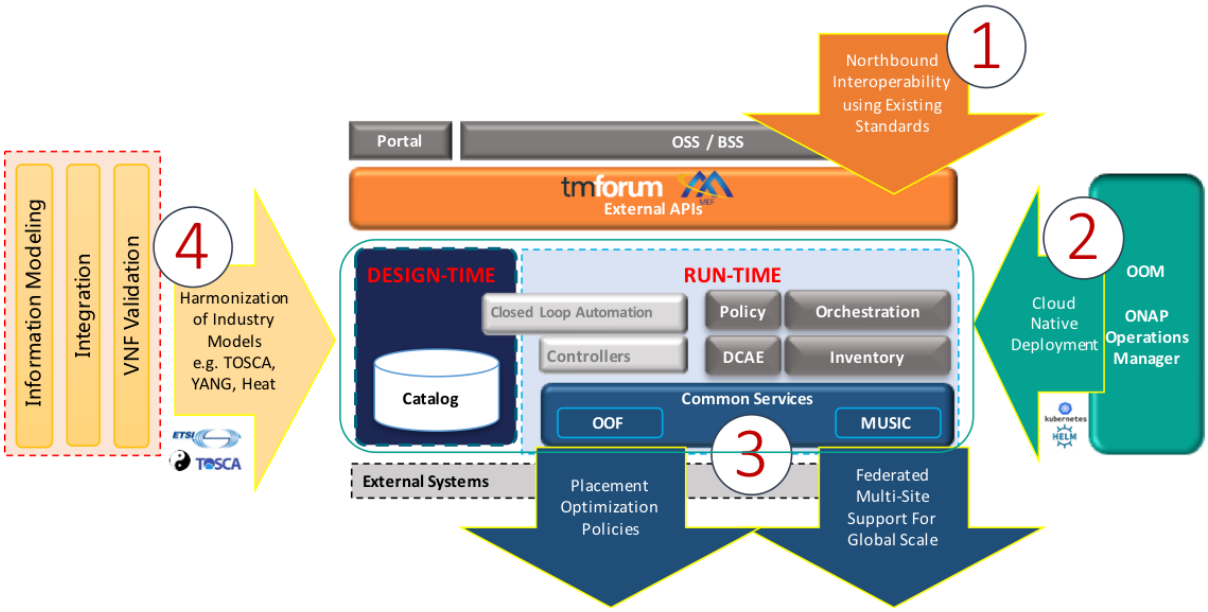

Figure 5 ONAP functional overview [1]

This is relevant in order to understand relationships between ONAP and other services and integration points, including assurance components relevant for our project.

CLAMP, which stands for Closed Loop Automation and Management for Processes, is a project within ONAP framework and it is framework or methodology used in network automation. It involves the integration of automation technologies to create a closed-loop system that enables continuous monitoring, analysis, and automation of processes in a network.

In simple terms, CLAMP aims to automate the entire lifecycle of processes in a network, from monitoring and analysis to decision-making and execution. It combines various automation technologies such as artificial intelligence, machine learning, and data analytics to achieve this goal.

The closed-loop aspect of CLAMP refers to the feedback loop created within the system. It means that the automation process continuously collects data, analyses it, makes decisions

based on the analysis, and then implements those decisions. This closed-loop system ensures that processes in the network can be constantly optimized and adjusted based on real-time data and analysis.

By implementing CLAMP, organizations can enhance operational efficiency, reduce manual errors, improve network performance, and enable faster decision-making. It is particularly useful in complex network environments where manual management and troubleshooting may not be feasible or efficient.

Several design and runtime elements of ONAP are in charge of closed loop automation. Functionality for Fault, Configuration, Accounting, Performance, Security (FCAPS) is provided collectively by Control Loop components:

- Data Collection, Analytics and Events (DCAE) is in charge of collecting configuration, usage and performance data
- "Holmes" task is to connect to DCAE and enable alarm correlation on the whole platform

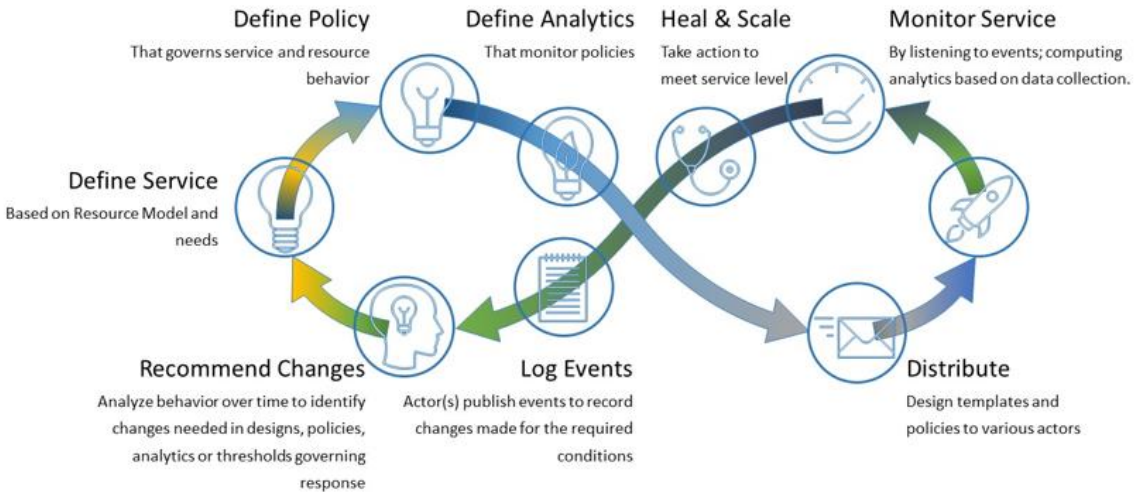

Figure 6 High level overview of Closed Loop Automation as defined in ONAP [1]

Figure 6 graphically represents stages in closed loop automation process, from service definition to service configuration, monitoring, observing events and change recommendations. It's preferred that remediation changes are fully automated, but sometimes manual intervention of the operator is needed to perform the change.

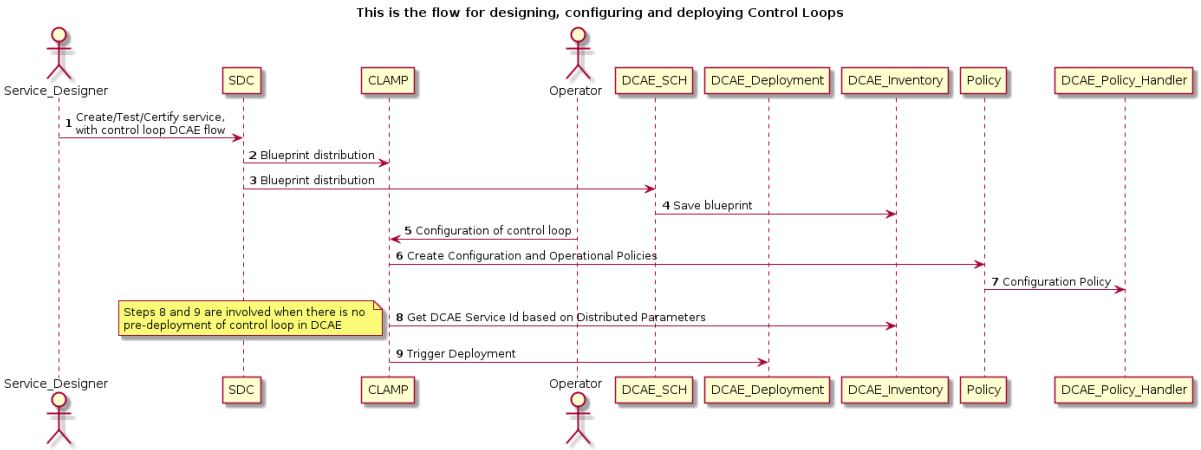

Figure 7 CLAMP flow, depicting design, configuration, and deployment of Closed Loops

Sequence diagram provided on Figure 7 outlines interactions between components and actors in closed loop automation.

From mention these reasons and techniques explained above, which could be summarised as follows:

- improving telemetry efficiency by compression, yet still retaining large quantity of data which is present, but not reelvant for the status of the network service
- large quantity of telemetry data being sent from the edges to the hub location, hence causing even more load on the network infrastructure
- drawing conclusions on the state of network service by leveraging synthetic tests in top of existing network services traffic, and thus introducing margin of error in the actual network traffic

we could conclude that further advantages could be achieved by introducing our assurance techniques which are different in following ways:

- improved relevance of the telemetry data by introducing network service-aware relevant telemetry fields in the model-driven telemetry
- performing telemetry analysis at the network edge, within service aware agents and thus avoiding sending large quantities of telemetry data to central location
- avoiding leveraging sythetic tests and hence eliminating introduction of margin of error
- introducing service assurance language (DSL) which could simplify assurance expressions

Our assurance techniques and approach imply introducing a level of abstraction into telemetry, and then exporting only that newly created abstraction. Essentially this means sending less data, but more relevant data which has been already processed at the network edge. The papers [33] [34] [35] [36] [37] [38] have tried to solve the defined problem. The main problem of these state-of-the-are solutions is there is a lack of evidence regarding performance of the mentioned semantic definition-based approaches. Model driven telemetry inherently enables the real-time collection of hundreds of thousands of counters on large-scale networks, with contextual information to each counter provided in the telemetry data structure definition. Explaining network events in such datasets implies substantial analysis by a domain expert. There is obviously that there is a huge potential of this solution direction type of approaches, but still a lot of questions that need answers in future research & development. Therefore, it is a great time to dedicate a thesis to this topic especially because artificial intelligence and machine learning are developing incredibly fast nowadays and could be an excellent basis to outstandingly mitigate problems related to telemetry in next-generation networks.

3 Model Driven Telemetry using YANG for Next Generation Network Applications

It is difficult to imaging service assurance without good quality measurements and data from the assured network elements. Measurements are carried as data stream within Model Driven Telemetry (MDT). Models for the telemetry are defined using YANG data modelling language for next generation networking. There are also several advantages of using YANG models to configure network elements as compared to traditional legacy approaches of using Command-Line Interface (CLI) and Simple Network Management Protocol (SNMP). The explanation of shortcomings of SNMP when it is used for telemetry purposes with the recent scale of modern networks is also given. Finally, this chapter provides outline on recent tool chain available to easily produce YANG based code as well as process model-driven telemetry data.

3.1 Model Driven Telemetry (MDT) Overview

For the past 25 years, Simple Network Management Protocol (SNMP) and Command Line Interface (CLI) were main sources of operations data exposed by the network elements. As networks grew in size and complexity, SNMP and CLI became insufficient and impractical to address the challenges of operational data modelling, acquisition and processing.

Configuration of network elements is traditionally done using CLI, making it convenient for human interaction. As a result, retrieval and processing of operational data from network elements was also done using CLI. Such approach is consequently leading to significant effort invested in parsing unstructured information received from CLI. It is relatively easy to develop scripts to automate configuration tasks done via CLI, however maintaining such scripts is becoming increasingly complex and costly since there is continuous effort to keep automation scripts up to date with little guarantee that CLI structure remains the same over different software versions on the network devices. Even though there are SNMP writable management information base (MIB) models which are allowing configuration to take place, in practice writable MIBs are either not being implemented or not adopted at large scale.

When it comes to monitoring, there is of course SNMP as logical choice to poll statistics and operational data, however it's clear that configuring network elements using one method – CLI and data statistics polling using another method – SNMP introduces additional levels of complexity, which is determining if CLI supported configuration has exposed capability to be monitored by SNMP. Additionally, it is unclear whether SNMP MIB needed for data monitoring has been implemented on the actual monitored devices within the required software versions. It is very common that some objects defined in SNMP MIB could be omitted from the implementation on the network device making it difficult to answer the question: "Given the CLI configuration, can it be monitored using SNMP at all? What are the available objects which would be relevant for the monitoring?". To illustrate given question through one simple example: After polling an OID from router using SNMP value of 0 is returned. Value 0 could well be its valid counter with the value; however it could also mean that actual OID is not implemented. The advantage of using Network Configuration Protocol (NETCONF) as a protocol and Yet Another Next Generation (YANG) as data-modelling language is that is provides list of the supported models in the HELLO message in NETCONF, hence making it very clear which models are supported by the management device. For a client it's sufficient to connect to device via NETCONF protocol in order to obtain list of supported YANG modules. Such a straightforward approach of using HELLO message came as a result of

experience and improvements done in IETF. Changes in regard to HELLO mechanism identifying supported YANG modules came though as RFC 7895, YANG Module Library.

Taking into account that IETF is the standard development organization that develops specification SNMP and MIBs as well as for NETCONF and YANG it is clear that focus on configuration management is moved from writable MIB modules to NETCONF/YANG with the following statement issued by IETF's body - IESG: "*IETF working groups are therefore encouraged to use the NETCONF/YANG standards for configuration, especially in new charters".* [39]

Using Yet Another Next Generation (YANG) – model-driven approach both for configuring network elements as well as for monitoring those same network elements the requirements for additional levels of complexity to translate data from one protocol to another or to parse unstructured data has been eliminated. YANG is a full, formal contract language with rich syntax and semantics to build applications on.

In period of 2010 until 2018, YANG data modelling language has undergone extensive developments since it was firstly introduced by Internet Engineering Task Force Request for Comment 6020 (IETF RFC 6020) [40] as a data modelling language for the Network Configuration Protocol (NETCONF) [41]. YANG as data modelling language is defined in IETF RFC 6020 (YANG 1.0) with improvements learned over the years incorporated in form of YANG 1.1 in RFC 7950. YANG became de facto standard as the modelling language for configuration of network devices.

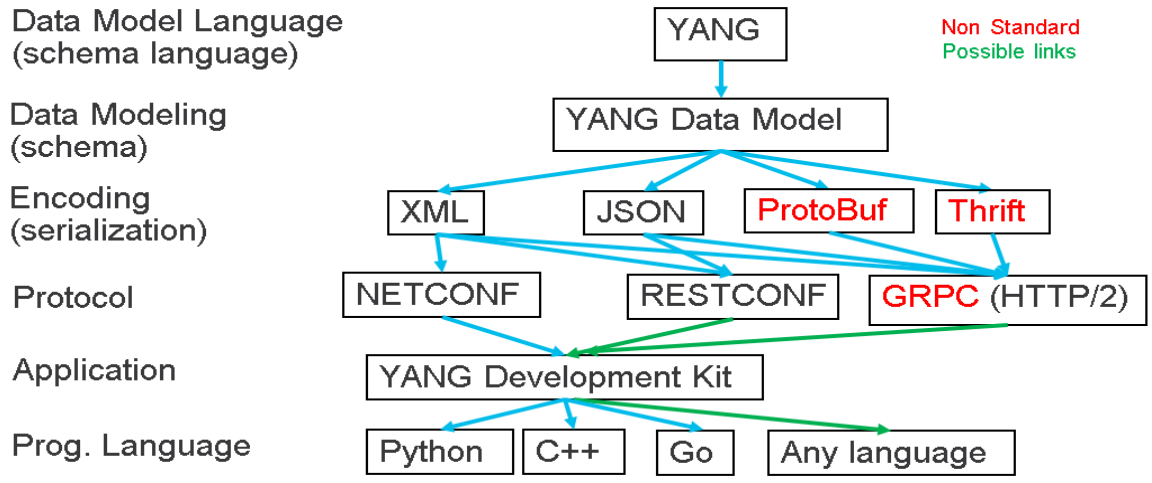

Figure 8 YANG and related protocols

Figure 8 outlines the relationship between YANG as data model language and possible encodings, protocols that could be used in implementing applications in different programming languages.

As YANG data modelling language adoption grew in the industry RFC 8199 [42] provided guidelines on YANG module classification in two dimensions and additional subdivision within the dimensions:

1. YANG Module Abstraction Layers
    1.1. Network Element YANG modules
    1.2. Network Service YANG modules

2. YANG Module Origin Types
    2.1. Standard YANG Modules
    2.2. Vendor-Specific YANG Modules and Extensions
    2.3. User-Specific YANG Modules and Extensions

IETF as the main driving force behind development of YANG has foreseen "tsunami of YANG models" which was materialized in over 320 standard YANG models and additional 10099 vendor modules as it exists at the time of writing this article. [43]

Figure 9 Growth of IETF YANG Models since 2015 depicts growth of total number of IETF YANG modules developed over time along with its compilation results. Initially there were quite a high number of compilation warnings compared with successful compilations. As industry started to adopt modelling language and gains experience in writing new modules using YANG, successful compilations are following the growth trend.

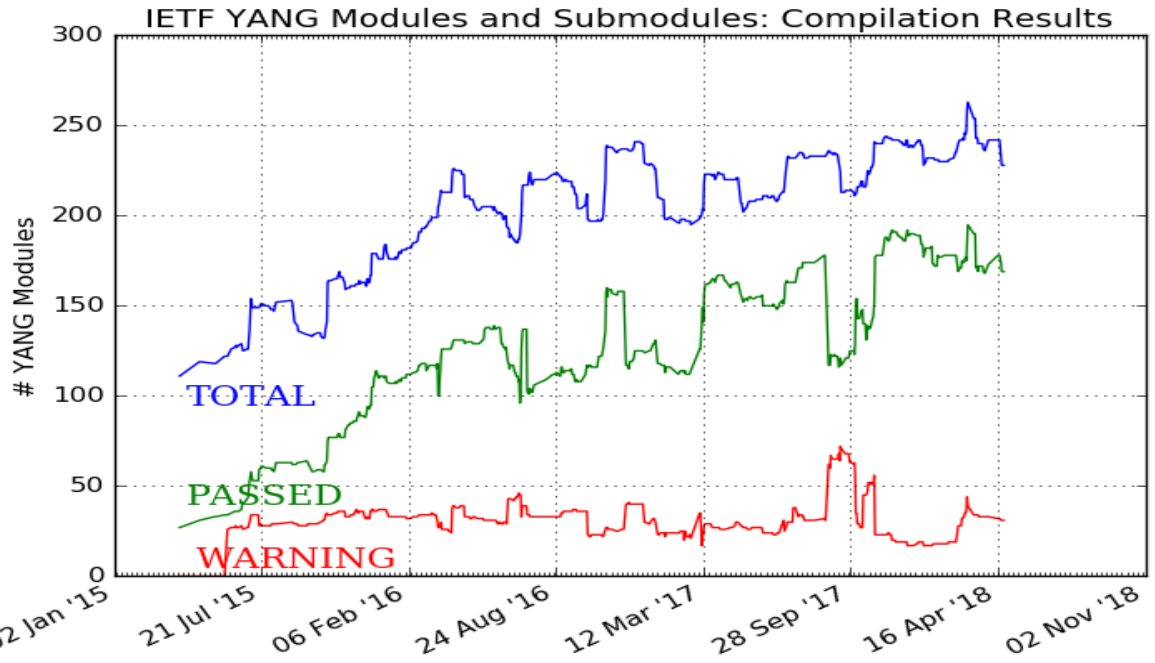

Figure 9 Growth of IETF YANG Models since 2015 [42]

With growing popularity of YANG modelling language there are significant number of models built by different organisations. Consequently, relationships and interdependency between YANG modules is becoming increasingly complex. There are number of tools created on yangcatalog.org [42] to facilitate search, use, creation and collaboration on YANG models.

3.2 Model Driven Telemetry

For long period of time acquisition of network state information is done by polling data from the network devices or by using specialized network protocols. Since YANG can also be used to model state data of the network elements there is tendency to use YANG models for telemetry.

Network monitoring based on legacy protocols such as SNMP is long overdue for significant improvements, as it has been designed for legacy implementations. Modern network deployments with high-density platforms require much higher scalability, performance, agility and extensibility. Streaming telemetry addresses major challenges outlined above, where data is being exported – streamed from network elements continuously with incremental updates. Operators may use data models to subscribe for specific data points needed. Comprehensive list of the requirements for telemetry are provided in the RFC 7923 [44] *"Requirements for Subscription to YANG Datastores"*.

By using YANG and Model-Driven Telemetry, issues with SNMP polling have been addressed. Telemetry is more efficient over SNMP since it there is no poll-response process altogether. By simply eliminating the polling and thus avoiding need for network element to wait for the next instruction significant gain is achieved, since network element does not need to wait for the instruction on what to do after every export cycle. Configured telemetry policy already instructs network element onto what data to collect, how often and where it should be exported.

YANG as data modelling language has been introduced and defined in IETF. However, there are currently 2 two main organizations that steer Model-Driven Telemetry: Openconfig [44] and IETF .

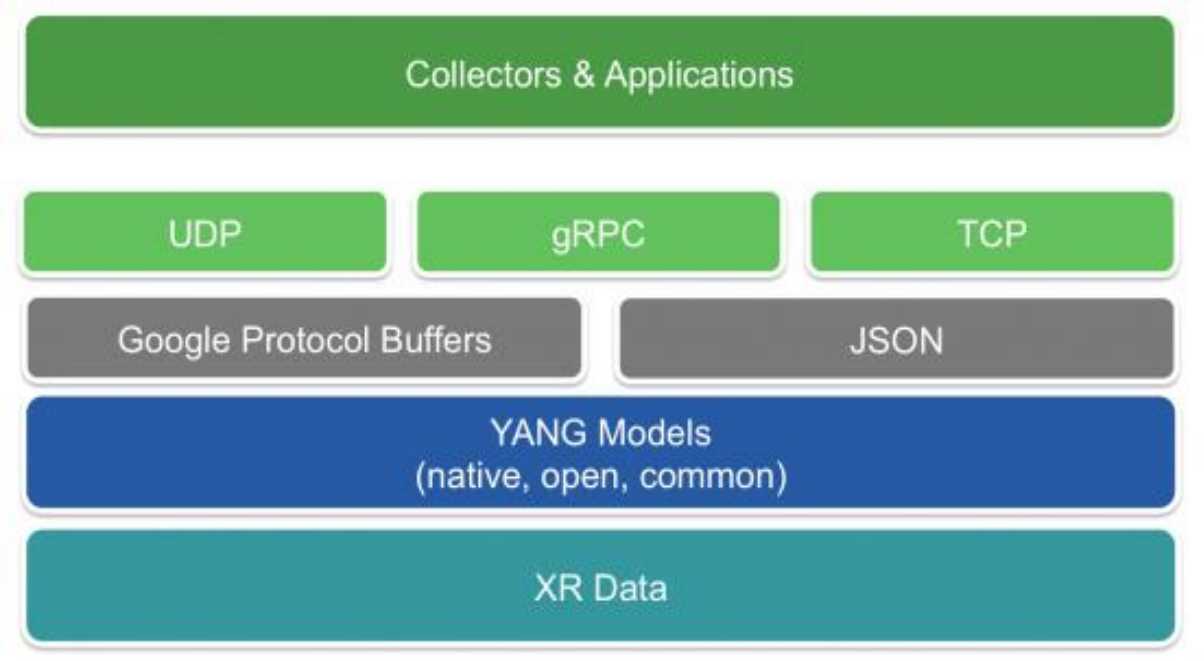

Figure 10 Block diagram of high-level view of Model-Driven Telemetry and components [44]

Figure 10 depicts block diagram of Model-Driven Telemetry (MDT) and relationship of different layers in relation to MDT and YANG models.

For both IETF and Openconfig same architecture applies in terms of Telemetry data acquisition and processing, with following components/layers as part of the simplified solution:

- Collection Layer

Tasked with receiving telemetry stream, aggregating and normalizing data for further processing

- Storage Layer

Performs store, indexing and search functionalities

- Applications Layer

Takes care of automation, visualizing and alerting

For Model-Driven Telemetry there are important options available which need to be taken into account for session initiation, encoding and transport.

Session Initiation: Two distinct options are available for initiation of a telemetry session.

- "dial-out" - Network device initiates connection to the Collector
- "dial-in" – Collector initiates session to the network device

As depicted at Figure 10, components of telemetry publishing and consumption model-driven telemetry are defined in a way to make it versatile, performant, and flexible. Aspects of this design are discussed below.

Transport carries telemetry data using UDP, TCP or gRPC over HTTP/2. TCP and UDP transport protocols offer simplicity, however gRPC brings TLS encryption as security advantage.

Encoding: Network element streams telemetry data using2 types of encoders: JavaScript Object Notation (JSON) or Google Protocol Buffers (GBP).

JSON encoding is performed over TCP transport service and can optionally be compressed by zlib.

GBP offers both UDP and TCP as transport service with two encoding formats: Compact GBP and Key-value.

With Compact GBP encoding will be the most efficient, but it requires unique ".proto" file for each YANG model/path used by the receiver to decode the data.

With key-value encoding single ".proto" file is used to encode data in self-describing format. However, data on the wire is much larger when compared to compact mode, since with key-value encoding names are included in the message content.

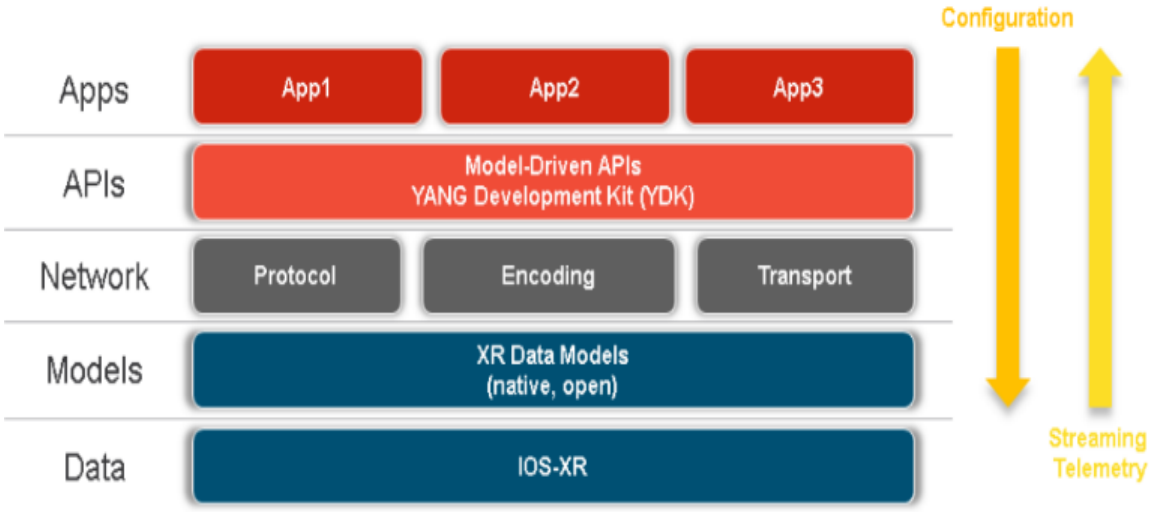

Figure 11 Streaming Telemetry architecture on Cisco XR platform

Figure 11 represents design architecture of both configuration using YANG models as well as publishing of telemetry data using operational YANG models. This information is relevant since we're using IOS XR based routers in the experiments conducting in this research.

3.3 YANG Tools

Over the past several years, as YANG gained adoption, there has been significant advancements of the tools and utilities which make it easier to develop applications and generate code leveraging YANG models. In the following section we'll name some of the most widely used tools and how are they helping developers and users.

YANG Development Kit (YDK) [45] is a Software Development Kit which helps generate APIs based on YANG Modules and can produce output in different programming languages. While data models provide structured representation of capabilities of the network device, creating functional code leveraging data models could be quite demanding since there are several other points to address, such as protocol, encoding, transport. Therefore, having SDK which can help generate APIs and code templates could be quite useful and this is exactly where YDK excels.

YANG Catalog [42] is a tool and searchable database which can be accessed to find relevant models for the desired use case. Typical use case would be finding relevant YANG module for development of new application, where user could search correct module by name, keyword in module content, by vendor etc. YANG Catalog provides capability to download complete

PNDA complex architecture for Telemetry data acquisition, processing and visualization. PNDA is based on open-source components and designed in such way to decouple data sources from applications. Such an approach enables seamless integration of data sources and data reuse by different data-analytics applications without impacting data sources. As example one could indicate polled data source – such as SNMP Agents, which are usually polled heavily by multiple consumers. With high performance pub/sub bus which exists in PNDA, polling would be done only once, and data made available to all consumers.

Pipeline is an Open-Source collector tailored for Streaming telemetry with capability to export data to multiple destinations. It can be used as part of PNDA architecture as well as standalone application. Of course, true power of collector is exhibited in its capability to efficiently process large quantities of telemetry data and make it available for easy processing by consumers or even better by streaming publish and subscribe platform such as Apache Kafka, which is also one of the main components of PNDA architecture.

Ncclient [46] is a Python library for NETCONF clients which is in use for quite a while and facilitates development of Python based tools leveraging NETCONF protocol. Since all NETCONF protocol operations are implemented in Ncclient library, this enables quick network element configuration retrieval, editing, committing and so on, which helps maintain model driven telemetry configuration using relatively simple code base.

3.4 Conclusion on MDT and YANG versus legacy monitoring

Model-driven Telemetry (MDT) using YANG models is undoubtedly huge step forward and the path to take to enable advanced data analytics and machine learning applications. With proper data structures provided by MDT, infrastructure for applications such as PNDA conditions have been created to focus on data consumption and harvesting information contained in vast amounts of data, rather than focusing on preparing and parsing data, so it could be consumed. Over time, intensive data polling, retrieval and parsing of unstructured data is going to be practiced certainly too much lesser degree, probably only to support some legacy networking deployments – if any.

Some of the relevant points have been made in the document which could prompt questions such as: Will SNMP disappear?

Short answer is: No. SNMP and MIB models did solid job for monitoring. SNMP MIBs are configuration and state information but represented in a way that is unsuitable for configuration.

NETCONF/YANG is doing better job enabling data model driven configuration management. With addition of model driven telemetry there are numerous advantages both in terms of performance and especially in automation segment, enabling much faster configuration as well as telemetry data acquisition.
In the following chapter we'll discuss

4 Mismatch between network service configuration and network service monitoring

What is the mismatch in question and where is it stemming from? Simple answer is that mismatch is caused by the distinct configuration options and models used between configuration and monitoring/operations. When the same YANG model defines both the configuration data and the operational state data, then establishing relationship between configured object and operational data is relatively easier. This unified approach can simplify the task of designing and managing network devices and services, as it ensures consistency between the configuration and operational state.

However, the exact use of YANG models may vary between different implementations and vendors. Some might choose to have separate models for configuration and monitoring. Establishing a relationship between network service configuration and the YANG models used for configuration and monitoring is common challenge and can be considered complex depending on the specific requirements and context.

1. Design Complexity: Creating YANG models that accurately represent the configuration, operational state, and statistics of network devices and services requires a deep understanding of the network domain and the specific requirements of the devices and services being modelled. This can be a complex task that requires careful design and expertise.

2. Standardisation Many organizations adopt standardized YANG models developed by industry groups like the IETF. This can simplify the relationship between network service configuration and the YANG models because the models are already designed to represent common network concepts and practices.

3. Tool Support: There are tools available that can aid in the creation, validation, and implementation of YANG models. These tools can make it easier to establish and maintain the relationship between the network service configuration and the YANG models by providing features like syntax checking, simulation, and code generation.

4. Interoperability: When working with network elements and services from different vendors, aligning their diverse configuration practices with standard YANG models might be challenging. Different vendors may implement or interpret the YANG models slightly differently, leading to potential inconsistencies.

5. Customisation: In scenarios where custom or specialized configurations are required, the process of designing and implementing the YANG models can be more complex. It might necessitate a deeper collaboration between network engineers, architects, and developers to ensure that the YANG models accurately represent the unique aspects of the network service configuration.

Therefore we could conclude that relationship between network service configuration and YANG models can also be a complex task that requires specialized knowledge and careful design, especially when dealing with custom or unique configurations or when aiming for interoperability across diverse systems.

Let’s take following configuration for example. This XML example represents service definition as provisioned by Cisco Network Service Orchestrator (NSO) [15] [47]. Network service configuration is represented as XML document specifying multiple components as per YANG module definition.

```
<devices xmlns="http://tail-f.com/ns/ncs">
    <device>
        <name>sain-pe-1</name>
        <config>
            <interface-configurations xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg">
                <interface-configuration>
                    <active>act</active>
                    <interface-name>tunnel-ip12</interface-name>
                    <vrf xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-infra-rsi-cfg">vrf1</vrf>
                    <ipv4-network xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ipv4-io-cfg">
                        <addresses>
                            <primary>
                                <address>10.0.12.1</address>
                            </primary>
                        </addresses>
                    </ipv4-network>
                    <tunnel-ip xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-tunnel-gre-cfg">
                        <source>
                            <address>10.0.0.13</address>
                        </source>
                        <destination>
                            <address>10.0.0.14</address>
                        </destination>
                    </tunnel-ip>
                </interface-configuration>
            </interface-configurations>
            <vrfs xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-infra-rsi-cfg">
                <vrf>
                    <vrf-name>vrf1</vrf-name>
                    <create/>
                </vrf>
            </vrfs>
        </config>
    </device>
```

Figure 12 Cisco NSO Network Service configuration for device pe-1

Figure 12 Is XML formatted output generated by NSO, representing network service configuration to create Tunnel service over the network topology discussed in this work. XML configuration is leveraging YANG models to instil state on the network element – PE router running Cisco IOS XR operating system.

On Figure 13 depicted below, we could see relevant configuration for the second tunnel endpoint located on the device pe-2. Both configuration excerpts are aimed to create configuration state on the respective network elements, which are part of the network service tunnel-ip12.

```
    <device>
        <name>sain-pe-2</name>
        <config>
            <interface-configurations xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg">
                <interface-configuration>
                    <active>act</active>
                    <interface-name>tunnel-ip12</interface-name>
                    <vrf xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-infra-rsi-cfg">vrf1</vrf>
                    <ipv4-network xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-ipv4-io-cfg">
                        <addresses>
                            <primary>
                                <address>10.0.12.2</address>
                            </primary>
                        </addresses>
                    </ipv4-network>
                    <tunnel-ip xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-tunnel-gre-cfg">
                        <source>
                            <address>10.0.0.14</address>
                        </source>
                        <destination>
                            <address>10.0.0.13</address>
                        </destination>
                    </tunnel-ip>
                </interface-configuration>
            </interface-configurations>
            <vrfs xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-infra-rsi-cfg">
                <vrf>
                    <vrf-name>vrf1</vrf-name>
                    <create/>
                </vrf>
            </vrfs>
        </config>
    </device>
</devices>
```

Figure 13 Cisco NSO Network Service configuration for device pe-2

However, even if provided confirmation template is being sent to the network elements (devices) and configuration accepted by the devices, no errors reported, this still doesn't mean that network service will be functional. Basically, even though configuration is correct, desired network service might be down and non-functional due to many reasons. Generally establishing status of the service – performing service assurance and troubleshooting the root cause is increasingly complex task.

In order to address the challenge – assure and continuously monitor network service, monitoring loop should be closed using telemetry, so that appropriate data could be used to better understand service status.

Let's examine the configuration models, examples given in Table 1 and consider the relationship between configuration models used and the models used to monitor the configuration.

For configuration, Cisco IOSXR native YANG models are used:

Configuration Component	YANG Model
Interface	http://cisco.com/ns/yang/Cisco-IOS-XR-ifmgr-cfg
VRF	http://cisco.com/ns/yang/Cisco-IOS-XR-infra-rsi-cfg
Network	http://cisco.com/ns/yang/Cisco-IOS-XR-ipv4-io-cfg
Tunnel	http://cisco.com/ns/yang/Cisco-IOS-XR-tunnel-gre-cfg

Table 1 YANG Models used for the configuration.

However, mentioned YANG models are used for configuration only. In order to find out which parameters to monitor it's needed to determine which YANG models provide relevant operational data. Additionally, it's possible to perform monitoring using legacy approach – such as Simple Network Management Protocol (SNMP), Command Line Interface (CLI) or other means.

4.1 Decomposition of the network service configuration

In order to understand which data is available for monitoring and how to monitor the network service in question it's necessary to perform service decomposition and establish which service components

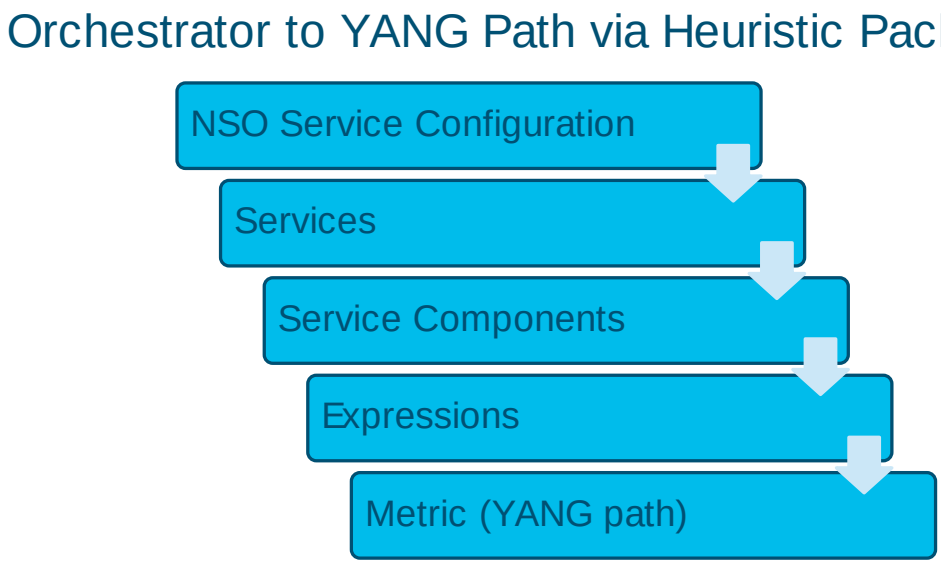

Figure 14 From Orchestrator (NSO, ONAP) Service Configuration to YANG path via Heuristic Package

Figure 14 Depicts relationship between different stages in network service decomposition, from higher towards lower layers, including the actual metrics collected from the network elements.

Example of the Assurance Expression Tree

- Assurance hierarchy between following Service components
    - Service components: tunnel, VPN, ...
    - Service components: (sub)interface, device, interior routing protocol
    - Service components can depend on other service components
    - Service components are assured on the agents

The above-mentioned decomposition is illustrated on Figure 15**.** where GRE Tunnel network service has been decomposed to its Service components. GRE Tunnel network service obviously depends on health of Tunnel Interface, which in-turn depends on the underlying physical/logical interface. Ultimately, underlying interface health will also depend on the device where the particular interface resides. Apart from the GRE Tunnel Interface, GRE Tunnel as a service also depends on the Layer3 connectivity, which is also dependant on the health of interior routing protocol on the device itself. Finally, routing protocol's health

depends on the health of egress interface, which is forward the network traffic, along with the routing protocol information as well.

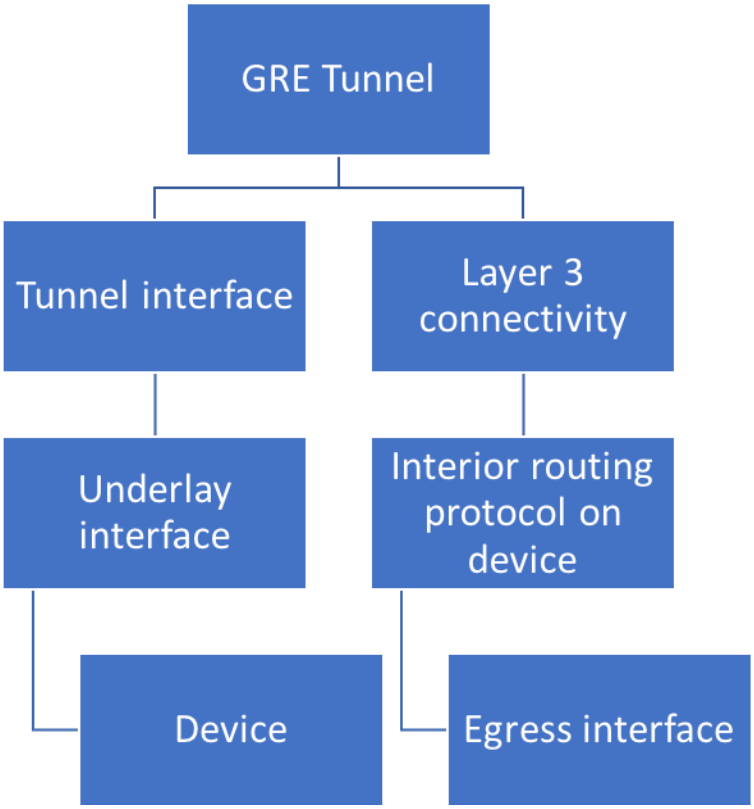

Figure 15 GRE Tunnel Network Service assurance decomposition

Figure 15 represents graphically dependencies and decomposition of the actual tunnel service, along with all its service components. Each box represents the result of the evaluation for the particular service where boxes coloured in Green represent service components which have been determined as healthy based on the telemetry data, CLI or SNMP outputs, whatever it may it depend upon. Grey boxes represent service which state couldn't be conclusively determined, since there is insufficient data to deem Service healthy or unhealthy

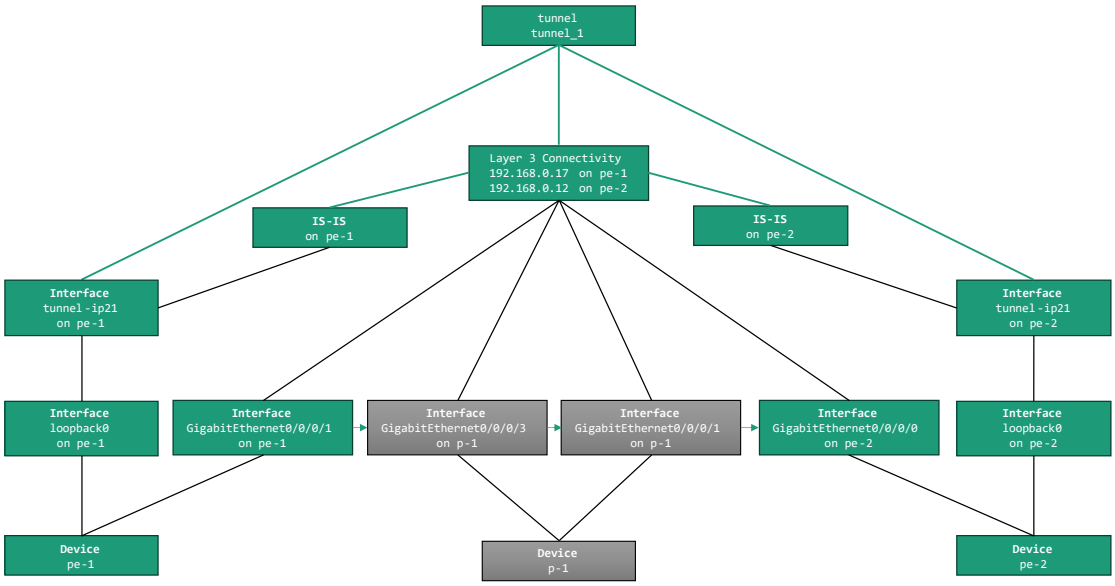

Figure 16 Network Service Assurance Tree

Figure 16 represents detailed view of the network service assurance tree and service components related to the network service, in the example it's tunnel service. Health and assurance tree of the whole network service depends on the status of the service components, which are integral part of the network service. We perform assessment of the health status of each of the service components that are taking part in the network service.

Service Component Assurance Expression for health of a service component – interface:
Values marked in Red are raw data received from the network elements by means of Model Driven Telemetry (MDT), SNMP, CLI etc. Values in Blue colour are the derived and calculated to be used as intermediary value for the final calculation. Once all needed values have been obtained, then we can perform the calculations and inference on the status of the service component.

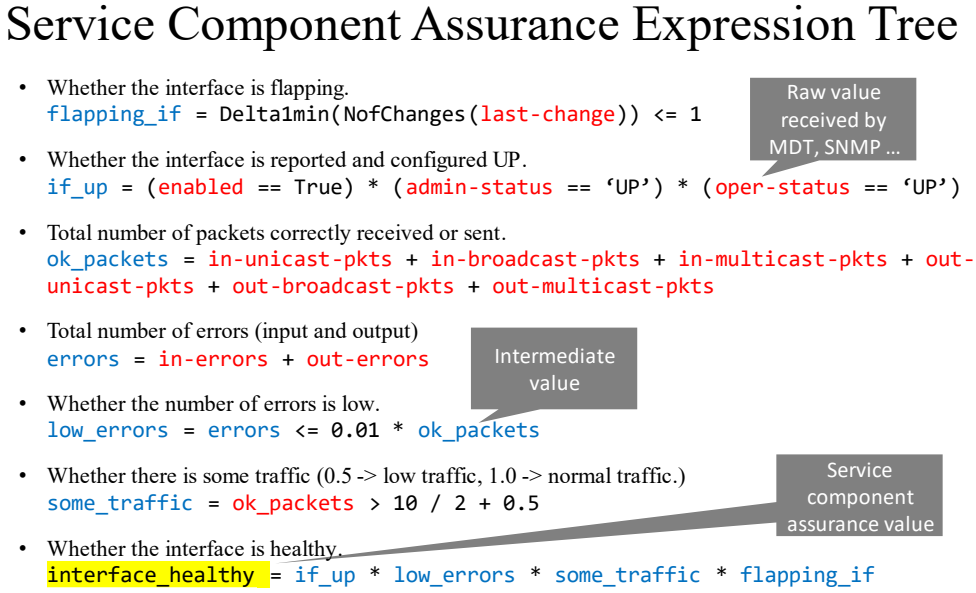

Figure 17 Service Component Assurance Expression Tree

Above mentioned expression and conditions are graphically represented on the Figure 17 where it can be observed that interface could be considered as healthy (health=1.0, meaning 100% healthy) if all Service components are also healthy: interface is up, number of errors is low, there is traffic on the interface and interface is not flapping. Same procedure is then performed on each of the

\

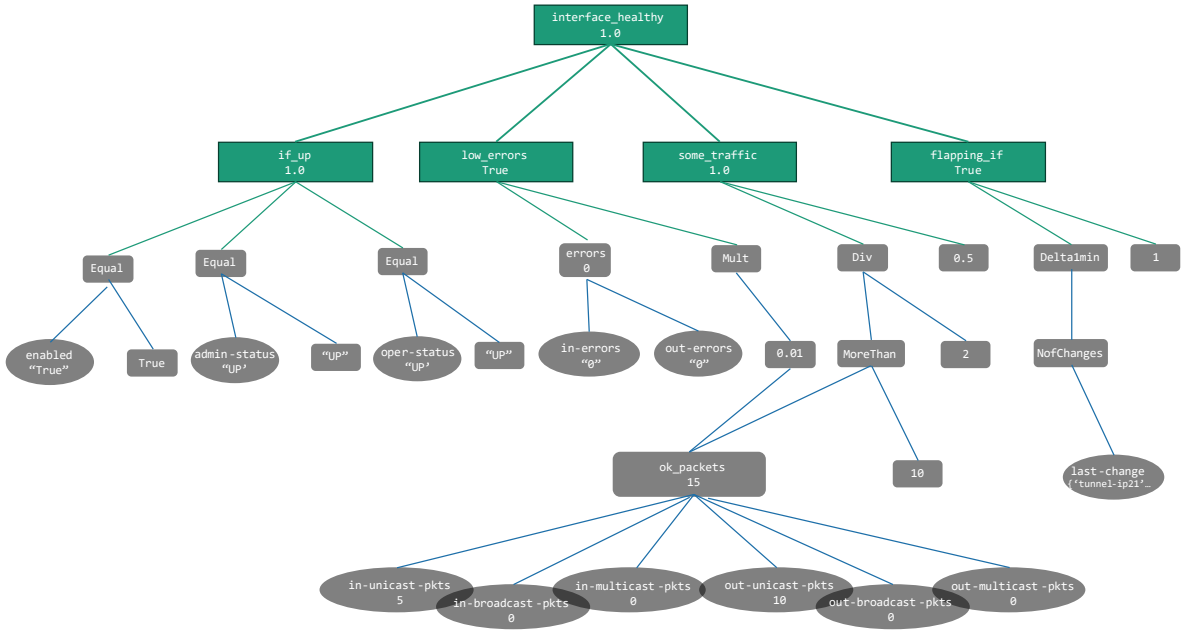

Figure 18 Graphical representation of the Service Component Assurance Tree

Branches of the Assurance tree and their logic are discussed below:

Interface is considered "UP" if both administrative enabled state = True and operational state = True

Interface errors are considered low if both in-errors and out-errors are 0.

… and so on

4.2 Architecture of the proposed solution

In the proposed architecture of the solution, as depicted in Figure 19 architecture consists of Assurance Orchestrator and Assurance Agents in addition to standard network configuration components – Configuration orchestrator such as Cisco NSO, ONAP etc and device being configured.

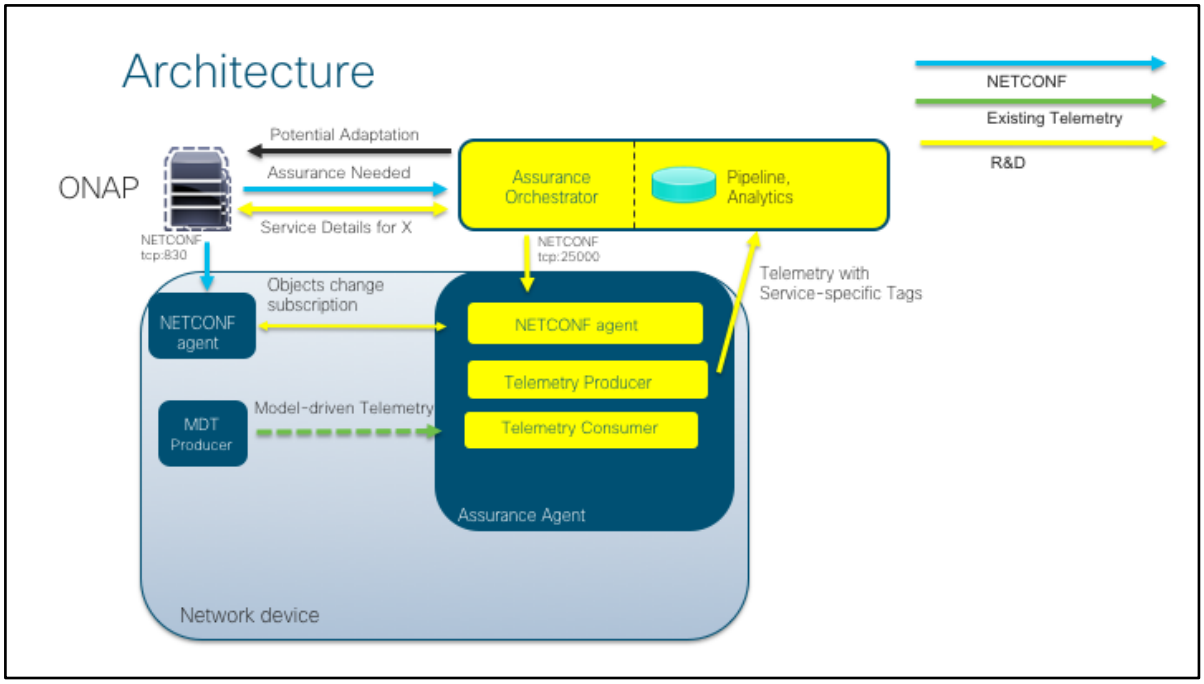

Figure 19 Architecture of the proposed assurance solution with a single assurance agent

In the standard configuration flow, orchestrator is communicating to the device and sending the configuration using NETCONF over TCP protocol, port 830 by default. Any kind of telemetry would be configured as well and exported to the analytics platform. However, with the mentioned standard configuration flow correlation between telemetry data and service configuration is almost non-existent because configuration models are different from operational models which are exported in the telemetry.

To address this gap between configuration and operational models it's necessary to introduce additional components which will introduce tags in data exported via telemetry, so that values send via telemetry can be correlated back to the configuration. By introducing tags in the telemetry, it will be possible to correlate data in the telemetry with configuration model and therefore get clear insight into status of the provisioned network service.

4.3 Functional architecture of the Assurance agent

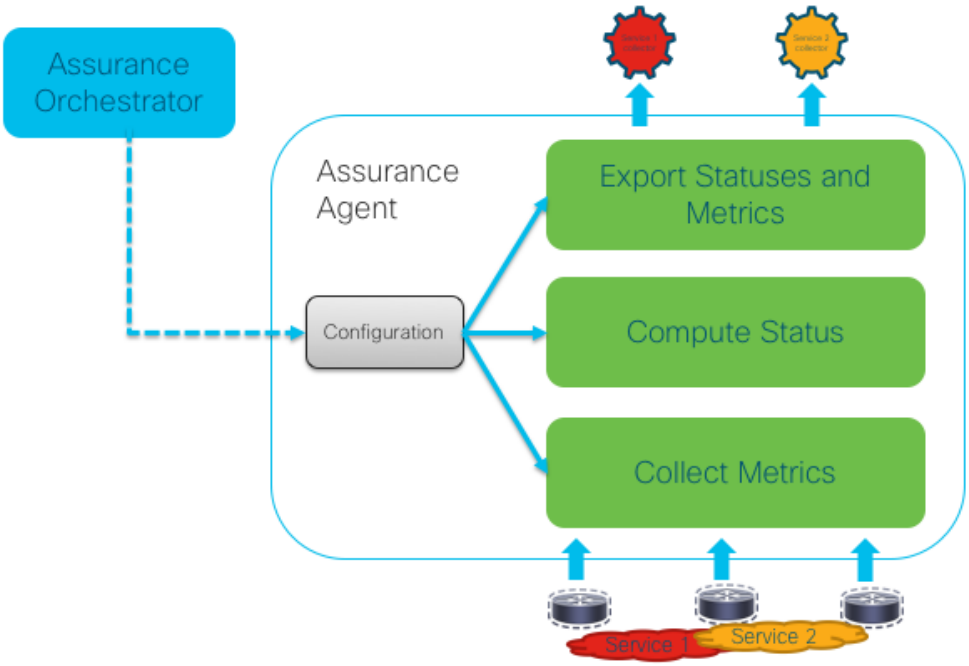

Figure 20 Proposed Architecture of the Assurance Agent

As described on Figure 20 proposed architecture of the assurance agent, we could observe that agent receives configuration from the orchestrator. In turn, configuration is used to perform following actions:

- Collect metrics from the underlying network devices which have been configured to export data: telemetry, SNMP or other data sources
- Compute status of the service based on the received data in order to determine whether service is healthy or degraded
- Finally, Agent would export status and metrices to the service collectors using enriched telemetry and tags which would facilitate correlation of model driven telemetry with the network service configuration

4.4 Simplified graphical representation of the network service path through the network infrastructure

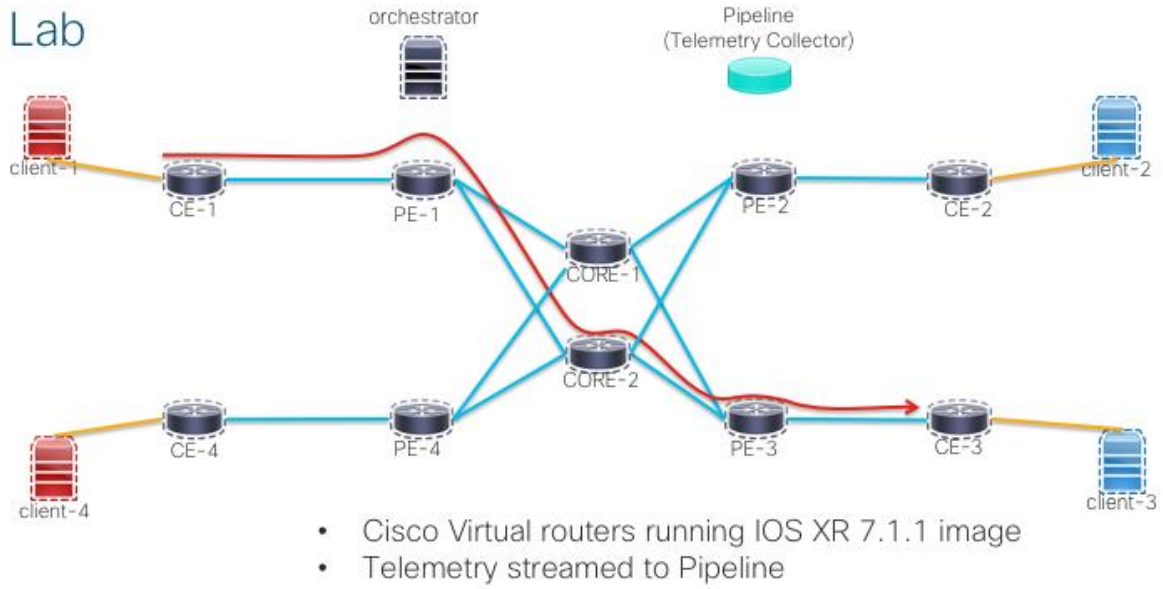

Figure 21 Diagram of the configured Network service in the lab environment

As depicted on the Figure 21 diagram of the configured Network service in the lab environment, Network service is a tunnel between customer's router CE-1 and CE-3. Path between two mentioned customer's routers traverses over the provider's network, forwarded by provider edge (PE) and core routers. In the shown example we could observe that tunnel traverses PE-1, CORE-2 and PE-3 on the data path between the customer edge devices.

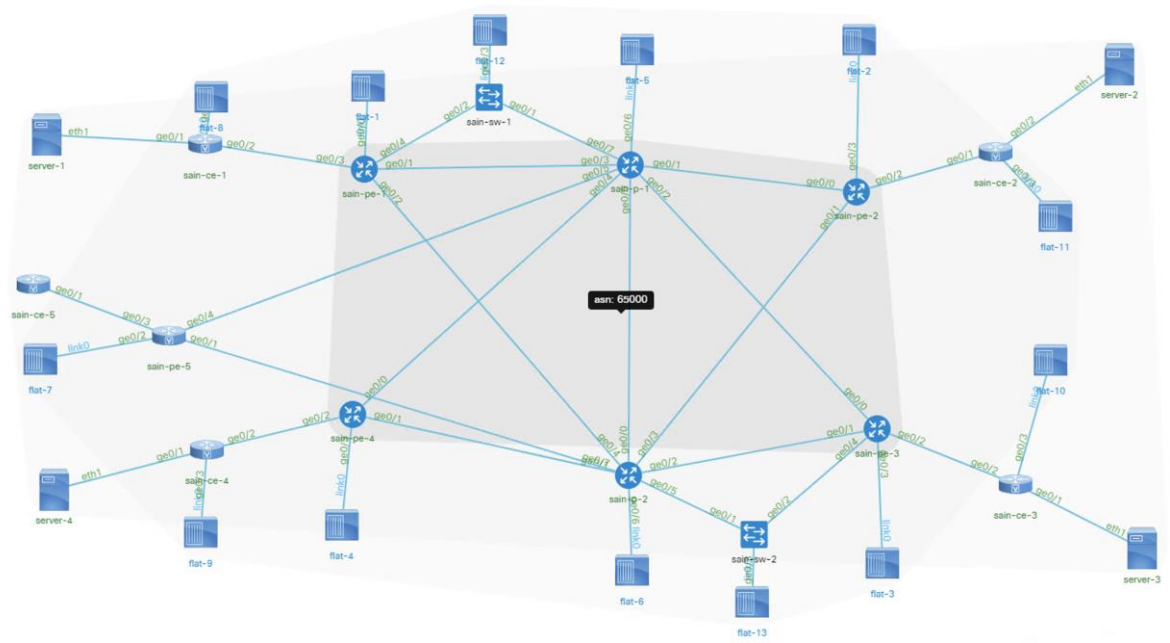

Figure 22 Topology Diagram as drawn by Cisco Virtual Internet Routing Lab software– VIRL [47]

Figure 22 depicts network device topology, connectivity diagram as modelled by Cisco's Virtual Internet Routing Lab software – VIRL [47]. BGP Autonomous system used is AS:65000, while we have various devices in the topology:

- P routers Cisco IOS XRv 9000 running Cisco IOS XR 7.1.1 software
- PE routers Cisco IOS XRv 9000 running Cisco IOS XR 7.1.1 and IOS XR 6.5.1 software
- CE routers Cisco CSR1000v
- Servers runing Linux Ubuntu 14.04 LTS

4.5 Experimental methodology

One of the main hypotheses of the research is to confirm whether amount of the telemetry data could be reduced by leveraging service awareness. In the Figure 23 lab topology along with the deployed assurance agents have been shown.

4.6 Lab Topology used in the experiment.

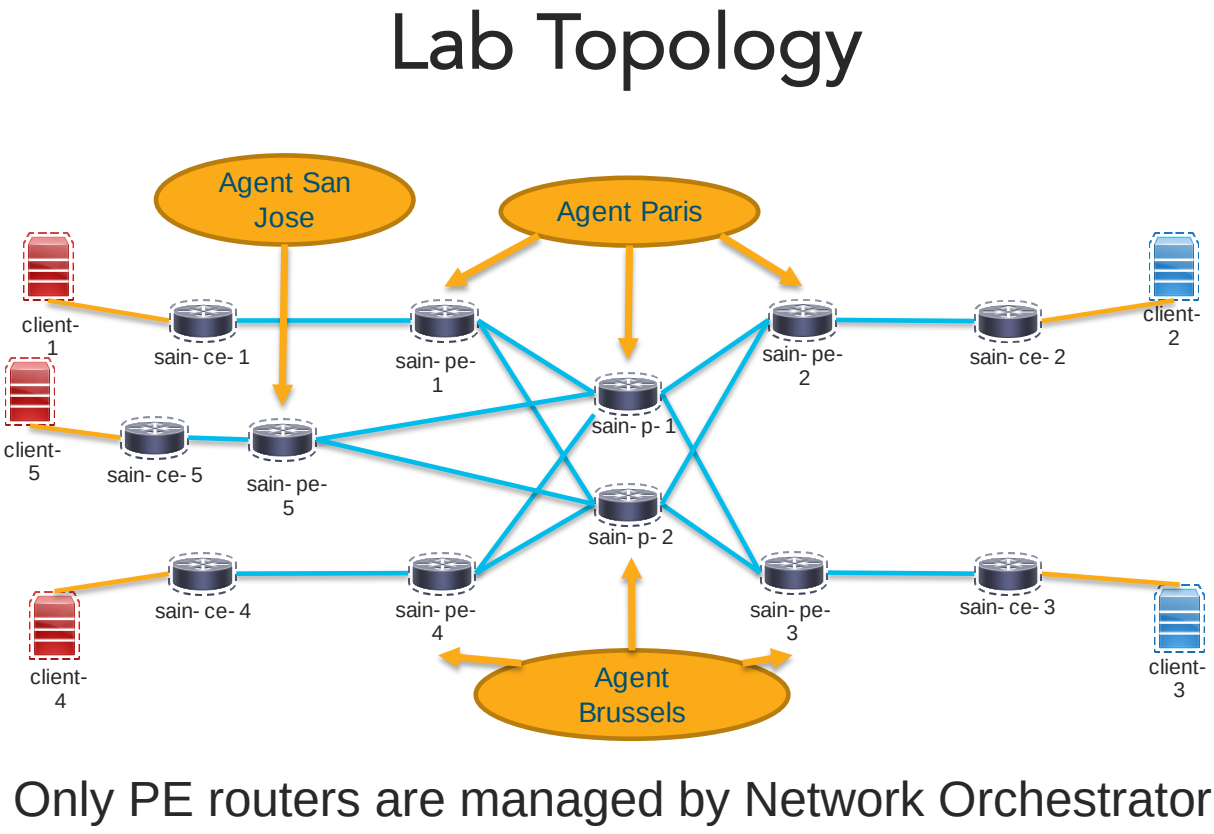

Figure 23 Lab topology depicting Assurance Agents

Results have been produced and collected using the topology described in the Figure 23. It's confirmed that indeed there could be significant reduction of the telemetry data if the suggested solution architecture would be deployed. Data reduction is so significant since essentially agent exports only service specific information and filters out all the bulk data which is used to compute network service status. Such a computation of is of course enabled by the recent developments in the network devices which are not only routing and switching traffic, but also have sufficient CPU capacity to perform complex computations.

type	device	value	percentage
CLI	p-1	69.5 MB	1.26%
CLI	pe-1	174.4 MB	3.17%
CLI	pe-3	69.6 MB	1.26%
MDT	p-1	447.2 MB	8.12%
MDT	pe-1	728.2 MB	13.22%
MDT	pe-3	732.8 MB	13.3%
SNMP	pe-3	38.0 kB	0.0%
SNMP	p-1	25.3 kB	0.0%
SNMP	pe-1	50.7 kB	0.0%
MDT	Data centre 1 (Diegem)	3.0 GB	55.87%
MDT	Data centre 2 (San Jose)	210.1 MB	3.81%

Table 2 Incoming data repartition

Results shown in Table 2 provide information regarding the effective amount of data received from each of the devices in the topology which were involved in the service configuration or make part of the data path for the network service in question.

Experimental setup and results achievements

This experimental setup consists of customer premises routers (CE) as well as provider core routers (P) and provider edge routers (PE) running in a physical and virtualised environment. These devices, whether physical or virtual handle real traffic.

As shown in Figure 24, the network with service models is configured using the orchestration network architecture.

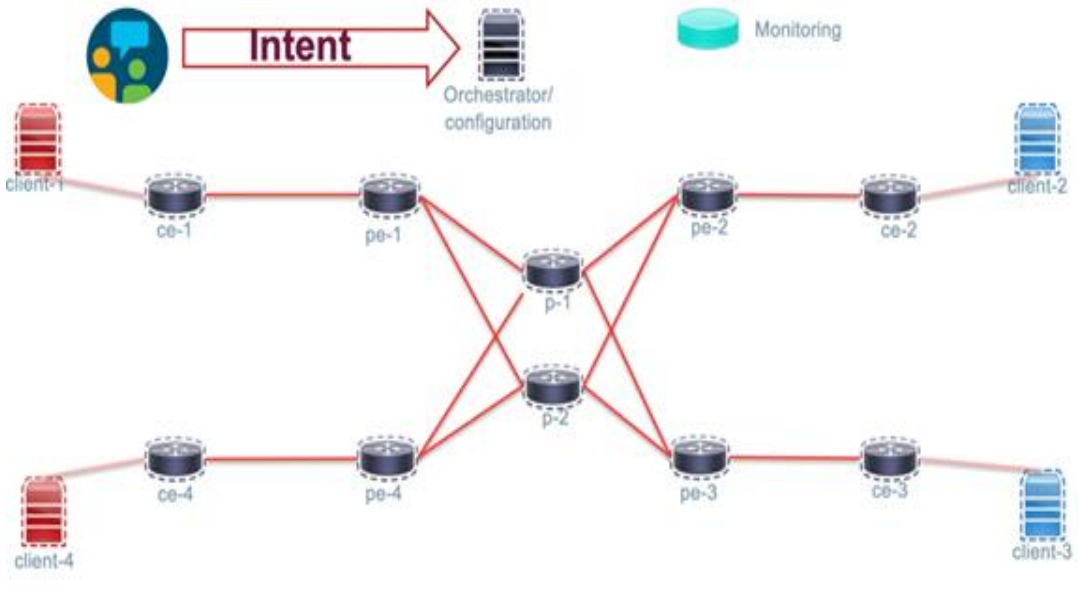

Figure 24 Business Intent communicated to the configuration orchestrator.

In Figure 25 orchestrator is configuring devices in order to fulfil desired service intent. Orchestrator uses Netconf protocol to access and configure network elements which are taking part in the data path to enable desired service. We could see example of the network service configuration in XML format depicted in Figure 12 and Figure 13.

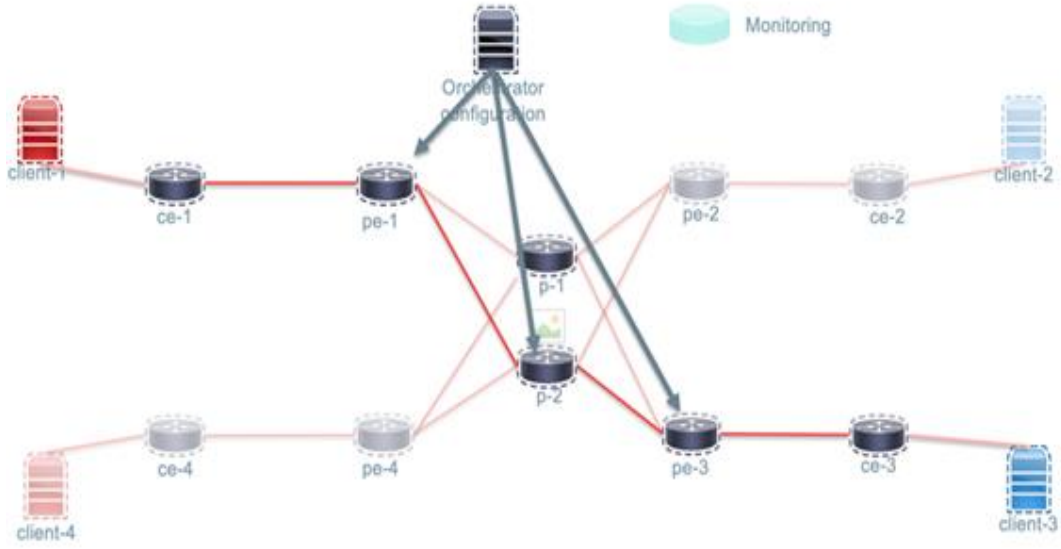

Figure 25 Orchestrator sends configuration to network devices

In the provided example, actual intent is to establish communication – tunnel service between ce-1 and ce-3 network device in order to enable communication between Client-1 and Client-3. In order to traverse path between Client-1 and Client-3, data packets need to cross pe-1, p-2 and pe-3 as shortest path between the endpoints. Of course, this trajectory may be different in function of routing protocols and connectivity in function of time, but topology discovery and update events will be discussed in future work. At this time, we're focusing on fixed path through the experimental network and assuming there won't be topology changes throughout the experiment shown in Figure 26.

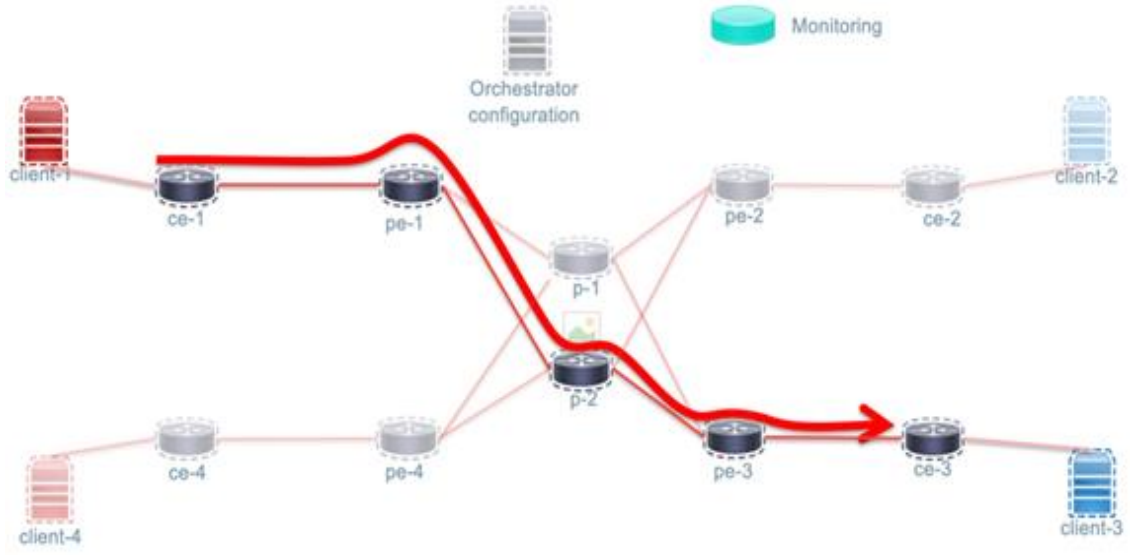

Figure 26 Tunnel service is configured.

However, there are important questions to be answered:

Question: Service running within acceptable KPIs?

Question: Is configuration model mapped to monitoring model?

In Figure 27 we can observe each of the network devices streaming telemetry data to the collector, monitoring platform which is receiving and processing all telemetry data.

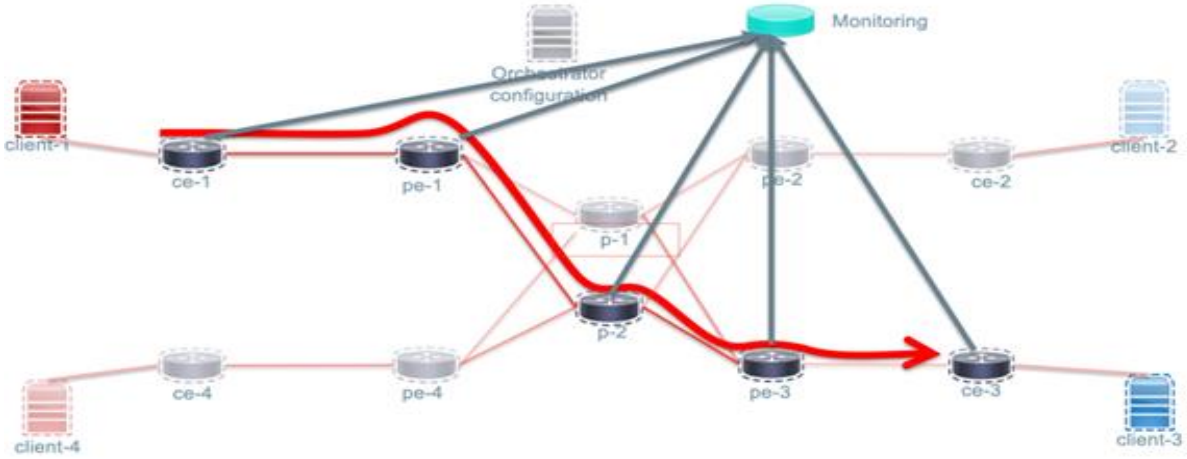

Figure 27 Telemetry data streamed to Monitoring/Analytics platform. 250000 different stats per router (740 kbps of data)

Thanks to the fact involved network elements are already using Model Driven Telemetry processing data points by collector is simpler. However, as there are so many different data points which are being monitored on devices, there may be information overload since on average router there could easily be 250000 different monitored data points. Such as large number of collected data points could essentially mean that amount of generated telemetry data may be significantly high and could pose challenge for network infrastructure as well as could cause impact to collector processing capacity.

Instead of monitoring all relevant and non-relevant data points, causing unnecessary increase of traffic and compute resources to process large amount of data, we're proposing significant reduction in amount of telemetry data by ensuring that only minimal set of relevant data points is exported from the network devices by means of intent-aware assurance agent. Data reduction task is accomplished by deploying assurance agent locally to the network devices, thus leveraging local area network (LAN) links and avoiding use of wide-are links (WAN) for large amount of data points. assurance agent is aware of the service details and is also capable of receiving telemetry data. As represented on assurance agent architecture in Figure 20**Error! Reference source not found.**, service intent is received by from the orchestrator while MDT is received from network devices.

In the block diagram Figure 20 Proposed Architecture of the Assurance Agent is performing analysis on the received datasets and series of computations in order to determine actual state of the service. Steps performed by assurance agent: collecting MDT, processing and exporting reduced – yet more relevant MDT is called assurance pipeline. Final result of assurance pipeline is significantly reduced amount of MDT containing only high-level status of the monitored service, as per pre-defined Key-Performance Indicators (KPIs).

Measuring objective was to determine how much data is actually received via MDT under usual telemetry export, with typical data points for router such as environmental, interface stats etc. Result of this work outlines amount of measured data after performing analysis of the incoming

telemetry and mapping to service aware MDT. All routers and all incoming data points were taken into account.

	Intent-Aware Monitoring Efficiency			
	Total MB	**Average network traffic rate 1 min in kbps**	**Average Network traffic rate 5 min in kbps**	**Average network traffic rate over 15 min in kbps**
Incoming Telemetry from routers	5200	740.7	700.1	711.7
Telemetry generated by Intent Aware Assurance Agents	130.8	17.3	17.2	17.1
Combined Telemetry to Analytics platform	224.9	29.5	29.1	29.3
This work efficiency ratio	40.9	42.8	40.6	41.7

Table 3 Experimental results

Results shown on Table 3 outline achieved efficiency in regard to the raw telemetry data received from the routers vs reduced telemetry data focused only on the network service.
As outlined in Table 3, demonstrated experimental results have reduced the amount of incoming MDT from routers from 5.2 GB to 130 MB, while preserving relevant information which is – is service running and operational per pre-defined KPIs. Network traffic is referring to the data being transferred over the tunnel, configured as part of the experiment.

Configuration is simple, monitoring & assurance is complex. By means of assurance agent, we have accomplished reduction of telemetry data exported over WAN, while enriching MDT with the most relevant data – Service status in relation to the KPIs. Therefore, the amount of Telemetry data has been reduced by injecting service aware information in MDT and removing all overhead MDT data points which do not need to be exposed to the network operator who is monitoring the service. Of course, full MDT can also be enabled if desired.

The PhD candidate has prepared conference paper where he has put all these achievements.

4.7 Abstraction of device configuration model

Why an “abstract device configuration model” ?

In the context of the service assurance, device configuration is retrieved from a network orchestrator (NSO in our example). The device configuration is contained in various YANG models, hence various namespaces. From this configuration, it’s necessary to generate requirements by applying a set of rules as depicted on Figure 28.

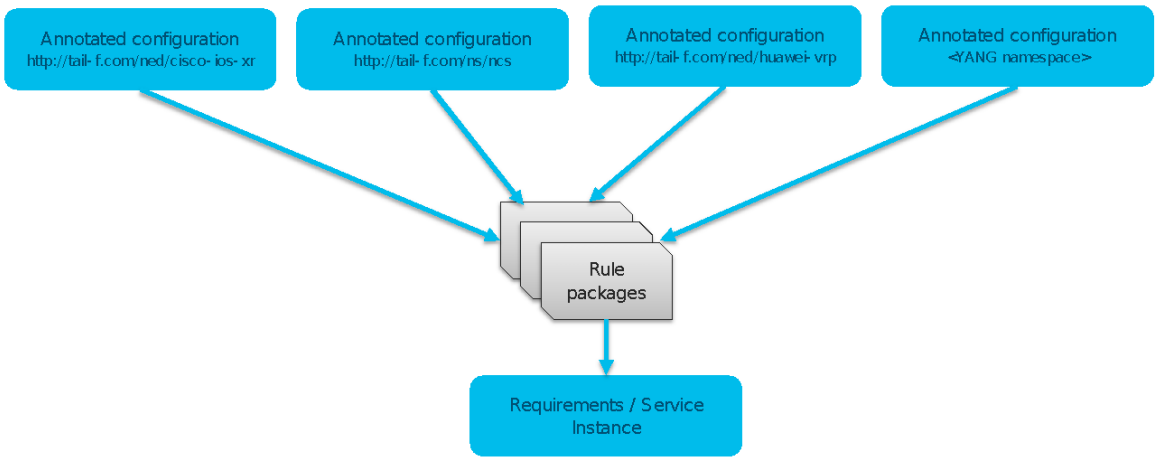

Figure 28 Generating requirements from rules.

The root YANG model is the one of NSO, and it is augmented by the YANG model of the services and of the device’s configuration. Currently, there is a single YANG model for the configuration of an XR device (http://tail-f.com/ned/cisco-ios-xr), but NSO could theoretically support NETCONF devices and thus have an arbitrary set of models for the configuration of a single device.

4.8 Current implementation

The current orchestrator implementation directly generates the requirements from a mixture of the NSO YANG model and the Cisco NED YANG model. The situation is similar to the one depicted in Figure 29, where for each device configuration model, we do have a different set of rules.

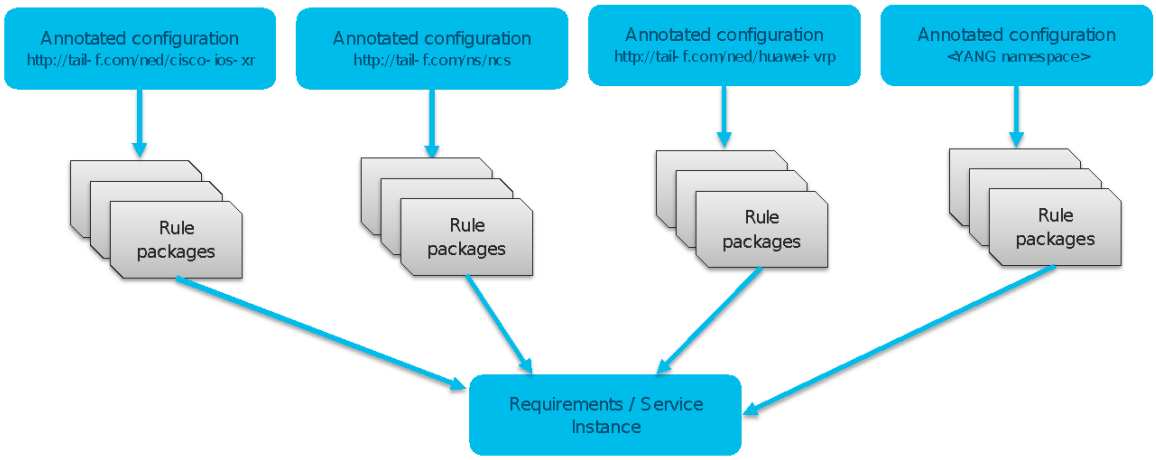

Figure 29 Direct configuration-to-requirement rules

In this context, a rule contains:

- A trigger, which is an XPath query. If an element matching that query is found among the elements configured for a specific service, we apply the rule and use that trigger as base node for subsequent XPath queries. Example: http://tail-f.com/ned/cisco-ios-xr:tunnel-ip is present among the elements configured for the instance *tunnel1* of service *sain-tunnel* -> we apply the rule.
- A set of requirements patterns whose values are filled by XPath queries. Note that these XPath queries may refer to elements in other namespaces. Example:
  DeviceHealthy(<result of query ../../<NSO namespace>:name.text()>)
- Some unformalized mapping code. Example: for each tunnel-ip directive, we define an endpoint in the overlay: Endpoint(<device name>, <./tunnel/vrf/name.text()>, <./tunnel/ip/version.text()>, <./tunnel/ip/address.text()>) and if we have only two endpoints, then we add the requirement Layer3Connectivity(endpoint1, endpoint2).

To summarize, the rules are directly querying the XML containing the global config of NSO and its managed devices.

The drawbacks of this are:

- Need to adapt or redefine rules for each configuration output from the orchestrator

- If several models are used in conjunction, we need a rule for each possible conjunction of the models (i.e., cisco-ned/ncs, huawei-ned/ncs, cisco-ned/onap, huawei-ned/onap, pretty sure the last two have no meaning, but they could potentially exist)
- We might reimplement the same query (i.e., get the IP of the source interface of the tunnel) various time

Proposition: use an abstract model to represent the configuration

The goal of this abstract model is to represent the device configuration (and perhaps layer-2 / layer-3 topology) information that are required for the rule engine to build requirements.

The metamodel for representing such models can be built incrementally, as we are currently building rules. In order to support various configuration formats, we need to be able to map them to this metamodel, as shown in Figure 30.

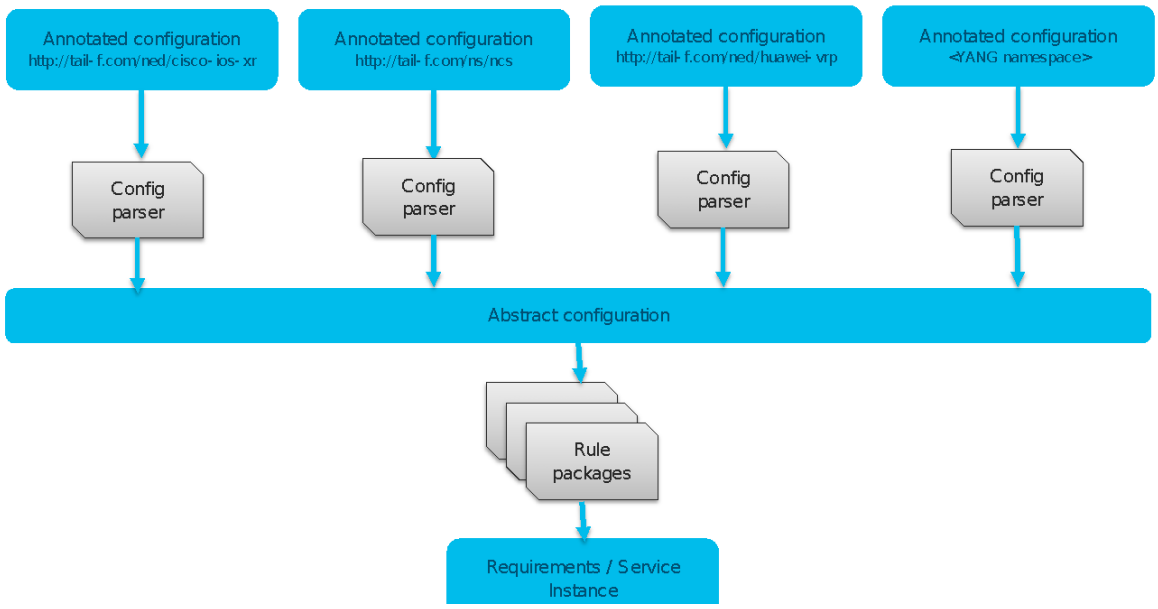

Figure 30 Using an abstract configuration model.

Using this approach, we could parse the whole configuration of each device to generate the model. (+ LLDP to have the topology). We depict in Figure 31 possible instance fragment of such a model. It contains two devices and four interfaces.

Then the example rule from above would become:

- Trigger: presence of a TunnelInterface in the elements configured for a given service instance. Let's call it tunnel-if

- The list of requirements for the rule will be the same, but instead of filling them with XPath Query, we can use the model: DeviceHealthy(tunnel-if.owning_device)
- The same heuristic for connectivity between tunnel endpoints can apply (i.e., we match the 2 endpoints defined in the same service instance, which is not shown on Figure 30). In that case, the connectivity in the overlay can be expressed as:

Layer3Connectivity(tunnel-124_r1.ipv4, tunnel-124_r4.ipv4)

Note that we can also build the underlay connectivity independently for each tunnel interface with Layer3Connectivity(tunnel-if.destination, tunnel-if.source.ip)

After normalization, these two requirements (the one with tunnel-124_r1 and the one with tunnel-124_r2) should be identified as the same requirement.

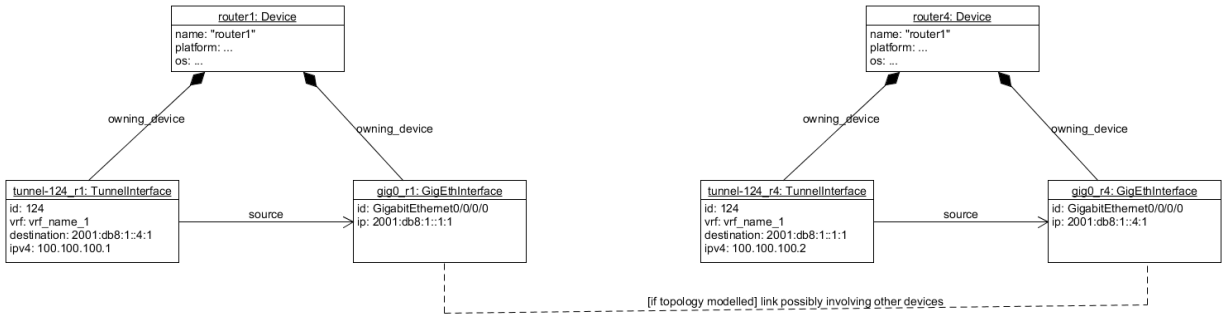

Figure 31 Possible fragment of an instance of the configuration model.

Potential advantages of this approach:

- Simplified writing of the rules (maybe even a GUI for writing rules/requirements).
- Possible mapping between YANG models, i.e., possible workflow for building the model:
    - List devices from NSO
    - Get-config from each device via Netconf and integrate the data from the various YANG models to the abstract configuration model.
    - [Get-oper LLDP from each device via Netconf -> build topology]

    - Get-config from NSO -> list backpointers and identify corresponding elements in the model built.

- Requirement are defined over elements of the model, not strings.
- Simplified checking of the configuration (i.e., check that the source of a tunnel interface always has an IP) as global constraints

Main challenges: sufficiently generic approach to encode the whole configuration.

4.9 Heuristic packages

Heuristic packages are used to encode human knowledge within the rule files with the objective to:

- Automate building and monitoring of assurance cases for network services
- Enabling exchange and reuse of assurance case between experts and operators

By naming *heuristic* objective is to cover a large part of most common issues. However, in some cases:

- A service reported as “Broken” might actually be functional
- A service reported as “Healthy” might actually be non-operational

In such cases, heuristics packages can be extended to improve coverage and accuracy.

As we depicted on Figure 32 heuristic packages could be decomposed in 3 hierarchical layers, higher ones depending on the lower ones.

- Rules
- Service Components
- Metric Engine

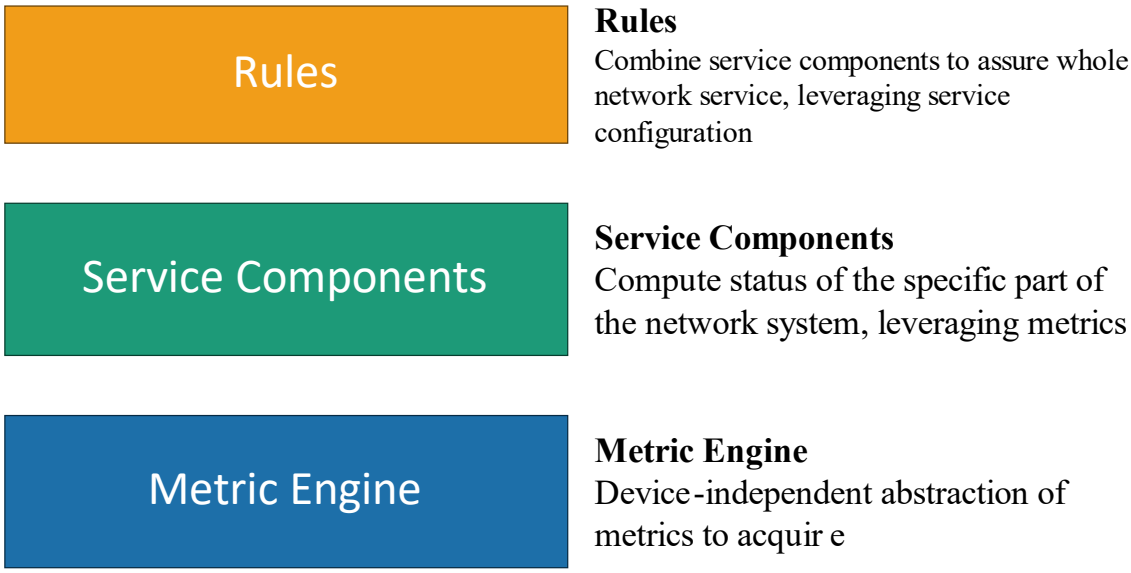

Figure 32 Heuristics packages decomposition.

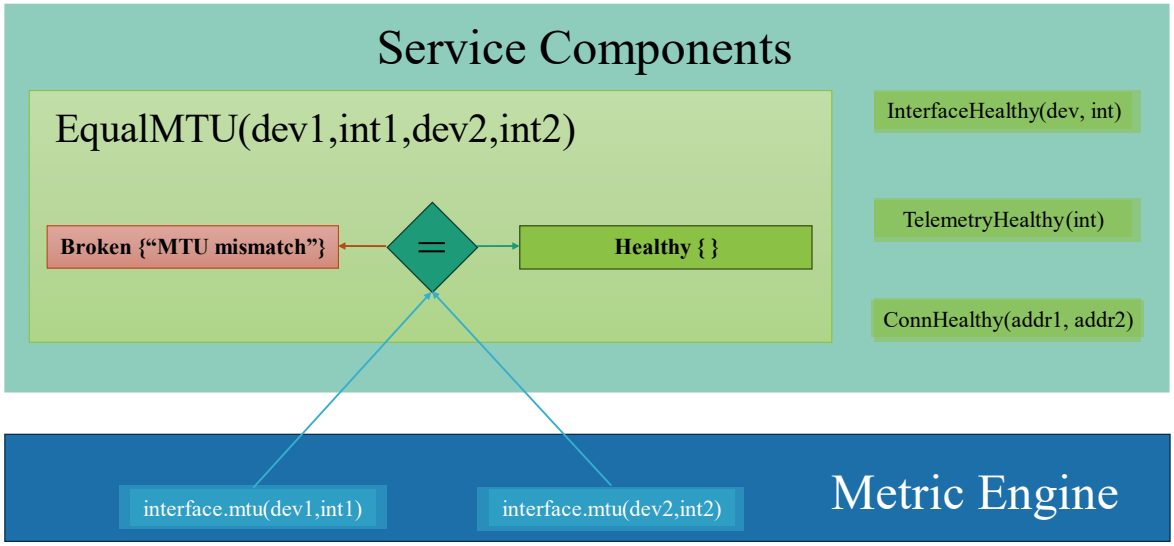

Figure 33 Service components

Figure 33 provides visualisation of the service components, which are expected to focus on a very specific and well-scoped part of the networking system. Service components are report status: score (0/broken to 1/healthy) + symptoms. Service components are reusing metrics from the metrics engine.

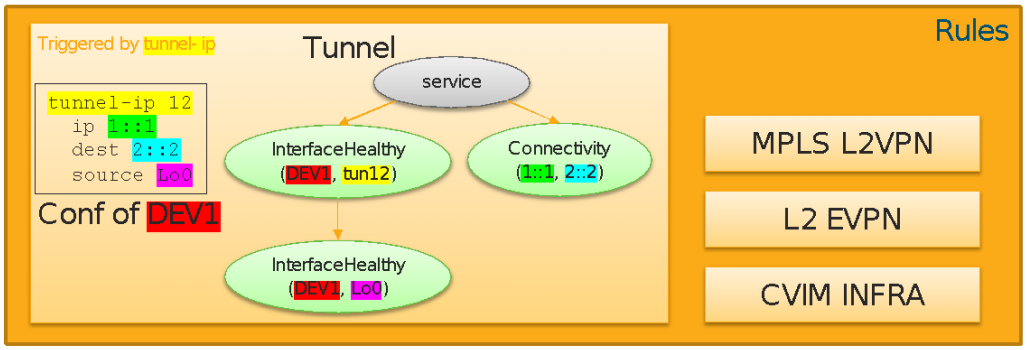

Figure 34 Rules

Figure 35 depicts rules which are expected to parse configuration pushed by NSO to enable a service to produce an assurance graph. Assurance graph: service components with parameters from the configuration and dependencies. Dependencies aggregate symptoms to explain services malfunction.

4.10 Service Language

Domain-Specific Language (DSL), service language was developed to describe the computation needed to evaluate the status of a service components. Detailed description of the service language has been provided in Appendixes

Appendix A - Service Language

4.11 Network service and Intent connection

The connection between network GRE (Generic Routing Encapsulation) tunnel service configuration and network intent lies in the way network intent can be achieved and managed through the use of GRE tunnels. We're indicating some of the possible connections:

- Intent-based routing: By using GRE tunneling, we could define specific intent-based routing policies. For example, we can create tunnels that prioritize certain types of traffic, such as video conferencing or VoIP, to ensure a smoother user experience.

- Security and isolation: GRE tunnels can be configured to create isolated network segments, providing secure connectivity between different parts of a network. Network intent can be defined to enforce security policies and ensure that traffic from one segment is not accessible to another.

- Traffic engineering: GRE tunnels can be used to optimize network traffic by creating virtual links between different locations. With network intent, we could define policies to dynamically adjust the traffic flow through these tunnels based on real-time conditions, such as bandwidth availability or network congestion.

- Multi-tenancy and service chaining: GRE tunnels can enable multi-tenancy in a network environment, where different customers or tenants can have their own isolated virtual networks. Network intent can be used to define policies for service chaining, where specific services or applications are automatically routed through different tunnels based on predefined rules.

- Cloud connectivity: GRE tunnels can be used to establish secure connections between on-premises networks and cloud environments. Network intent can define policies to ensure optimized and secure connectivity, such as routing traffic through specific tunnels to preferred cloud service providers.

Overall, the configuration of GRE tunnel services, as we also demonstrated in our work, can be aligned with network intent to achieve specific goals, such as improved performance, security, isolation, and efficient management of network resources.

4.12 Machine learning techniques

In order to achieve improve prediction and better efficiency using machine learning it's imperative to provide sufficient datasets to train the machine learning models. [48]To answer the question on to what would be considered as sufficient training data in it's necessary to

perform further analysis of the business intent, network configuration, amount of generated telemetry data and service aware telemetry data. In either case, advancements in prediction with machine learning algorithms significantly depends on the quality of the datasets. Since it's essentially difficult to obtain high-quality training datasets [48] telemetry data discussed in this article could be used as a base for such a dataset repository.

With the data acquired using service aware telemetry it should be possible to facilitate machine learning objectives. In Table 1 there is short overview of the Machine learning:

Machine Learning Objectives

Supervised	Unsupervised
Discrete Data Classification (predict a label)	Clustering (group similar items)
Continuous Data Regression (predict a quantity)	Dimensionality Reduction (reduce number of variables)

Table 4 Machine Learning Objectives

Application#1a: Discover Service-Impacting Objects

- Discover new correlation or non-correlation
- Goal is to improve the set of service-impacting objects (and hence the agent):

- Discover other objects that “impact ” the service KPIs

=> Add those service-impacting objects to telemetry

- Discover “ service-delivery ” objects that do not influence the service

=> Remove those service-delivery objects from telemetry

- Hope: by providing the known service-related objects to ML, we help the search

Since there is service tag (label) available in the telemetry it could be possible to discover new relationships and dependencies in the acquired data sets. Such machine learning application could in turn predict service impacting behaviours and trigger corrective actions before service degradation below acceptable KPIs.

Machine Learning (ML) and Ontology Based systems are important building block in conjunction with model telemetry data and network element sources to perform service decomposition in order to identify anomalies and outliers, perform root-cause analysis and possibly conclude by predicting service impact

Assurance automation may be classified in the following areas.

- **Intelligent process automation:** encoding the daily operations performed by level 1 NOC engineers and incident management teams, empowering the operations with guided automations using software modules and increasing the level of accuracy and degree of automation.
- **Root-cause analysis:** automating and using machine learning to increase the efficacy of root-cause analysis to generate insights and guided actions. Service quality degradation can be predicted to enable proactive service assurance.
- **Closed-loop assurance:** integrating with configuration and orchestration systems to enable network changes to resolve performance and service issues.

5 Closed loop automation for Intent-Based Networking

This chapter focuses on exploring the proposed solution for closed-loop automation in the context of intent-based networking. We will delve into the key concepts, principles, and technologies that underpin closed-loop automation in IBN.

Firstly, we will discuss the importance of closed-loop automation in maintaining network reliability, scalability, and security. We will explore how closed-loop automation enables the network to dynamically adapt to changing conditions and proactively address potential issues before they impact business operations.

5.1 Background

A compulsory step for intent-based networking involves closing a loop with telemetry for service assurance. Discovering whether a service fulfils its service level agreement (SLA) is relatively easy when monitoring synthetic traffic mimicking the service. However, such an over-the-top mechanism only provides SLA compliance results that considers a network on which the service is enabled as a "black box," without knowledge of inner workings or low-level components of the service. Therefore, a network operator tasked with the monitoring of the service has limited or no insights on which specific degraded or faulty network components/features are responsible for service degradation. This issue is particularly difficult when the network is composed of heterogeneous network components. Telemetry exists today to report operational information, but an issue arises in that telemetry from network devices in the network does not provide service context information. Hence, troubleshooting the service based on the telemetry is very complex, with, on one side, the service information, and on another side, network device-specific telemetry information. In the event that the network operator discovers that a service is underperforming, e.g., is not fulfilling its SLA, it may be near impossible for the network operator to identify in an efficient manner which low-level components of the service are responsible for such underperformance. The inability to identify efficiently the problematic low-level components hampers efforts to make repairs at the component-level in order to restore acceptable performance to the service.

5.2 Overview

Aspects of the invention are set out in the independent claims and preferred features are set out in the dependent claims. Features of one aspect may be applied to any aspect alone or in combination with other aspects.

A method is performed at one or more entities configured to configure and provide assurance for a service enabled on a network. The service is configured as a collection of service components on network devices of the network. A definition of the service is decomposed into a Service dependency graph that indicates the service components and dependencies between the service components that collectively implement the service. Based on the Service dependency graph, the service components are configured to record and report Service metrics indicative of Service health states of the service components. The Service metrics are obtained from the service components, and the Service health states of the service components are determined based on the Service metrics. A health state of the service is determined based on the Service health states. One or more of the service components are reconfigured based on the health state of the service.

Corresponding systems and apparatus for implementing the methods described herein area also provided.

5.3 Service Assurance for Intent-Based Networking

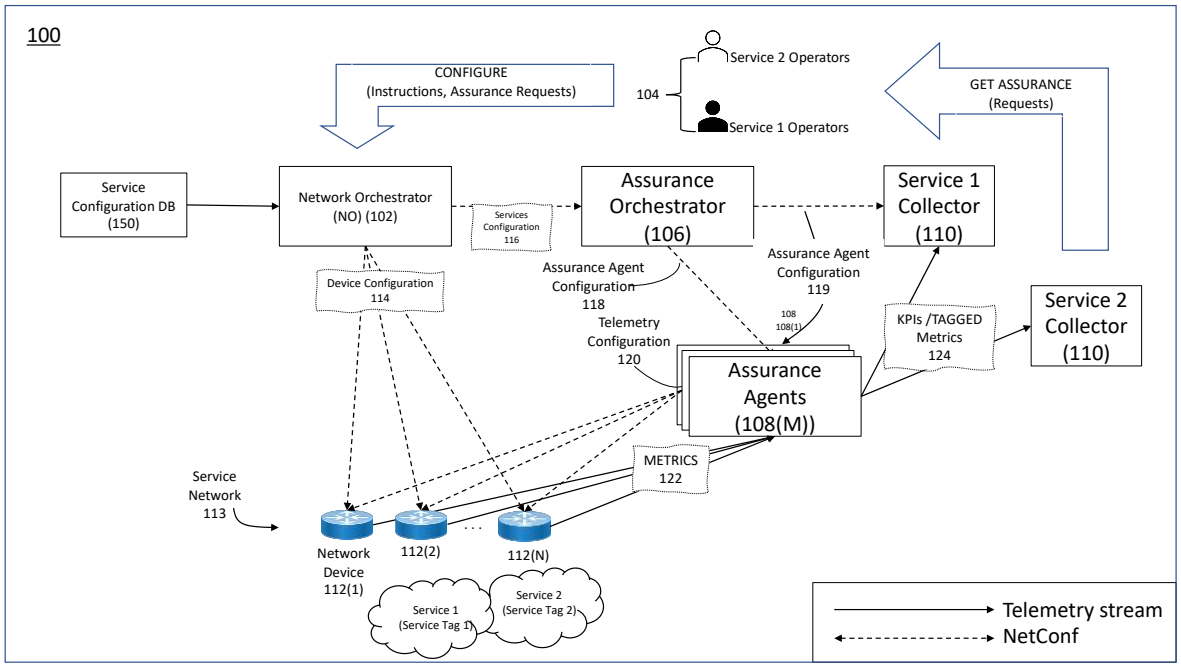

Figure 35 Block diagram of network service assurance architecture

With reference to Figure 35, there is a block diagram of an example network service assurance system or architecture (also referred to as a "service assurance system"). Service assurance system may provide service assurance for intent-based networking, for example. The service assurance system leverages programming capabilities of network devices in the intent-based network (also referred to as a "service network" or simply a "network"), and model/event driven metrics in telemetry obtained from the network devices, to deliver end-to-end service assurance for various services. Service assurance system includes a network orchestrator (NO) 102, service operators to provide instructions to the network orchestrator, an assurance orchestrator that communicates with the network orchestrator, Assurance agents(1)-(M) (collectively, "assurance agents") that communicate with the assurance orchestrator, assurance collectors that communicate with the Assurance agents and the service operators, and network devices(1)-(N) (collectively, "network devices") that communicate with the network orchestrator and the assurance collectors. Network orchestrator configures network devices(1)-(N) to implement an intent-based service network enabled to provide a variety of services to end users. Network devices may include routers, switches, gateways, and other network devices (physical or virtual). Assurance orchestrator, Assurance agents, and assurance

collectors are generally referred to as one or more “assurance entities” (or simply “entities”) configured to provide assurance for services on a network.

Network orchestrator may include applications and/or services hosted on one or more server devices (more simply referred to as servers), for example, in a cloud-based data centre. Assurance orchestrator may also include applications and/or services hosted on one or more server devices, which may be the same as or different from the servers used by network orchestrator. Similarly, assurance collectors may also include applications and/or services hosted on one or more servers, which may be the same as or different from the servers used by assurance orchestrator. In an embodiment, assurance collectors are applications integrated into assurance orchestrator. Assurance agents(1)-(N) may each include applications and/or services hosted on one or more servers and may be distributed geographically to be near respective ones of network devices(1)-(N) enabled for services to be monitored under control of the assurance agents. Network orchestrator, assurance orchestrator, Assurance agents, assurance collectors, and network devices may communicate with each other over one or more communication networks, including one or more wide area networks (WANs), such as the Internet, and one or more local area networks (LANs).

In the example of Figure 35, service assurance system supports multiple services, including service 1 and service 2 (collectively, “the services”). To this end, service operators include a service 1 operator for service 1 and a service 2 operator for service 2, and assurance collectors include a service 1 collector for service 1 and a service 2 collector for service 2. Service operators (e.g., service 1 operator and service 2 operator) provide to network orchestrator network and service intent-based instructions to setup/configure the services (e.g., service 1 and service 2) for end users. Service operators also receive requests for assurance (e.g., “get assurance” requests) for the services from assurance collectors (e.g., service 1 collector and service 2 collector), and forward the requests to network orchestrator.

5.4 Network Orchestrator

Responsive to the aforementioned instructions and the requests sent by service operators, network orchestrator derives and sends to network devices intent-based network device configuration information to configure the network devices/service network for the services (e.g., for service 1 and service 2). In addition, network orchestrator derives and sends to assurance orchestrator service configuration information for providing assurance for the

services (e.g., service 1 and service 2) enabled on service network. Service configuration information includes, for each service deployed or implemented on service network, respectively, a definition of the service, including a service type (e.g., a type of network connectivity), a service instance (e.g., an identifier or name of the service), and configuration information that describes how the service is actually implemented of service network. That is, the definition of the configuration of the service is reflective of how the service is instantiated as a collection of the service components in service network.

For network device configuration information, network orchestrator may employ, for example, the Network Configuration Protocol (NETCONF) (or, similarly, Representational State Transfer (REST) Configuration (RESTCONF)) to push intent-based network device configuration objects, such as Yet Another Next Generation (YANG) models or objects, to network devices. Similarly, for services configuration information, network orchestrator may also employ, for example, NETCONF to push intent-based service configuration YANG objects to assurance orchestrator. YANG is a data modelling language used to define data sent over a NETCONF compliant network to configure resources. NETCONF are used to install, manipulate, and delete configurations of the resources, while YANG is used to model both configuration and state data of the resources. YANG models/objects used to implement embodiments presented herein may include YANG models/objects extended to include service-specific metadata annotations in accordance with RFC 7952 [49], for example, or any other format that may be the subject of a future standard.

Network orchestrator configures a wide range of different service components on one or more of network devices to enable/support each of the services on service network.

To do this, network orchestrator (i) generates Service configuration information that includes network device configuration commands/instructions and associated configuration parameters for the service components to be configured, and (ii) pushes the Service configuration information to network devices in network device configuration information, as mentioned above. Network orchestrator also provides the Service configuration information to assurance orchestrator in service configuration information, as mentioned above.

Network orchestrator stores in a service configuration database (DB) a definition of each of the services that the network service orchestrator configures on service network. In an example, service configuration database may be hosted on network orchestrator.

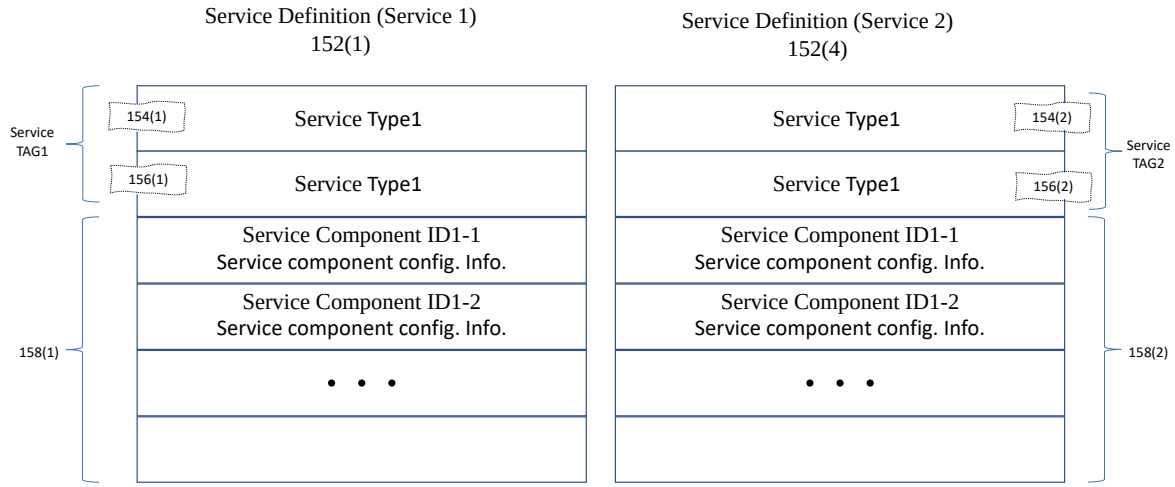

Figure 36 Example of service configuration database

With reference to Figure 36 Example of service configuration database, there is an illustration of an example of service configuration database. In the example of Figure 36 Example of service configuration database, service configuration database stores definitions (1) and (2) (also referred to as "service definitions") for service 1 and service 2, from Figure 35. Each definition (i) may be similar to or the same as the definition of a service described above. Each definition (i) may include a service type (i) and a service instance (i) for the service to which the service definition pertains, and configuration information 158(i) that describes how that service is actually implemented/configured on service network. More specifically, configuration information 158(i) for a given service includes, for each of the service components of the given service, a respective Service identifier (ID) (e.g., Service ID1-1, Service ID1-2, and so on), and respective Service configuration information (e.g., specific operations and/or intent-based network device configuration objects used to configure that Service on a network device). Network orchestrator may use a service type, a service instance, and a Service identifier of a Service as indexes into service configuration database 150 to search for and find respective Service configuration information for the service component.

Some sample service component configurations have been provided in the Appendix B – Examples of service component configuration.

5.5 Assurance Orchestrator

Returning to Figure 35, assurance orchestrator operates as a central controller for assurance of the services deployed on service network. That is, assurance orchestrator employs “service awareness” to control assurance for the services deployed on service network. In this role, assurance orchestrator performs several main operations. First, assurance orchestrator generates, from the service type and the service instance in the definition of each service defined in service configuration information, a unique service tag for the service. In an example, the service tag for a given service may be a tuple that includes the service type and the service instance from the definition of the given service. The service tag may be used to distinguish the service to which it pertains from all other services.

Second, assurance orchestrator decomposes the definition of each service defined in service configuration information into a respective Service dependency graph of service components and dependencies/interdependencies between the service components that collectively (actually) implement the service on a network. That is, assurance orchestrator dissects each service into the respective Service dependency graph. The Service dependency graph includes (service component) nodes that represent the service components and links between the nodes that represent the dependencies between the service components. The Service dependency graph may include the service type and the service instance (e.g., the service tag) for the service represented by the Service dependency graph. To assist with the aforementioned decomposition, assurance orchestrator may poll or query various network devices identified in the definition to discover service components, such as packet routing protocols, implemented on the network devices and that are to be incorporated into the Service dependency graph.

In a non-limiting embodiment, the Service dependency graph includes a Service dependency tree having a root node that represents the services, and nodes that represent the service components and that have parent-child relationships (i.e., the dependencies) between the nodes/service components that lead back to the root node. An example of a Service dependency tree is described below. Other types of graph constructs/data structures may be used to represent the Service dependency graph, as would be appreciated by one of ordinary skill in the art having read the present specification.

Third, assurance orchestrator derives from each Service dependency graph a respective set of heuristic packages for the service described by the Service dependency graph. The heuristic

packages (i) specify/define service-related metrics (i.e., Service metrics) to be monitored/recorded and reported by the service components, and that are indicative of health statuses/states of the service components, i.e., that are indicators of health states of the service components, (ii) include rules to determine/compute key performance (KPIs) including the health states of the service components (also referred to individually as a "Service health state," and collectively as "Service health states") based on the Service metrics as recorded and reported, and (iii) which sensor paths (i.e., telemetry paths) are to be enabled for reporting telemetry, i.e., to report the Service metrics recorded by the service components from the service components. The heuristic packages may also include or be associated with the service tag for the service to which the heuristic packages correspond. Assurance orchestrator employs the heuristic packages to configure Assurance agents to monitor the service components of the services, and to compute the health states of the service components based on the monitoring, as described below.

Fourth, assurance orchestrator provides to Assurance agents assurance agent configuration information including the heuristic packages and their corresponding service tags in association with each other. Assurance orchestrator may employ NETCONF to push the heuristic packages as YANG objects to Assurance agents. Assurance orchestrator may also provide the Service dependency graphs to assurance collectors in assurance collector configuration information.

Benefits of assurance orchestrator:

- Centralized management: The assurance orchestrator allows for centralized management and control of multiple assurance agents. It provides a single point of control for monitoring and controlling the network assurance solution.

- Simplified deployment and configuration: The orchestrator can simplify the deployment and configuration process for multiple assurance agents. It ensures consistent configurations across the agents, reducing the chances of errors and misconfigurations.

- Scalability: The orchestrator enables easy scalability by allowing the addition or removal of assurance agents as per the network requirements. It can dynamically allocate resources to different agents based on the network conditions.

- Improved efficiency: With an orchestrator, the coordination between different assurance agents can be optimized, resulting in improved efficiency and faster problem resolution.

- Enhanced visibility and analytics: The orchestrator can collect and consolidate data from multiple assurance agents, providing a holistic view of the network. It enables advanced analytics and reporting, helping in identifying patterns, trends, and potential issues.

Drawbacks of assurance orchestrator:

- Single point of failure: Since the orchestrator is responsible for managing multiple assurance agents, any failure or downtime of the orchestrator can impact the entire network assurance solution.

- Increased complexity: Implementing an orchestrator adds complexity to the network assurance solution. It requires additional setup, configuration, and maintenance efforts.

- Performance overhead: The orchestration process can introduce some performance overhead due to the additional processing required by the orchestrator. This overhead can impact the real-time responsiveness of the assurance agents.

5.6 Assurance Agents

Assurance agents act as intermediary assurance devices between network devices, assurance collectors, and assurance orchestrator. More specifically, Assurance agents translate assurance agent configuration information, including the heuristic packages, to telemetry configuration information, and provide the telemetry configuration information to network devices, to configure the network devices to record and report the Service metrics mentioned above. For example, Assurance agents generate monitoring objects that define the Service metrics to be recorded and reported by the service components and provide the monitoring objects to the service components in telemetry configuration information, to configure the service components to record and report the Service metrics. Assurance agents may maintain associations/bindings or mappings between the heuristic packages, the monitoring objects

generated by the heuristic packages, and the services (e.g., service tags) to which the heuristic packages and the monitoring objects pertain. Assurance agents may employ NETCONF (or RESTCONF), for example, to push YANG monitoring objects to network devices.

In response to receiving the monitoring objects in telemetry configuration information, network devices record the Service metrics specified in the monitoring objects and report the Service metrics (labelled as “metrics” in Figure 35) back to Assurance agents in telemetry streams. In an example, the telemetry streams carry Service metrics in telemetry objects corresponding to the monitoring objects. In turn, Assurance agents tag Service metrics with service tags to indicate which of the Service metrics are associated with/belong to which of the services, to produce service-tagged Service metrics (labelled “tagged metrics” in Figure 35). In other words, Assurance agents apply the service tags to the Service metrics for the services to which the service tags belong. In the example in which Service metrics are carried in telemetry objects, Assurance agents tag the telemetry objects with the service tag to produce service-tagged telemetry objects). Thus, the service tags provide service context to the Service metrics.

In one embodiment, Assurance agents do not perform any specific analysis on the Service metrics, leaving such analysis to assurance collectors and/or assurance orchestrator. In another embodiment, Assurance agents perform analysis on Service metrics as instructed by the heuristic packages, to produce health states of the service components (e.g., KPIs used as indicators of health states of the service components) to which the Service metrics pertain. Assurance agents provide to assurance collectors service-tagged Service metrics, along with health states of the service components when computed by the assurance agents. For example, Assurance agents provide flows of service-tagged Service metrics tagged with service tag 1 to indicate service 1 to service 1 collector, and service-tagged Service metrics tagged with service tag 2 to indicate service 2 to service 2 collector. Assurance agents may also provide service-tagged Service metrics to assurance orchestrator.

Benefits of assurance agent(s):

- Distributed monitoring: Assurance agents can be deployed at various points in the network, allowing for distributed monitoring and ensuring comprehensive coverage. They can monitor specific network elements or segments in real-time.

- Proactive issue detection: Assurance agents continuously monitor the network and can proactively detect and alert about any potential issues or anomalies. This helps in identifying and resolving problems before they impact the network performance.

- Fast and localized problem resolution: As agents are deployed closer to the network elements they monitor, they can quickly identify the root cause of issues and facilitate localized problem resolution. This reduces the troubleshooting time and minimizes the impact on the overall network.

- Resource optimization: Assurance agents can optimize the utilization of network resources by monitoring and analyzing the traffic patterns. They can provide insights into resource usage, helping in optimizing network capacity and improving efficiency.

Drawbacks of assurance agent(s):

- Limited scope: Each assurance agent can monitor only a specific subset of the network. This can result in blind spots where certain network elements or segments might not be monitored.

- Scalability challenges: Adding more assurance agents to cover a larger network can introduce scalability challenges. Coordinating and managing a large number of agents might become complex and resource-intensive.

- Higher deployment and maintenance costs: Each assurance agent requires separate deployment, configuration, and maintenance efforts. Managing multiple agents can increase the overall cost of the network assurance solution.

5.7 Assurance Collectors

Assurance collectors receive/collect service-tagged Service metrics, and health states of the service components when available, from Assurance agents for various services, as uniquely identified by the service tags with which the Service metrics are tagged. Assurance collectors associate service-tagged Service metrics with respective ones of the various services based on the service tags. Assurance collectors determine a respective overall health state of each service based on the health states of the service components of the service, as indicated by the service-

tagged Service metrics and their KPIs/health states. When Assurance agents do not provide to assurance collectors health states of the service components along with service-tagged Service metrics, assurance collectors compute the health states of the service components from the service-tagged Service metrics as instructed by corresponding ones of the heuristic packages (e.g., by the heuristic packages tagged with the same service tag as the Service metrics).

5.8 NETCONF/YANG (Object-Based) Implementation in Assurance System

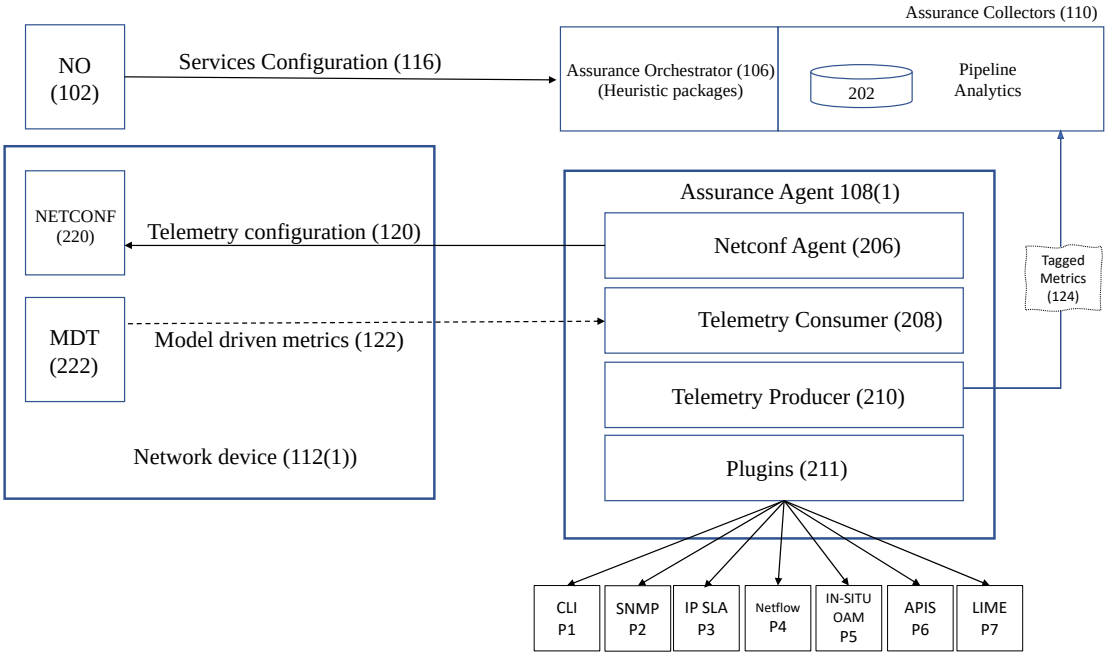

Figure 37 Block diagram of assurance orchestrator

With reference to Figure 37 Block diagram of assurance orchestrator, there is a block diagram that shows additional details of assurance orchestrator, assurance collectors, a representative assurance agent (e.g., assurance agent (1)), and a representative network device (e.g., network device (1)) from Figure 35. Assurance collector includes pipeline analytics to analyse service-tagged Service metrics including the KPIs (if any) from Assurance agents, to determine health states of the service components and then service health states based on the health states of the service components.

Assurance agent (1) includes a NETCONF agent, a telemetry consumer, a telemetry producer, and plugins. Plugins provide various functional capabilities to assurance agent (1) to assist with tasks/operations performed by the assurance agent, including communicating with entities

external to the assurance agent. Examples of plugins include, but are not limited to, one or more of the following: a command line interface (CLI) plugin P1; a Simple Network Management Protocol (SNMP) plugin P2; an IP service- level agreement (SLA) plugin P3; a NetFlow protocol plugin to communicate with NetFlow- enabled network devices P4; an in-situ operations, administration, and maintenance (IOAM) plugin P5 to provide real-time telemetry of individual data packets and flows; application programming interfaces (APIs) P6; and Layer Independent OAM Management in the Multi- Layer Environment (LIME) P7.

NETCONF agent digests heuristic packages sent by assurance orchestrator. NETCONF agent generates monitoring objects (in telemetry configuration information) as network device configuration YANG objects based on the heuristic packages and pushes the monitoring objects to network device (1) to configure the network device for model-driven telemetry (MDT) used to report recorded Service metrics. NETCONF agent may include in the monitoring objects respective identifiers of the service components to which the monitoring objects pertain (e.g., an identifier of network device (1), since the network device is a service component), and the service tag for the service to which the Service pertains. Telemetry consumer 208 receives from network device (1) Service metrics recorded in (model-driven) telemetry objects corresponding to the monitoring objects. The telemetry objects include the Service metrics, the identifier of the Service (e.g., the identifier of network device (1)) to which the Service metrics pertain and may also include the service tag copied from the corresponding monitoring object. Telemetry consumer passes the (received) telemetry objects to telemetry producer. Telemetry producer tags the (received) telemetry objects with service tags, as mentioned above, and sends resulting service-tagged telemetry objects (representing service-tagged Service metrics) to assurance pipeline analytics of assurance collectors, and optionally to assurance orchestrator. Telemetry producer may also copy into the service-tagged telemetry objects any KPIs/health states of service components computed by assurance agent (1) in the embodiment in which the assurance agent computes that information.

Network device (1) includes a NETCONF agent and an MDT producer. NETCONF agent receives network device configuration information from network orchestrator and configures service component(s) on network device (1) based on the network device configuration information. NETCONF agent also receives the monitoring objects from NETCONF agent, and configures the network device, including MDT producer, based on the monitoring objects. MDT producer, records its local Service metrics and its Service identifier in telemetry objects

as instructed by the monitoring objects, and may optionally include the corresponding service tags in the telemetry objects, and reports the telemetry objects to telemetry consumer.

ID	Role
102	Network Orchestrator
104	Service Operator
106	Assurance Orchestrator
108	Assurance Agent
110	Assurance Collectors
112	Network Device
113	Service Network
206	Netconf Agent
208	Telemetry Consumer
210	Telemetry Producer
211	Plugins

Table 5 Summary of the component ids in assurance orchestrator block diagram

5.9 Distributed Assurance System

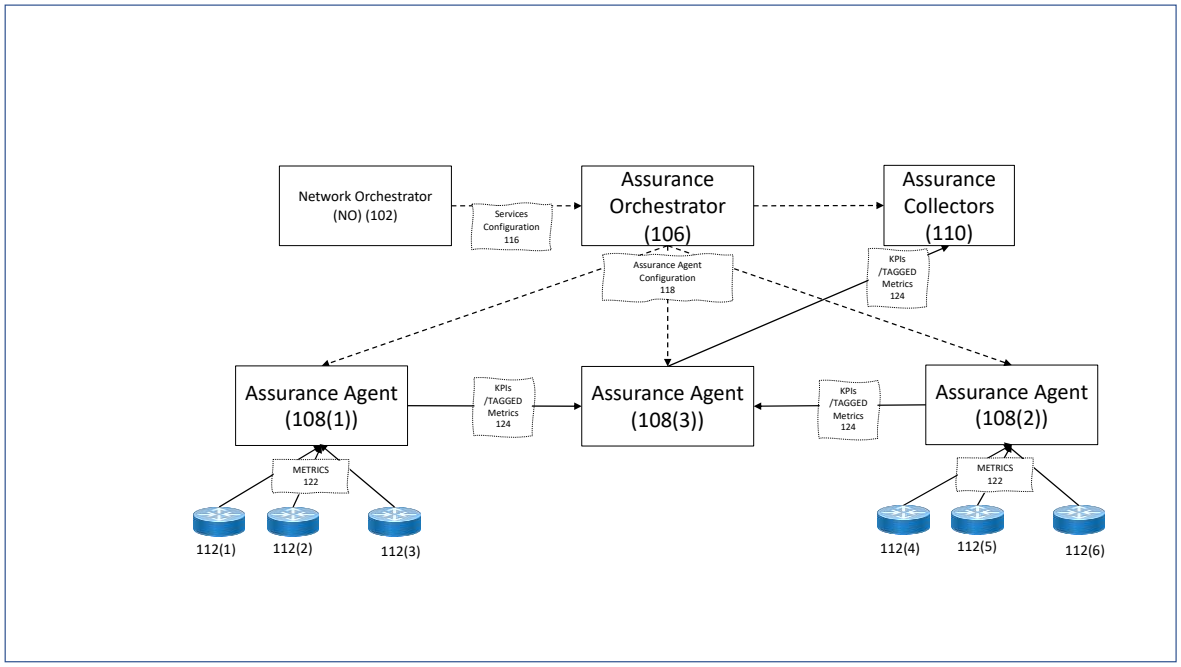

Figure 38 Distributed arrangement of assurance agents

With reference to Figure 38 there is a block diagram that shows an example of a distributed arrangement of Assurance agents and network devices of service assurance system. In the example of Figure 38, assurance agent (1) is co-located with network devices(1)-(3) at a first geographical location and assurance agent (2) is co-located with network devices(4)-(6) at a second geographical location separated from the first geographical location. Service 1 (see Figure 35) may be implemented on network devices(1)-(3), and Service 2 may be implemented on network devices(4)-(6). Geographically distributed Assurance agents(1) and (2) report their service-tagged telemetry objects to centralized assurance agent (3), which forwards the service-tagged Service metrics to assurance collector 110.

Examples of service configuration information for a service instance “xyz” (e.g., for a customer xyz) of service type L2 virtual private network (VPN) L2VPN, which is a peer- to-peer (p2p) connectivity type (i.e., L2VPN-p2p), are now described with reference to Figures. 3-5. In Figure 12, the example service configuration information is represented as eXtensible Markup Language (XML) encoding of YANG models.

5.10 Service Configuration Information/Definition Examples

With reference to Figure 36, there is an illustration of first example service configuration information for a first network device and an interface of service instance xyz. More specifically, lines introduce a “GigabitEthernet” interface for/on a first provider edge (PE) network device “pe-1” (e.g., a router) for service instance xyz of type “l2vpn” indicated at line. As indicated at lines 302, first network device pe-1 is running an XR operating system, by Cisco. Interface configuration line provides an identifier “0/0/0/3” for the GigabitEthernet interface. There is also definition of a maximum transmission unit (MTU) for the interface. Groups of lines define parameters for IPv4 and IPv6 addresses configured on the interface.

Second example service configuration information is present for a second network device of service instance xyz. More specifically, we’re introducing a second PE network device “pe-2” (e.g., a router) for service instance xyz of type “l2vpn”. We also define a QoS classification, as default, for traffic handled by the network device pe-2. Alternatively, or additionally, service configuration information may define a Quality-of-Experience (QoE) classification.

We also introduce third example service configuration information for a first cross-connect (“xconnect”) associated with second network device pe-2 for service instance xyz. An “xconnect” is a L2 pseudowire (L2 PW) used to create L2 VPNs (L2VPNs). Examples of xconnects are provided at [50], authored by C. Pignataro. In the present context of service assurance, “xconnect” refers to a syntax of a command used to realize the pseudowire in, for example, a Cisco internetwork operating system (IOS)-XR/IOS-XE operating system.

Second network device pe-2 provides with service instance xyz. We define the first xconnect, which is associated with a GigabitEthernet subinterface 0/0/0/2.600 with an IPv4 address 192.168.0.17.

5.11 Service Dependency Graph Example

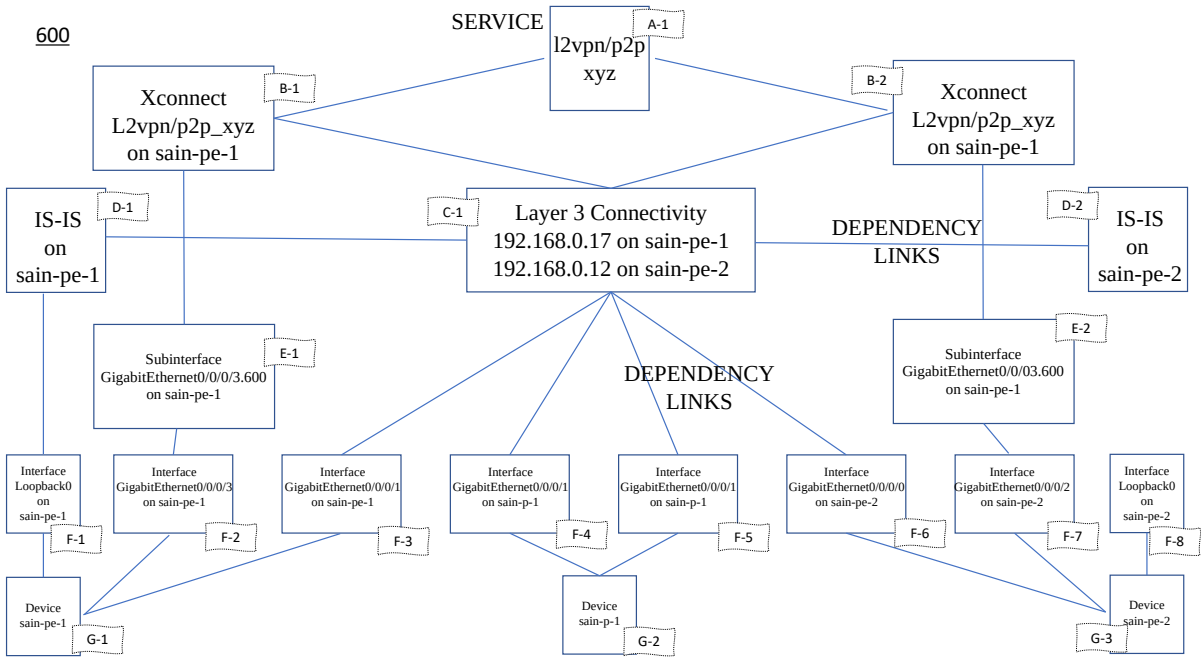

Figure 39 Service dependency graph

With reference to Figure 39, there is an illustration of an example Service dependency graph in the form of a Service dependency tree for service L2VPN-p2p, meaning an L2 VPN for a peer-to-peer connection. Service dependency tree (or "tree" for short) includes a service node A-1 at the highest level of the tree. Service node A-1 identifies/represents the service by a service tag tuple that includes service type and service instance, e.g., tuple <service type, service instance>. In the example of Figure 39, service node A-1 represents service <L2VPN-p2p, xyz>. Lower levels of tree are populated with Service nodes (shown as boxes) that identify/represent respective service components of the service <L2VPN-p2p, xyz>, and that connect back to service node A-1 through Service dependencies or parent-child links (shown as lines connecting boxes that depend on each other). Each of the Service nodes includes an identifier (e.g., a plain text identifier, as depicted in Figure 39) of the Service represented by that Service nodes. In the example of Figure 39, the lower levels of tree include:

a) second level that includes Service nodes B-1 and B-2 for xconnect service components implemented on network devices pe-1 and sain-pe2.
b) A third level that includes a Service node C-1 for an L3 network connectivity Service with components on network devices pe-1 and sain-pe2.

c) A fourth level that includes Service nodes D-1 and D-2 for routing protocol service components (e.g., Intermediate System to Intermediate System (IS-IS)) on network devices pe-1 and pe-2.
d) A fifth level that includes Service nodes E-1 and E-2 for subinterface service components on network devices pe-1 and pe-2.
e) A sixth level that includes Service nodes F-1 - F-8 for interface service components on network devices pe-1 or pe-2, as indicated.
f) A seventh level that includes Service nodes G-1 – G3 for network devices pe-1 and pe-2 as service components as indicated.

In one example branch of Service dependency tree, service <L2VPN-p2p, xyz> depends on the Service of Service node B-1, which depends on the Service of Service node E-1, which depends on the Service of Service node F-2, and so on down the levels of the tree. As indicated by the Service links, a given Service may depend on multiple other service components. Traversing the levels of tree downward from the highest level to the lowest level of the tree, the service components of service <L2VPN-p2p, xyz> include network xconnects on network devices (e.g., on pe-1 and pe-2), L3 network connectivity on the network devices (L2 network connectivity on the network devices may also be a service component), routing protocols on the network devices, interfaces of the network devices, subinterfaces of the network devices, and the network devices themselves.

Generally, the service components include: xconnects on network devices; L1 (e.g., optical), L2, and L3 network connectivity on the network devices; routing protocols on the network devices; interfaces of the network devices; subinterfaces of the network devices; communication behaviour of the interfaces and the subinterfaces; the network devices themselves and operations performed on/by the network devices. Service components also include logical network functions and groupings of logical and physical elements, such as: ECMP/ECMP groups of network devices; network tunnels; link protection functions executing in a network; network device protection functions executing in a network; and logical overlays on a physical network.

Logical overlays may include link aggregation for a link aggregation group (LAG); Virtual Extensible (Vx) LAN (VxLAN); VxLAN-Generic Protocol Extension (GPE); Generic Routing Encapsulation (GRE); service function chaining (SFC) functionality including Network Service Header (NSH) implementation; and Multiprotocol Label Switching (MPLS); for

example. The service components may also include applications such as application categorization as per RFC 6759. The service components may also include one or more multicast subnets on network devices.

5.12 Heuristic Packages

With reference to the above Figure 39 Service dependency graph , there is an illustration of an example generalized heuristic package generated based on a Service dependency graph. Heuristic package includes a header that identifies a Service of the Service dependency graph that is targeted by the heuristic package, and an overall function for which the heuristic package is to be used. For example, header may identify any specific one of xconnect, L3 connectivity, routing protocol, subinterface, interface, or network device, and the header may specify that the heuristic package is to be used to determine a health of the indicated service component.

Heuristic package may include arguments, which indicate various conditions under which the heuristic package is to be used, such as a time duration over which the Service is to be monitored. Heuristic package also includes expressions, which include measure and compute. Measure specifies Service metrics of the Service that are to be recorded. For example, for a network device service component, the Service metrics may include central processor unit (CPU) usage, free memory, temperature, power, and the like. For an interface of the network device, the Service metrics may include traffic rate, and so on. Compute provides rules and/or instructions to compute KPIs based on the Service metrics, and instructions to determine a health state for the service component, such as thresholds against which computed values are to be compared to determine the health state.

Compute may include rules to compute a health state that is binary, i.e., a health state that indicates either a passing health state when the Service is operating properly (e.g., meets a desired performance level) or a failing health state (which is a degraded health state) when the Service is not operating properly (e.g., does not meet the desired performance level). Alternatively, the rules may compute a health state that is graded, i.e., a health state that indicates a health state within a range of possible health states from passing to failing, e.g., including a passing health state, a failing health state, and a degraded health state that is not a passing health state or a failing health state (in this case, degraded means between passing and failing). In an example, the health states may include the following computed health state values: failing = 0, 0 < degraded < 1, passing = 1.

With reference to Figure 32 and Figure 39, there is an illustration of an example Heuristic package for a network device service component. Heuristic package includes header and arguments. Heuristic package includes compute to compute health indicators (KPIs) for a flash disk, flash, a hard disk, and storage, generally. For example, compute includes rules to set the health state to indicate a degraded health state if memory of a flash disk is full, and further rules to evaluate the following Boolean operation: flash_disk_free/flash_disk_size > 0.05, and so on. Heuristic package includes measure that lists power metrics to be measured (e.g., power demand), and compute to compute health states based on the power metrics. Heuristic package also includes compute to compute an overall health state (KPI) for the network device based on values computed in prior computes. That is, compute defines a rule expression to evaluate the overall health state of the Service based on the Service metrics and the computed (intermediate) values mentioned above.

With reference to Figure 39, there is an illustration of an example heuristic package for a network protocol (e.g., IS-IS) Service implemented on a network device. Heuristic package includes header and arguments. Heuristic package includes measure to measure metrics associated with IS-IS, including to determine lists of valid IPv4 and IPv6 IS-IS routes on the network device (e.g., from a forwarding or routing table in the network device). Heuristic package includes compute to compute KPIs that include various counts and stabilities of the IPv4 and the IPv6 IS-IS routes based on the metrics from measure, and to compute an overall health state, which is also a KPI, for IS-IS based on previously computed values/KPIs.

5.13 Assurance Collector Operations and User Interfaces

Further operations of assurance collectors are now described in connection with Figure 39, and with reference again to Figure 35. As mentioned above, assurance collectors receive/collect service-tagged Service metrics from Assurance agents for various services, Service dependency graphs for the various services, and heuristic packages for the various services. The Service dependency graphs each includes the service tag for the service to which the Service dependency graph pertains. The heuristic packages each includes the service tag to which the heuristic package pertains. Assurance collectors associate all service-tagged Service metrics (and health states of service components when available) tagged with a given service tag to the Service dependency graphs that includes the given service tag, and to the heuristic packages that include the given service tag. In other words, assurance collectors associate all

service-tagged metrics (and health states of service components), Service dependency graphs, and heuristic packages that have a matching (i.e., the same) service tag to each other and to the service identified by that service tag.

For each service, assurance collectors may populate the Service dependency graph with corresponding health states of the service components of the Service dependency graph as represented by the service-tagged Service metrics. For example, assurance collectors may populate the nodes of a Service dependency tree for the service with the health states of the service components represented by the nodes. In an embodiment in which Assurance agents provide the health states of the service components along with the service-tagged Service metrics to assurance collectors, the assurance collectors may populate the Service dependency tree with the provided health states. Alternatively, assurance collector 110 computes the health states of the service components from the corresponding service-tagged metrics in accordance with the corresponding heuristic packages, and then populates the Service dependency tree with the health states as computed.

The resulting Service dependency graph, populated with health states of the service components, may be generated for display to an administrator in a graph form (e.g., tree) or otherwise, e.g., as a list of service components for the service. Also, for each service, assurance collectors may determine an overall health state of the service (also referred to simply as a "health state" of the service) based on the health states of the service components of the service. For example, if all of the service components have health states that indicate passing health states, assurance collectors may set the overall health state to indicate a passing overall health state. Alternatively, if the health states of one or more of the service components indicate failing health states, assurance collectors may set the overall health state to indicate a failing overall health state.

5.14 Monitoring and Service-Tagged Telemetry Objects

Monitoring object includes a Service identifier (ID) and configuration information. Configuration information may include YANG network device configuration information, for example, and identifies Service metrics to be recorded and reported, in accordance with a heuristic package. Configuration information may include one or more configuration code snippets to configure a service component, e.g., a network device, to perform the recording/reporting of the Service metrics. For example, a heuristic package with instructions

to monitor (memory) “space available” for MPLS in a network device running IOS-XR may result in the following command line interface (CLI) code snippet in a monitoring object destined for the network device:

CLIMetric:

```
Command: show resource detail, regex_type: textfam,

regex: ios_xr/show_oef_rsource_detail.txt,

key: “space available” filter:

“node”

“mpls”

post_processing: convert2byte (GetTuple (value, 0),
GetTuple (value, 1)
```

Alternatively, the monitoring object may include a YANG object that performs the same function as the CLI code snippet. Alternative, the monitoring object may include binary information such as a packet.

Monitoring object may also include a service tag for the service to which the Service identified by the Service ID pertains.

With reference to Figure 36, there is an illustration of an example service-tagged telemetry object. Service-tagged telemetry object includes a Service identifier, a service tag, and information. Information includes recorded/reported Service metrics, computed values, and KPIs (including a health state of a service component) in accordance with a heuristic package from which a corresponding monitoring object was generated.

5.15 Service Assurance Operational Flow

With reference to Figure 37, there is a flowchart of an example method of performing assurance for a service enabled on a network. Assurance method may be performed by a system including

one or more entities to provide assurance for the service on the network. The one or more entities may include one or more of assurance orchestrator, Assurance agents, and assurance collectors.

Within the assurance flow, a definition of a configuration of a service is received, e.g., by assurance orchestrator. The definition includes a service type, a service instance, and configuration information used to enable or implement the service in the network.

Next, a service tag is generated from the service type and the service instance. For example, assurance orchestrator generates the service tag. The service tag identifies the specific instantiation of the service in the network, and is unique so as to distinguish the service from other services. The service tag may be a tuple that includes the service type and the server instance.

Following step, based on the configuration information of the definition, the service is decomposed into a graph of service components and dependencies between the service components that collectively actually implement the service in the network. The service tag is applied to the Service dependency graph. For example, assurance orchestrator decomposes the service into the Service dependency graph, and may provide the Service dependency graph to assurance collectors.

The service components are configured to record and report Service metrics indicative of health states of the service components (e.g., a respective health state of each of the service components) based on the Service dependency graph. The health states may respectively indicate either a passing health state or a failing health state. Alternatively, the health states may respectively indicate a health state within a range of health states including a passing health state, a failing health state, and a degraded health state that is not a passing health state or a failing health state. Operation may include the following further operations:

a) Based on the Service dependency graph, assurance orchestrator generates heuristic packages, typically one per service component, that specify the Service metrics that the service components are to record and report, and include rules to compute the health states of the service components based on the Service metrics. Assurance orchestrator provides to Assurance agents the heuristic packages and the service tag.

b) Responsive to the heuristic packages, Assurance agents generate from the heuristic packages monitoring objects that define the Service metrics that the service components are to record and report, and provide the monitoring objects to the service components to configure the service components to record and report the Service metrics.

Responsive to the configuring of operation, the Service metrics are obtained from the service components. For example, responsive to the monitoring objects, the service components record and then report to Assurance agents the Service metrics in telemetry objects corresponding to the monitoring objects.

Sservice tag is applied to the Service metrics to produce service-tagged Service metrics. For example, Assurance agents receive the telemetry objects, insert the service tag into the telemetry objects, and then send the (resulting) service-tagged telemetry objects to assurance collectors. Optionally, Assurance agents also analyse the Service metrics to compute health states of the service components in accordance with the rules in the heuristic packages, and insert the health states into the service-tagged telemetry objects before sending them to assurance collectors, which receive the service-tagged telemetry objects.

The service-tagged Service metrics are analysed to determine a health state of the service. For example, assurance collectors (i) associate the Service metrics in the service-tagged telemetry objects with the service based of the service-tagged telemetry objects, (ii) analyse the Service metrics to compute individual health states of the service components (unless the health states are included with the service-tagged telemetry objects), e.g., one health state per service component, based on the rules in the heuristic packages, and (iii) determine an overall health state of the service based on the individual health states of the service components, which were associated with the service based on the service tags at (i). For example, if all of the health states of the service components indicate passing health states, the overall health state may be set to indicate a passing overall health state. Alternatively, if one or more of the health states of the service components indicate failing health states, the overall health state may be set to indicate a failing overall health state. Alternatively, if one or more of the health states of the service components indicate degraded (not failing or passing) health states, and there are no failing health states, the overall health state may be set to indicate a degraded (not failing or passing) overall health state.

In addition, assurance collectors populate indications of the service components in the Service dependency graph with their respective health states, and generate for display the populated Service dependency graph to provide visual feedback. In various embodiments, operations performed by assurance collectors as described above may be shared between the assurance collectors and assurance orchestrator. In another embodiment in which assurance collectors are omitted, Assurance agents send service-tagged Service metrics (and health states) directly to assurance orchestrator, and the assurance orchestrator performs all of the operations performed by the assurance collectors as described above. That is, assurance orchestrator operates as the assurance orchestrator and assurance collectors.

In an environment that includes multiple services, assurance method is performed for each service, by the one or more entities, to produce, for each service, respectively, a unique service tag, a Service dependency graph, heuristic packages, monitoring objects, telemetry objects, tagged telemetry objects, health states of service components, and an overall service health state. The one or more entities use the unique service tags to distinguish between the services and the aforementioned information generated for the services.

5.16 Closed Loop Automation for Intent-based Networking

Closed loop automation for intent-based networking is now described. Closed loop automation for intent-based networking discovers an overall health state of a service comprising a collection of service components based on health states of the service components, using techniques described above, for example. If the closed loop automation discovers that the overall health state of the service (also referred to as the “service health state”) indicates a failing overall health state, the closed loop automation reconfigures the service components so that the overall health state indicates a passing overall health state. In other words, the closed loop automation provides feedback to “close the loop” in service assurance system to restore the overall health state of the service to an acceptable, passing overall health state.

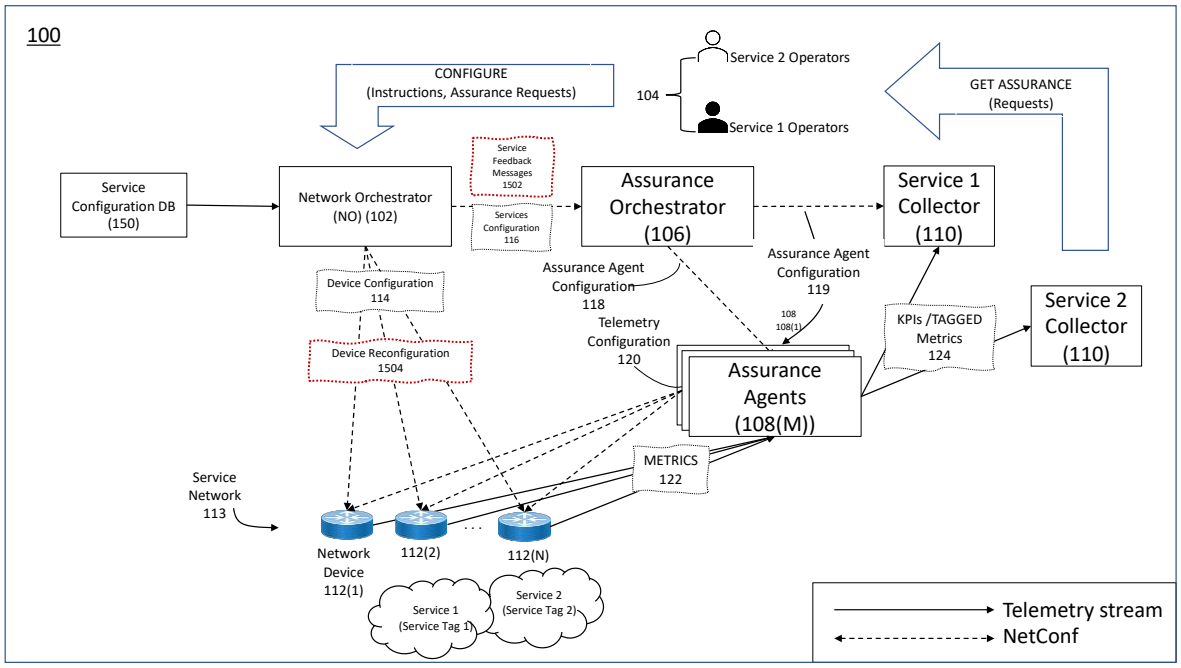

Figure 40 Service Assurance system adapted to perform the closed loop automation.

With reference to Figure 40, there is shown a block diagram of service assurance system adapted to perform the closed loop automation. Figure 40 is similar to Figure 35, except that Figure 40 shows additional flows used for the closed loop automation. For closed loop automation, assurance orchestrator determines an overall health state of each of the services implemented on service network, and then provides to network orchestrator service assurance messages (also referred to as "flow"). Service assurance messages include the overall health states for the services as determined by assurance orchestrator, and may also include health states of service components for each of the services. Service assurance messages may also include, for each of the services having an overall health state that indicates a failing (or degraded) overall health state, a corresponding request to reconfigure service components of that service, so as to return the overall health state to a passing overall health state. The request to reconfigure may also be referred to as a "Service reconfiguration request."

ID	Role
100	Service Assurance System
102	Network Orchestrator
104	Service Operator
106	Assurance Orchestrator
108	Assurance Agent
110	Service Collector
112	Network Device
113	Service Network

Table 6 Device ID summary table

Responsive to each request to reconfigure service components of a service received in service assurance messages, network orchestrator reconfigures the service components of the service, as identified in the request. To reconfigure the service components, network orchestrator provides Service reconfiguration information (also referred to as "flow") to the network devices among network devices that host/implement the service components to be reconfigured.

Service reconfiguration information may be formatted similarly to network device configuration information, and may be provided to network devices similarly to the way in which the network device configuration information is provided to the network devices.

We also need to mention the flowchart of an example assurance method of closed loop automation for intent-based networking performed in service assurance system, as depicted in Figure 40, for example. Assurance method incorporates various operations described above. The terms "health state of a service component" and "Service health state" are synonymous and interchangeable.

Network orchestrator configures a service as a collection of service components on network devices of a network, stores a definition of the service in service configuration database Figure 36, and provides the definition to assurance orchestrator. The definition includes a service type, a service instance, and configuration information, as described above.

Assurance orchestrator generates a service tag based on the definition of the service, and decomposes the definition into a Service dependency graph that indicates the service components and dependencies between the service components that collectively implement the service.

Based on the Service dependency graph, assurance orchestrator and Assurance agents, collectively, configure the service components to record and report Service metrics indicative of health states of the service components.

Assurance agents and assurance orchestrator, collectively, obtain the Service metrics from the service components, tag the Service metrics with the service tag, and determine the health states of the service components based on the Service metrics.

Assurance orchestrator determines an overall health state of the service based on the health states of the service components. In an example, assurance orchestrator populates Service nodes of the Service dependency graph with data representing respective ones of the health states of the service components, and searches the Service nodes for any of the health states of the service components that indicate a failing health state. Assurance orchestrator may generate for display the Service dependency graph populated with the data representing the health states of the service components, as shown in Figure 39, for example.

If the overall health state indicates a failing overall health state, assurance orchestrator identifies one or more of the service components as the service components that are responsible for the failing overall health state based on the health states of the service components. Assurance orchestrator generates one or more service assurance messages that include (i) the identifier of the service (e.g., the service tag), (ii) the overall health state that indicates the failing overall health state, (iii) identifiers and health states of at least the one or more service components that are responsible for the failing overall health state, and (iv) a request to reconfigure the one or more of the service components. The one or more service assurance messages may report health states of all of the service components, not just those of the one or more service components responsible for the failing overall health state. Assurance orchestrator provides the one or more service assurance messages to network orchestrator, as indicated in flow of Figure 40.

Responsive to the one or more service assurance messages, including the request to reconfigure the one or more service components, network orchestrator reconfigures the one or more service components. To do this, network orchestrator (i) uses the identifier of the service and the identifiers of the one or more service components from the one or service assurance messages as indexes to access/retrieve the Service configuration information for the one or more service components from the service definition stored in service configuration database Figure 36, (ii) generates Service reconfiguration information based on the Service configuration information retrieved from the service definition, and (iii) provides the Service reconfiguration information to network devices among network devices that host/implement the one or more service components, as indicated in flow of Figure 40. In response to the Service reconfiguration information, network devices reconfigure the one or more service components to implement the requested Service reconfiguration, thus closing the loop for the closed loop automation process. The entire closed loop automation is performed without manual intervention.

Network orchestrator may reconfigure the one or more service components in many different ways to improve the overall health state of the service, e.g., to change the overall health state from failing to passing. In one example, network orchestrator may simply repeat the operations used to configure the one or more service components as performed in the flow, in which case the Service reconfiguration information may include the same intent-based network device configuration objects that were used to initially configure the one or more service components. For example, network orchestrator may repeat the operations described above in connection with Figure 36 Example of service configuration database to configure one or more of an L1 connection/interface, an L2 connection/interface, an L3 connection/interface, a packet routing protocol, ECMP, traffic shaping, and so on, as identified in the request to reconfigure.

In another example, network orchestrator may reboot the one or more service components. To reboot a service component, network orchestrator may first validate permissions and user authorizations for the Service as provided in the service definition, force a process that implements the Service to enter a privileged mode, and then reboot the process or an operating system that hosts the process. Alternatively, network orchestrator may implement a process to perform a network device repair or link repair for critical network devices and/or links associated with the one or more service components.

In yet another example, network orchestrator may not completely reconfigure the one or more service components, but rather may adjust/modify selected operational parameters of the one

or more service components (from their initially configured values) to improve their operational performance. This constitutes only a partial or limited reconfiguring of the one or more service components. For example, for the one or more service components, network orchestrator may:

a) Adjust routing metrics, such as cost routing.
b) Modify L2 QoS, MTU, or adjust storm control policies (L2).
c) Change optical transmission power or hardcode interface speed (L1).
d) Adjust L3 QoS or MTU (L3).
e) Change ECMP hashing inputs (e.g., use destination IP instead of source IP).
f) Perform traffic shaping by modifying QoS to assure a desired level of traffic throughput.
g) Adjust interface traffic throughput (e.g., bandwidth).

In even further examples, service assurance system may reprogram the level of detail and frequency of telemetry collection per network device in order to investigate in greater detail why the overall health state of the service indicates the failing health state.

If the overall health state indicates a passing overall health state, assurance orchestrator generates one or more service assurance messages that include (i) the identifier of the service, (ii) the overall health state that indicates the passing overall health state, and (iii) identifiers and health states of the service components. The one or more service assurance messages do not include a request to reconfigure service components. Assurance orchestrator provides the one or more service assurance messages that do not include the request to network orchestrator in flow Figure 37. Responsive to the one or more service assurance messages that do not include the request, network orchestrator does not reconfigure any service components.

With reference to FIG. 17, there are shown example operations 1700 expanding on operation 1610 used to determine the overall health state of the service in method 1600.

At 1702, assurance orchestrator computes each of the health states of the service components to respectively indicate a passing health state or a failing health state for a corresponding one of the service components.

At 1704, assurance orchestrator determines the overall health state of the service as follows:

a) When one or more of the health states of the service components indicate the failing health state, set the overall health state of the service to indicate a failing overall health state.
b) When all of the health states of the service components indicate a passing health state, set the overall health state of the service to indicate a passing overall health state.

We compute each of the health states of the service components, respectively, to indicate a health state within a range of possible health states, including a passing health state, a failing health state, and a degraded health state that is neither the passing health state nor the failing health state.

Next is to determine the overall health state of the service to indicate that the overall health state is within a range of possible overall health states, including the passing overall health state, the failing overall health state, and a degraded overall health state that is neither the passing overall health state.

5.17 Computer System for Assurance Entities

In summary, embodiments presented herein, proposed solution for service assurance for intent-based networking, for example, uses service tagging of Service metrics recorded and reported by service components of a service to help an assurance orchestrator/collector "find a needle in the haystack" with respect to identifying Service problems that impact the service. This tagging helps the assurance orchestrator/collector asses all of the services that can be affected by particular telemetry data/sensor. The tagging facilitates specific export for data reduction, and filtering. The assurance orchestrator/collector can deterministically flag the services, including its service components, which need user attention or can provide feedback for remediation. Example high-level operations include:

a) Get a service configuration from an assurance orchestrator. The service configuration includes a service type and a service instance.
b) Create a specific service tag from the service configuration, e.g., <service type/service instance (e.g., identifier> tuple.
c) Using the service configuration, an assurance platform, e.g., the assurance orchestrator, decomposes the service into a series of service components for that specific service type/instance with rules of heuristic packages.

d) Tag service components metrics with the service tag.
e) To monitor a specific customer service instance, request all tagged Service metrics with the specific service tag.
f) When determining service performance based on key performance indicators (KPIs), in case of service degradation/failure, identify the specific component(s)/service components that has failed based on the service tag. Reconfigure the service (or network on which the service is enabled) to avoid the fault component.

In one form, a method is provided comprising: configuring a service as a collection of service components on network devices of a network; decomposing a definition of the service into a Service dependency graph that indicates the service components and dependencies between the service components that collectively implement the service; based on the Service dependency graph, configuring the service components to record and report Service metrics indicative of Service health states of the service components; obtaining the Service metrics from the service components and determining the Service health states of the service components based on the Service metrics; determining a health state of the service based on the Service health states; and reconfiguring one or more of the service components based on the health state of the service.

In yet another form, a computer readable medium is provided. The computer readable medium stores instructions that, when executed by one or more processors coupled to one or more network interface units, cause the one or more processors to perform: configuring a service as a collection of service components on network devices of a network; decomposing a definition of the service into a Service dependency graph that indicates the service components and dependencies between the service components that collectively implement the service; based on the Service dependency graph, configuring the service components to record and report Service metrics indicative of Service health states of the service components; obtaining the Service metrics from the service components and determining the Service health states of the service components based on the Service metrics; determining a health state of the service based on the Service health states; and reconfiguring one or more of the service components based on the health state of the service.

6 Conclusion and Future work

By leveraging the novel approach, the service definition and construction of this expression graph we were able to reduce amount of telemetry data exported from the network and export only service-intent relevant information instead of raw device data. This essentially means that it's possible to determine service health at the edge and contribute to service assurance in much more efficient manner than traditional means of telemetry compression or establishing different channels to send same amount of raw telemetry data.

We've presented the designed architecture which is able to, at almost real-time, perform analysis of the data streams and perform computations to establish the service health status.

Service decomposition is performed in at least two phases:

- Into an assurance graph, as depicted in Figure 41
- Transforming the assurance graph into an expression graph, as depicted in Figure 42

The information needed to execute these steps is contained in *heuristic packages*. Such a package contains 3 layers:

- *Rules*: specify how to build an assurance graph composed of service components out of the configuration pushed to enable the service.
- *Service component*: computes a health score and symptoms out of abstract metrics
- *Metric Engine*: maps abstract metrics to device-specific metrics implementations

There are several disadvantages and operational observations related to closed loop automation. Some of the most notable points, disadvantages include:

1. Complexity: Closed loop automation systems can be complex to design and implement. They require sophisticated algorithms and integration with various components, which can increase the complexity of the overall system.

2. Cost: Implementing closed loop automation can be expensive. It often involves investing in advanced technologies, software, and hardware, as well as training personnel to operate and maintain the system.

3. Maintenance and Support: Closed loop automation systems require ongoing maintenance and support. This includes regular updates, troubleshooting, and addressing any issues or bugs

that may arise. It's essential to have dedicated resources to ensure the smooth functioning of the automated system.

4. Limited Flexibility: Once a closed loop automation system is implemented, making changes or modifications can be challenging. The system is designed to operate in a specific way, and any changes may require significant reconfiguration or even redevelopment, resulting in a lack of flexibility.

5. Data Dependence: Closed loop automation heavily relies on accurate and up-to-date data. If the data inputs are incorrect or incomplete, it can negatively affect the system's performance and decision-making capabilities.

6. Potential for Errors: Despite being automated, closed loop systems are not immune to errors. Software bugs, hardware failures, or unforeseen circumstances can lead to errors or incorrect decisions. Regular monitoring and human oversight are necessary to identify and rectify any errors promptly.

7. Resistance to Change: The implementation of closed loop automation might face resistance from employees or stakeholders who may feel threatened by the automation of certain tasks or processes. Overcoming this resistance and ensuring proper training and support for employees is crucial for successful implementation.

Operational observations related to closed loop automation include:

1. Improved Efficiency: Closed loop automation can significantly enhance operational efficiency by automating repetitive tasks, reducing manual intervention, and streamlining workflows. This leads to faster processing times, reduced errors, and increased productivity.

2. Enhanced Decision-making: Closed loop automation systems can analyse vast amounts of data and make real-time decisions based on predefined rules and algorithms. This can lead to quicker and more accurate decision-making, especially in scenarios where immediate action is required.

3. Scalability: Closed loop automation systems can be scaled to accommodate growing workloads or to handle additional processes. This scalability allows organizations to adapt to changing demands and expand their automation capabilities as needed.

4. Risk Mitigation: Closed loop automation can help mitigate risks by detecting anomalies, identifying potential issues, and triggering appropriate actions or alerts. This proactive approach reduces the likelihood of errors, improves system resilience, and minimizes downtime.

5. Increased Customer Satisfaction: By automating certain processes, closed loop automation can improve response times, reduce errors, and enhance overall service quality. This leads to higher customer satisfaction levels as customers experience faster and more efficient service.

It is important to note that the specific disadvantages and operational observations of closed loop automation can vary depending on the industry, use case, and implementation approach.

Metric Engine: maps abstract metrics to device-specific metrics implementation

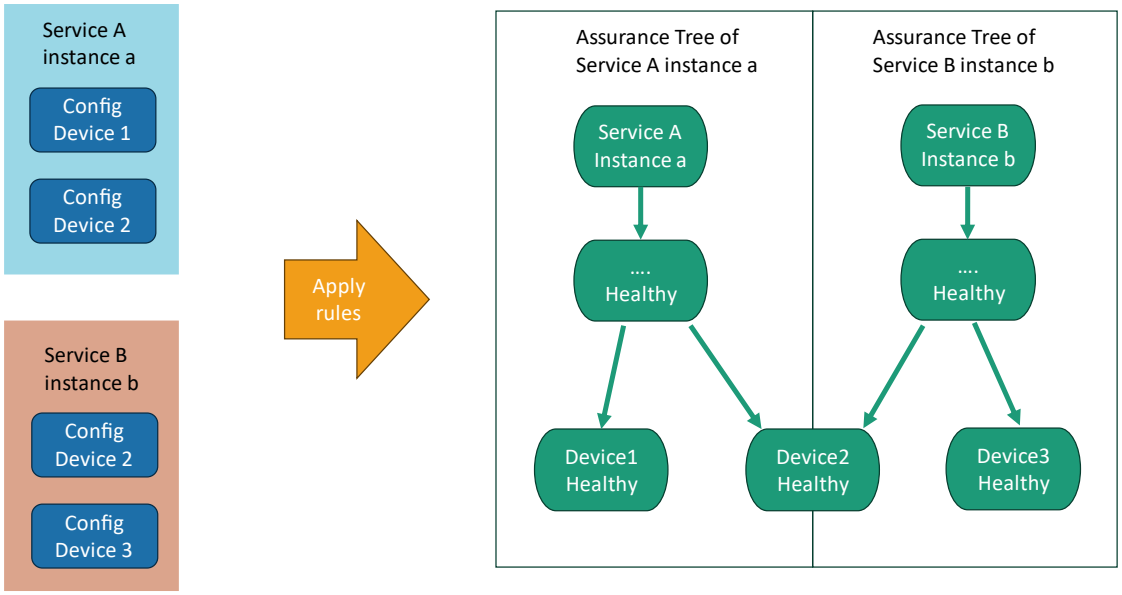

Figure 41 Applying rules on the configuration of a service instance results in an assurance graph that connects service components according to their dependencies.

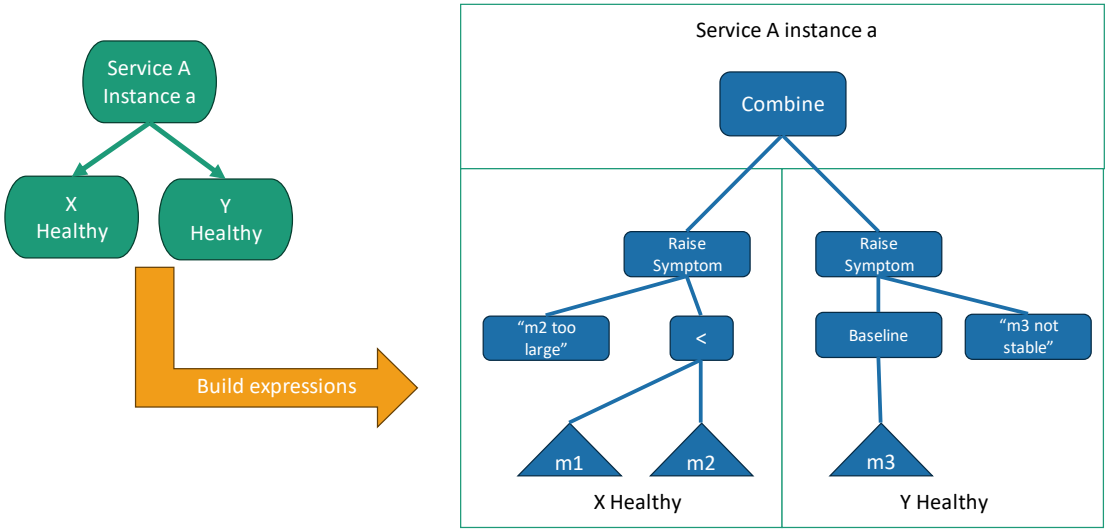

Figure 42 Transformation of an assurance graph with one service instance depending on two service components into an expression graph

Following step is to apply techniques described in the above work to the ontology-based system, which could be used as foundation to the reasoning engine – for analysis and automated error detection.

Expression tree for the Service can be quite complex, which makes it inherently difficult to navigate through large number of different computations and dependencies.

By applying advanced data modelling techniques, such as Ontologies, we could establish inferred relationships between raw data sources such the MDT and legacy protocols: SNMP, Syslog, RMON or even CLI to ensure higher level of precision and relevance within calculated assurance value for any specific Service assurance expression tree.

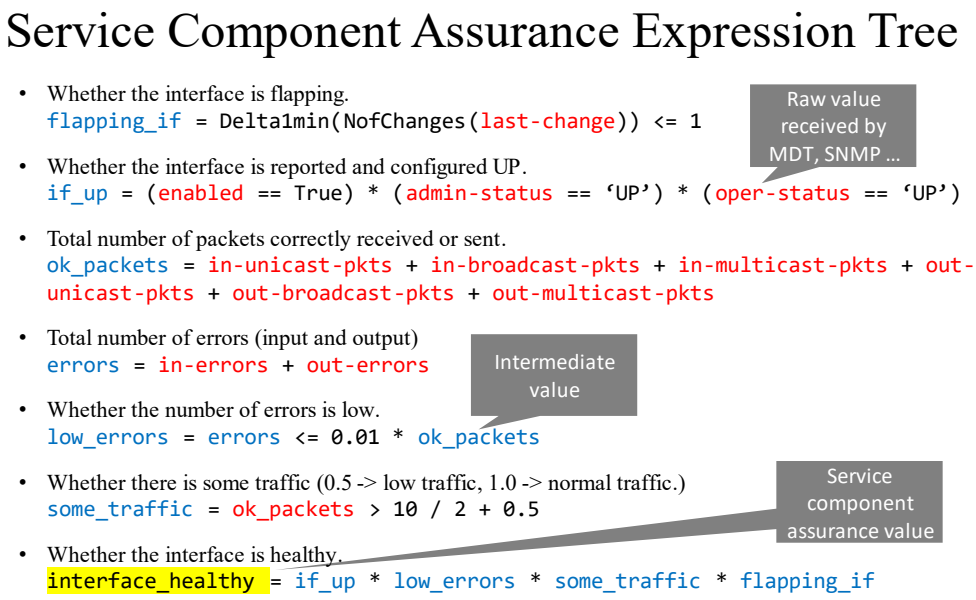

Figure 43 Relationship between raw data source and Service assurance calculation

Following the Ontology based model and reasoning engine, we could potentially insert inferred rules determine causal relationships using reasoning engine instead of heuristic packages build by a Subject Metter Experts.

List of publications

[1] Mioljub Jovanovic, Milan Cabarkapa and Djuradj Budimir, “Network Service Assurance and Telemetry optimisation using Heuristics”, WSEAS Transactions on Communications, Vol. 21, pp. 244-249, September 2022, DOI: 10.37394/23204.2022.21.29.

[2] Mioljub Jovanovic, Milan Cabarkapa and Djuradj Budimir,” Network Service Decomposition and Telemetry Optimisation using Heuristics, IEEE International Conference on Circuits, Systems, Communications and Computers, Chania, Crete Island, Greece July 19-22, 2022

[3] Mioljub Jovanovic, Milan Cabarkapa and Djuradj Budimir,” Design of a Network Topology Using CISCO NSO Orchestrator”, IcETRAN 2021 Conference, 8-10 September 2021, Ethno village Stanišići, Republic of Srpska, B&H.

[4] M. Jovanović, M. Čabarkapa, B. Claise, N. Nešković, M. Prokin и D. Budimir, “Model driven telemetry using Yang for next generation network applications” в *5th International Conference on Electrical, Electronic and Computing Engineering - IcETRAN*, 2018, Palic, Serbia.

7 References

[1] ONAP, April 2019. [Online]. Available: https://www.onap.org. [Accessed 23 4 2019].

[2] Apstra, “The Apstra Mission to Enable Business Agility through Autonomous Infrastructure,” 2019. [Online]. Available: https://www.apstra.com/company/about-apstra/.

[3] Veriflow, “Intent-Based Networking A business driven approach to align the network to desired outcomes,” 2019. [Online]. Available: https://www.veriflow.net/intent-based-networking/.

[4] Huawei, “Huawei Launches the Intent-Driven Network Solution to Maximum Business Value,” February 2018. [Online]. Available: https://www.huawei.com/en/press-events/news/2018/2/Huawei-Launches-the-Intent-Driven-Network-Solution. [Accessed 2019].

[5] B. Raouf and I. Aib, “Policy-based management: A historical perspective,” *Journal of Network and Systems Management,* vol. 15.4, pp. pp.447-480, 2007.

[6] J. Walker and A. Kulkarni, “Common Open Policy Service (COPS) Over Transport Layer Security (TLS) - RFC4261,” 2005.

[7] Simplified Use of Policy Abstraction (SUPA) Working Group, “Simplified Use of Policy Abstraction (SUPA),” [Online]. Available: https://datatracker.ietf.org/wg/supa/charter/. [Accessed].

[8] Y. Tsuzaki and . Y. Okabe, “Reactive configuration updating for Intent-Based Networking”,,” in *International Conference on Information Networking (ICOIN)*, Da Nang, 2017.

[9] Cisco Systens, Inc., “Intent Based Networking,” Cisco Systems, Inc., 2019. [Online]. Available: https://www.cisco.com/c/en/us/solutions/intent-based-networking.html. [Accessed 2019].

[10] M. E. Epperly and B. J. Walls, “Level-0 Telemetry Collection for Spacecraft Command and Data Handling Subsystems,” in *2001 IEEE Aerospace Conference Proceedings (Cat. No.01TH8542)*, Big Sky, MT, USA, 2001.

[11] J. Pérez-Romero, V. Riccobene, F. Schmidt, O. Sallent, E. Jimeno, J. Fernández, A. Flizikowski, I. Giannoulakis and E. Kafetzakis, “Monitoring and Analytics for the Optimisation of Cloud Enabled Small Cells,” 2018.

[12] Gartner, [Online].

[13] “Tacker,” [Online]. Available: https://docs.openstack.org/tacker/latest/.

[14] R. Mijumbi, J. Serrat, J. l. Gorricho, S. Latre, M. Charalambides and D. Lopez, “Management and Orchestration Challenges in Network Functions Virtualization,” *IEEE Communications Magazine,* vol. 54, no. 1, pp. 98-105, 2016.

[15] Cisco Systems, Inc., “Cisco Network Services Orchestrator,” April 2019. [Online]. Available: https://www.cisco.com/c/en/us/solutions/service-provider/solutions-cloud-providers/network-services-orchestrator-solutions.html. [Accessed 2019].

[16] J. Gonzalez, G. Nencioni, A. Kamisiski, B. E. Helvik and P. E. Heegaard, “Dependability of the NFV Orchestration: State of the Art and Research Challenges,” 2018.

[17] M. Pattaranantakul, R. He, Z. Zhang, A. Meddahi and P. Wang, “Leveraging Network Functions Virtualization Orchestrators to Achieve Software-Defined Access Control in the Clouds,” *IEEE Transcactions on dependable and secure computing,* pp. 1-14, 2018.

[18] A. Rao, “Reimagining service assurance for NFV, SDN and 5G,” 2018.

[19] A. D'Alconzo, I. Drago, A. Morichetta, M. Mellia and P. Casas, “A Survey on Big Data for Network Traffic monitoring and Analysis,” *IEEE Transactions on Network and Service Management,* vol. 16, no. 3, pp. 800-813, 2019.

[20] Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano and O. M. Caicedo, “A Comprehensive Survey on Machine Learning for Networking: Evolution, Applications and Research Opportunities,” *J. Internet Serv. Appl.,* vol. 9, no. 16, 2018.

[21] Cisco Systems, Inc, “GitHub Network Telemetry Pipeline,” 2017. [Online]. Available: https://github.com/cisco/bigmuddy-network-telemetry-pipeline.

[22] M. Jovanović, M. Čabarkapa, B. Claise, N. Nešković, M. Prokin and D. Budimir, “Model driven telemetry using Yang for next generation network applications,” 2018.

[23] B. Claise, J. Clarke and J. Lindblad, Network Programmability with YANG: The Structure of Network Automation with YANG, NETCONF, RESTCONF, and gNMI, Addison-Wesley, 2019.

[24] IETF, “Writable MIB Module IESG Statement,” 2018. [Online]. Available: https://www.ietf.org/iesg/statement/writable-mib-module.html. [Accessed 2018].

[25] Z. Liu, J. Bi, Y. Zhou, Y. Wang and Y. Lin, “Netvision: Towards network telemetry as a service,” in *2018 IEEE 26th International Conference on Network Protocols (ICNP)*, 2018.

[26] R. Hohemberger, A. G. Castro, F. G. Vogt, R. B. Manshilha, A. F. Lorenzon, F. D. Rossi and M. C. Luizelli, “Orchestrating In-Band Data Plane Telemetry with Machine Learning,” *Communications Letters IEEE*, vol. 23, no. 12, pp. 2247-2251, 2019.

[27] A. G. Castro, A. F. Lorenzon, F. D. Rossi, R. I. T. da Costa Filho, F. M. V. Ramos, C. E. Rotherberg and M. C. Luizelli, “Near-Optimal Probing Planning for In-Band Network Telemetry,” *Communications Letters IEEE*, vol. 25, no. 5, pp. 1630-1634, 2021.

[28] E. F. Kfoury, J. Crichigno and E. Bou-Harb, “An Exhaustive Survey on P4 Programmable Data Plane Switches: Taxonomy Applications Challenges and Future Trends,” *Access IEEE*, vol. 9, pp. 87094-87155, 2021.

[29] S. Tang, D. Li, B. Niu, J. Peng and Z. Zhu, “Sel-INT: A Runtime-Programmable Selective In-Band Network Telemetry System,” *Network and Service Management IEEE Transactions on*, vol. 17, no. 2, pp. 708-721, 2020.

[30] D. Scano, “Augmented In-Band Telemetry to the User Equipment for Beyond 5G Converged Packet-Optical Networks,” 2020.

[31] Netrounds, 2019. [Online]. Available: https://www.netrounds.com. [Accessed 2019].

[32] May 2020. [Online]. Available: https://onap.biterg.io/. [Accessed 5 2020].

[33] W. Zegeye, R. Dean, M. Dugda, F. Moazzami and A. Bezabih, “Modeling Networked Telemetry,” vol. 10, no. 45, 2021.

[34] R. Rokui, H. Yu, L. Deng, D. Allabaugh, M. Hemmati and C. Janz, “A Standards-Based, Model-Driven Solution for 5G Transport Slice Automation and Assurance,” 2020.

[35] T. Feltin, P. Foroughi, W. Shao, F. Brockners and T. H. Clausen, “Semantic feature selection for network telemetry event description,” 2020.

[36] R. A. K. Fezeu and Z.-L. Zhang, “Anomalous Model-Driven-Telemetry Network-Stream BGP Detection,” 2020.

[37] R. Potluri and K. Young, “Analytics Automation for Service Orchestration,” 2020.

[38] K. S. Mayer, J. A. Soares, R. P. Pinto, C. E. Rothenberg, D. S. Arantes and D. A. A. Mello, “Soft Failure Localization Using Machine Learning with SDN-based Network-wide Telemetry,” 2020.

[39] B. Martin, 2010. [Online]. Available: https://tools.ietf.org/html/rfc6020. [Accessed 2018].

[40] R. Enns, B. M. J. Schoenwaelder and A. Bierman, “Network Configuration Protocol (NETCONF),” [Online]. Available: https://tools.ietf.org/html/rfc6241. [Accessed 2018].

[41] D. Bodganovic, B. Claise and C. Moberg, “YANG Module Classification,” July 2017. [Online]. Available: https://tools.ietf.org/html/rfc8199. [Accessed 2018].

[42] B. Claise and J. Clarke, “YANG Catalog,” [Online]. Available: http://www.yangcatalog.org. [Accessed 2018].

[43] E. Voit, A. Clemm and A. Gonzalez Prieto, “Requirements for Subscription to YANG Datastores,” June 2016. [Online]. Available: https://tools.ietf.org/html/rfc7923. [Accessed 2018].

[44] Openconfig, “Openconfig,” [Online]. Available: http://www.openconfig.net/. [Accessed 2018].

[45] Cisco Systems, Inc., “YANG Development Kit,” [Online]. Available: https://developer.cisco.com/site/ydk/. [Accessed 2018].

[46] L. Poulopoulos, “Python library for NETCONF clients,” [Online]. Available: https://github.com/ncclient. [Accessed 2018].

[47] Cisco Systems, Inc, “Cisco Virtual Routing Lab,” 2018. [Online]. Available: http://virl.cisco.com/. [Accessed 2019].

[48] P. Amaral, “Machine learning in software defined networks: Data collection and traffic classification,” 2016.

[49] L. Lhotka, “Defining and Using Metadata with YANG,” 2016. [Online]. Available: https://www.rfc-editor.org/rfc/rfc7952.

[50] C. Pignataro and W. Luo, Layer 2 VPN Architectures, Cisco Press, 2004.

8 Appendixes

8.1 Appendix A - Service Language

The present language was developed to describe the computation needed to evaluate the status of a service component.

Overview

A service is specified by the following elements:

- a name and a list of arguments
- a list of metrics to collect
- a list of expressions to combine the metrics into a single status value

We show how to define each of these elements through examples. The complete specification of the syntax is detailed below.

Global structure of a file

Each file as the following structure:

- a single level 1 header (The name of the service component)
  A description of the Service
  - a single level 2 header 'Arguments'
    The arguments of the Service
  - a single level 2 header 'Expressions'
    a sequence of level 3 blocks (either 'Measure' or 'compute')
    - 'Measure' blocks contain a list of metrics to monitor
    - 'Compute' blocks contain a list of expressions to compute

Service component example

This file is a minimal example for a service component

Arguments

- device: The device on which the metric is collected

- str param_2: A second parameter

Expressions

Measure

A list of metrics to get. These need to be already defined as metrices, within our solution, to actually produce values.

- int metric_1 = example.first(device=device)
  a first metric of type int named metric 1. It is parameterized by the device.

Compute

We check whether metric 1 is positive. Otherwise, we raise a symptom saying that metric_one is not positive.

Note that since we raise a symptom, the value of m1_pos is actually a `Status` object.

- m1_pos: Whether metric_1 is positive degraded if false -> Metric 1 is negative
  - `metric_1 > 0`

Measure

Another list of metrics to get.

- str metric_2 = example.second(device=device, another_param=param_2)
  another metric that also depends on param2

Compute

We check that metric_2 is equal to foo. If not, we raise a symptom containing the actual value of metric 2.

- m2_foo: Whether metric_2 is equal to foo. broken if false: Metric 2 is not foo but `metric_2`
  - `metric_2 == "foo"`

Compute

We combine the two statuses to obtain the final value of the service component.

- Service component example: the top-level status
    - `Combine(m1_pos, m2_foo)`

Name and arguments

The name is the level 1 header of the file. Arguments are defined in a list using '*' as an item:

```
# InterfaceHealthy

Checks whether a given interface on a given device is healthy.

## Arguments

  * device:  Device supporting the interface to check.
  * str interface:  Name of the interface to check.
```

The above example defines the name and arguments of `InterfaceHealthy`. An instance of this Service is totally parameterized by a device (of type device) and an interface (of type str).

Metrics

Each metric to collect is defined via a name and a set of parameters. An example is

```
  *  str  admin_status  =  interface.administrative_status(device=device,
interface=interface)
     _ Whether the interface is currently enabled_
```

Without further specification, the parameters are picked among the arguments of the Service component. In order to allow a parameter to be the result of a computation, it is necessary to specify a dynamic parameter just after the name of the metric:

```
  *   underlay_mtu[underlay_interface]   =   interface.mtu(device=device,
interface=underlay_interface)
    _ The mtu of an underlay interface_
```

The parameter 'interface' of the metric is left unspecified. When referring to `underlay_mtu` in an expression, the `underlay_interface` value must be specified as well (see below.)

Expressions

Each expression is defined via a name, optionally a symptom to raise, and a list of potential expressions. An example reusing the previous metric is

```
  * is_up:  Whether the interface is currently enabled
    broken if false: Interface is down
    + `admin_status == "UP"`
```

Here a single expression is used. If this expression evaluates to false, then a symptom is raised. However, depending on the available metrics, we might want to use different expression to compute a given value. For instance, assuming that two devices:

- device 1 provides "total_memory" and "free_memory" metrics
- device 2 provides "total_memory" and "used_memory" metrics

In that case, if we want to check that at least 10% of the memory is available, one can write:

```
  * memory_healthy: At least 10% of the memory is available
     + `free_memory/total_memory > 0.1`
     + `Minus(total_memory,used_memory)/total_memory > 0.1`
```

Dynamic elements

Only the first expression for which all the names are available will be bound to the name "memory_healthy".

As for metrics, if an expression needs to be repeated with various parameters, it can be parameterized.

```
  * compatible_mtu_on[interface]: Check that the MTU of interface is large
enough
    + `underlay_mtu[interface] > encap_size + mtu_overlay`
```

Here

- `interface` is declared as a parameter of the `compatible_mtu_on` expressions.

- `interface` is passed as a parameter of `underlay_mtu`

Note that when calling a parameterized expression, we don't need to have a name as a parameter, it can be an expression as well. For instance, we could have written `underlay_mtu[Uppercase(interface)]`.

In some cases, we will want to apply the same expression to each element of a list of paramters. For instance, assuming that the expression `compatible_mtu_on` raises a symptom in case of incompatibility, we might want to check that expression on each interface in a given list, say all interfaces that are egress interfaces for a particular route.

This is done by writing:

```
Combine({compatible_mtu_on[egress_interfaces]}`
```

The meaning of the above expression is:

- evaluate egress_interfaces (should evaluate to a list) (For instance `[Loopback0, Loobback1]`)
- for each element, build the expression compatible_mtu_on with the value of the element as parameter, here we would build `compatible_mtu_on[Loopback0]` and `compatible_mtu_on[Loopback1]`
- pass the results of all expressions to combine for aggregating them.
- monitor for changes in `egress_interfaces` and update as the list changes

Convention

A possible way to organize the expressions section is to decompose the service component. For instance, DeviceHealthy can be divided into cpu_healthy, memory_healthy, storage_healthy ... For each subexpresssion, include a "Measure" block will all the metrics (i.e., relative to CPU) followed by a "Compute" block with all the expressions needed to assign a value to CPUHealthy.

Finally, the last expression shall combine all expressions of the subparts into a single expression summarizing the value.

External definitions

Operators

The language in itself does not define any operator, they have to be defined as expressions.

Metrics

In the current language, we abstract the definition of a metric to a name and a set of parameters. To actually specify how to retrieve a value for a given device with a given OS, one has to add an entry in the relevant metrics file.

Detailed spec

Generalities

Spaces tabs and new lines are ignored except otherwise specified.

The syntax of the language is compatible with Markdown: i.e., if spaces and new lines are correctly ordered, the file will correctly render in Markdown. However, it is also possible to write syntactically correct file that do not render well in Markdown.

Syntax

For a machine-readable version of the syntax see the ANTLR grammar and lexer.

Here is a human readable version of the syntax. Non terminals are in lower case, terminal are in upper case.

```
Service := header arguments expressions
```

Header

The file starts with a header that defines the name of the Service and a description of the service component.

```
header := '#' ID SUB_DESCRIPTION
```

```
ID : any sequence of letters, numbers or '_'  starting by either a letter or
an '_'

SUB_DESCRIPTION: anything that does not contain '#'
```

The ID terminal is used for every ID in the syntax. The spaces IN the description are kept, and as long as the character '#' is not met, the sequel is considered part of the description. Thus, it is possible to use any markdown except title with "#" in the description.

Arguments

The arguments start with a level-2 Markdown title, followed by a list of a least one argument and optionally some global parameters.

```
arguments :=  '## Arguments' (argument)+  (display_params)?

argument := '*' ID? ID ':' LINE_DESC

LINE_DESC: any string not containing a newline.
```

In the argument syntax, the first ID indicates (optionally) the type and the second one is the actual name of the argument. The LINE_DESC contains a description of the argument, terminated by a new line.

The display parameters indicate how to render the Service in a GUI.

```
display_params :=  'display level' '=' ID  'web_label' '=' web_label

web_label := BACKQ3 TEXT ('{' ID '}' TEXT)* BACKQ3

BACKQ3 : 3 backquotes (```)
TEXT: any string not containing '{'
```

The display level ID should be one of the predefined levels. The web label contains an arbitrary string with some IDs enclosed in braces. The IDs should be arguments declared before. The web label will be formatted using the python 'format' function to replace argument's ID in braces with the value of the argument.

Expressions

The expressions start by a level 2 title 'Expression' and contains a sequence of measurements and computations.

```
expressions := '## Expressions' ( measurements | computations )*
```

Measurements

A set of measurements is introduced by the level 3 title 'Measure' followed by a comment (optional), and then a list of measurement (i.e., metrics) to obtain. As explained above in this document, the metrics have to be defined.

```
measurements := '### Measure' COMMENT measurement+

measurement := '*' ID? ID '=' METRIC_NAME '(' measurement_parameter (',' measurement_parameter)* ')' '_' M_DESCR '_'

measurement_parameter := '-' ID '=' ID

COMMENT : any string not containing '*'
METRIC_NAME: identifiers separated by dots
M_DESCR: any string not containing '_'
```

For each measurement, we have two IDs, the first one, optional, indicates the type of the metrics, the second one indicates the name of the measurement. After the equal sign, the definition of the metric instance to associate to the measurement name is given. For instance `interface.mtu(device=source_device, interface=source_interface)`, where source_device and source_interface are arguments of the service component.

The METRIC NAME should match an existing metric in the metric engine. The parameters name should be existing parameters of that metric.

Finally, the description of the measurement, which should not contain _ is enclosed between _.

Computations

A set of computations is introduced by the level 3 title 'Compute' followed by a comment (optional), and then a list of computations.

```
computations := '### Compute' COMMENT computation+

computation := '*' ID ':' LINE_DESC symptom? expression_decl+

symptom := LEVEL CONDITION: LINE_DESC

LEVEL: 'broken' | 'degraded'

CONDITION: ('if false'|'if true')

expression_decl := '+' '`' expression '`'
```

A computation is defined by:

- a name
- a one-line description
- an (optional) symptom
- at least one expression declaration

The name is defined by the first ID in 'computation'. It should not be used by a previous argument, measurement or computation.

The symptom can only be added if the expression evaluates to a Boolean value. LEVEL and CONDITION indicates the level of the symptom (degraded =~ warning and broken =~ error).

There can be several expression definitions for the same expression name. If so, the first definition in order for which all subexpressions (i.e., references to other expressions, arguments or metrics) are available is taken.

This mechanism allows to have flexibility in the expression as shown above.

The label of the symptom can contain expressions enclosed between backquotes '`'. In that case, the expression is replaced by its value whenever the symptom is raised (the expressions are always evaluated.) As a corollary, the compiler does not allow the expressions used in the symptom labels to depend on a metric that is not already a dependency of all expression alternatives.

For instance:

```
### Measure

 * admin_status: Administrative status of the interface
   [...]
 * errors_count: Number of errors on the interface
   [...]

### Compute

 * interface_healthy: Whether the interface is healthy
   broken if false -> Interface is not up status: `admin_status` or too many
errors `errors_count`
   + `admin_status == "Status UP" and errors_count < 10`
   + `admin_status`
```

will not compile because the symptom label depends on `errors_count` but the last alternative doesn't.

Expression

The expressions have the following syntax:

```
expression := sum (cmp sum)?

cmp := '==' | '<=' | '<' | '>=' | '>'

sum := factor ('+'  factor)*

factor := atom (( '*' | '/') atom)*

atom := ID | INT | FLOAT | STRING | BOOL | '(' expression ')' | call

call := ID '(' expression (',' expression)* ')'
```

This syntax supports expressions using the arithmetic operators '+', '*' and '/', the comparison operators in cmp, identifiers, literals of floats, ints, Booleans and strings, and function calls.

Function calls are used for operators that do not have an infix version. The first ID is the name of the operator, which is checked against expression classes. In particular the number of arguments is checked.

The expression and metrics can be declared in any ordered. However, circular dependencies between expressions are not allowed.

8.2 Appendix B – Examples of service component configuration

Non-limiting examples of service components that network orchestrator may configure include layer 1 (L1), layer 2 (L2), and layer 3 (L3) connections/interfaces, packet routing protocols, logical network overlays such as equal-cost multi-path routing (ECMP), and service components related to traffic shaping. Non-limiting examples of operations employed by network orchestrator to configure the aforementioned example service components, on a network device among network devices, are provided below.

To configure an L1 connection/interface:

a) Enter L1 interface configuration mode.
b) Configure on the network device components and interface parameters, including hardware parameters, memory buffers, optical transmit power, and optical encoding/modulation employed by optical interfaces on the network device.
c) Exit the L1 interface configuration mode.

To configure an L2 connection/interface:

a) Select a type of interface (i.e., L2, virtual LAN (VLAN), port-channel).
b) Enter L2 interface configuration mode.
c) Assign a media access control (MAC) address, a maximum transmission unit (MTU), and an L2 Quality-of-Service (QoS) classification (referred to simply as "QoS").
d) Enable the L2 interface (no shutdown/enable L2 interface command).
e) Exit the L2 interface configuration mode.

To configure an L3 connection/interface:

a) Select a type of interface (i.e., L3).
b) Enter L3 interface configuration mode.

c) Assign an Internet Protocol (IP) address, an L3 MTU, and an L3 QoS.
d) Enable the L3 interface (no shutdown/enable L3 interface command).
e) Exit the L3 interface configuration mode.

To configure a packet routing protocol (e.g., Intermediate System to Intermediate System (ISIS)):

a) Check for pre-requirements of the packet routing protocol:

    i. IP address configured on at least one interface.

    ii. IP routing process running for an address family (e.g., IPv4, IPv6).

b) Enter interface configuration mode for packet routing protocol.
c) Select a routing protocol (e.g., ISIS) and start a routing protocol process on the network device (e.g., router Routing Information Protocol (RIP), router Open Shortest Path First (OSPF)).
d) Assign interfaces to include routing advertisements (selects IP networks for the advertisements).
e) Assign an IP address, an L3 MTU, and an L3 QoS.
f) Exit the interface configuration mode.

To configure ECMP:

b) Identify parallel links or parallel multi-paths and associated network device interfaces for ECMP.
c) Enter ECMP configuration mode.
d) Enter interface or routing configuration mode
    a. Configure equal costs among interfaces identified in step (a) (e.g., configure Routing Information Base (RIB), Forwarding Information Base (FIB) accordingly).
e) Exit the ECMP configuration mode.

To configure traffic shaping as its own Service or as a sub-component of another service component, e.g., an interface:

a) Identify classes of network traffic (e.g., policy-map/class-map).
b) Define shaping, specifying peak/average of traffic, and bursting profile.
c) Enter interface (or permanent virtual circuit (PVC)) configuration mode.
d) Applying the above-defined shaping to an interface.
e) Exit interface configuration.

The service components and operations to configure the service components listed above are provided by way of example, only, and may be modified and/or expanded to include additional service components and operations, as would be appreciated by one of ordinary skill in the relevant arts having read the present specification.