Service Oriented Network Operations using Abstraction and Automation

Document History

Versio n	Date Updated	Updated By	Comments
0.1	Sept 2015	James Bensley	First draft
0.2	Aug 2016	James Bensley	Updated glossary

Document Distribution

Author contact: jwbensley@gmail.com.

Contents

Glossary

API	Application Programming Interface
BoM	Bill of Materials
CDN	Content Delivery Network
CLI	Command Line Interface
DoA	Dead on Arrival
EoL	End of Life
EoS	End of Sale
IPAM	IP Address Manager
ISP	Internet Service Provider
MSP	Managed Service Provider
NOC	Network Operations Centre
NFV	Network Function Virtualisation
PEW	Planned Engineering Works
SLA	Service Level Agreement
SDN	Software Defined Networking
TISP	Telephony Internet Service Provider
VLAN	Virtual Local Area Network

1. Introduction

1.1 Purpose of This Document

This document aims to inform the reader about the potentially huge resource savings a company can achieve if it migrates to an agile and automated operating paradigm. Such a company would have its operations and products heavily focused on networking like an ISP/TISP, MSP, CDN, colocation provider, cloud platform provider etc. The reason the benefits can be so large with this paradigm shift is because the traditional paradigm of network operators has been to optimise operations reactively instead of proactively and to set operational goals that are just about achievable and then to simply meet them, but rarely exceed them or continually increase them.

By the end of this document the reader should have a fair understanding of operational improvements that can be achieved, the methods that can be used to deliver those improvements, and the technologies available which underpin the methods.

SDN and NFV are out of scope of this document. This document is about network programmability, operational abstraction and agile network operations. This are the building blocks that can in turn be used to provide SDN and NFV services as a separate exercise.

1.2 Background to the Problem

The most widely performed tasks within the networking world are manual tasks which are laboriously repeated again and again. Examples include writing devices configurations by hand or applying changes to devices on the CLI. Many networking companies have groups of employees whose sole job function is to write configuration files in a text editor or raise changes and apply device configurations. A scarily high number of wealthy and successful companies perform every step of the service delivery process from ordering through to operational handover manually. Many networking providers scale out their operations horizontally; “employing more people allows us to deliver more services” is a linear methodology offering no efficiency gains at all.

Many networking providers will reach a point when a simple linear approach to scaling is no longer financially sustainable; the constant drive to lower customer facing prices means that the ratio of “requirement vs. effort” must also decline. As an example; a 10:1 ratio of demand to delivery effort (10 services delivered per month by 1 engineer) assuming a suitable operating profit, will likely not be acceptable 24 months later when the demand has double meaning the delivery effort required has doubled too. Moving from 10 engineers delivering 100 services a month to 20 engineers delivering 200 services a month, whilst continually reducing customer facing prices and without losing operating profits is difficult. The target for network operation innovation should be to shift the ratio of requirement vs. effort from 10:1 to 100:1, to 1000:1, whilst increasing profitability and reducing customer facing pricing.

For a typical networking operator today trying to reduce operational costs without losing revenue and scale out operations, they will face barriers in efficiency similar to the following;

- Linear resource scaling (e.g. throwing more engineering resource at the problem)
- Tracking operational data manually by hand (e.g. capacity planning)
- Operational data accuracy (i.e. which customers are using this service?)
- Long lead times on service delivery (i.e. streamlining recurring tasks)
- Inconsistent service delivery (e.g. engineers/project manager’s unique working style)

The inefficient operations of networks today aren’t solely a technical issue that can be automated away, such as making changes to network devices automatically instead of by hand, or using an IPAM to allocation IP addresses and VLANs automatically instead of by hand. It is implicitly tied into the business operations and procedures that dictate how the network is managed.

Legacy procedures involving change control management, provisioning processes, resource tracking etc. all need to be tired into the overall shift to automation. This is a complete paradigm shift, not just a technical shift to automation; for most businesses different departments within the same business operate using a totally different paradigm. For example; the change board maybe focused on service impact risk and completely disconnected from compliance and auditing, delaying an urgent security upgrade to keep one customer happy whilst risking the service of many more customers. Equally a NOC maybe focusing on monitoring and recording as much data as possible to better capture network events as they happen, but not actually correlating them to the underlying service, carrier, supplier, deployment etc. One unified service oriented paradigm shift is required by all BUs to achieve the maximum efficiency improvement.

2. Operational Abstraction and Automation as the Solution

2.1 A Single Source of Truth

Automation of network operations depends on the business having well defined service models (exactly what their products are, the required components, their SLAs, expected performance, any restrictions to these services etc). Well defined service models should be backed up with well defined procedures that define the delivery, maintenance, and decommissioning of a service.

Service templates can be created which capture the required data to deliver a service (IPs, VLANs, circuit type, speed, etc) and link that data with a service model and a procedure. This allows the delivery teams to operate a “cookie-cutter” deployment process (meaning rapid, consistent and reliable service delivery).

The service planning and delivery life cycle usually starts when an order is accepted from a customer. This means that from the point of sale acceptance what has essentially been sold are service catalogue items that infer a set of service templates that need to be executed (a list of predefined service models, the data required for those models, and an accompanying list of which procedures must be completed to fulfil the service live criteria for those services sold).

A sales record and any data required for the service delivery should ideally be stored in one central system. Any system which is to efficiently automate network operations has one central place to lookup and update with regards to all this information. This central data storage location becomes the single source of trough for all network operations.

2.2 Implementing Abstraction

By creating well defined layers of abstraction in network operations a service oriented operational paradigm can be implemented. Having a service oriented paradigm built on layers of abstraction provides a more flexible workflow which allows for greater operating efficiency. This is achieved by disaggregating operational dependencies from service deliverables; for example, configuring customer connectivity on device using a vendor agnostic system means that any engineer can perform this task without prior knowledge of that device.

The most optimal implementation method in terms of efficiency to support a service oriented paradigm, is to have a programmatic network that uses software to automate its operations. Using software to automate all operations provides the highest levels of efficiency and data accuracy with the lowest level of overhead.

To have a programmatic network means that the network devices are configured and monitored by software processes (preferably through an API but also via the CLI) not manually by humans. From a technical perspective a common method for achieving this is to treat the network device configuration like software code and use similar tools that software developers use to manage code to manage device configuration.

2.3 Business Drivers for Abstraction and Automation

The operating costs of the business can be reduced through the following improvements if implemented with net-ops automation:

Reducing OpEx:

- Reduce change and maintenance planning overhead: automated service impact assessments can be generated which automatically seek peer review and change management approval from the required parties when changes are submitted.

- Reduce communications overhead: when changes are approved or supplier maintenance is scheduled automatic change/PEW notifications can be dispatched.

- Reduce failed change frequency: each change that is scheduled and approved can be automatically tested in a sandbox and rejected if the tests fail.

- Reduce overtime/out-of-hours work: network changes can be scheduled to run in advance of the required delivery date and unattended.

- Reduce self inflicted disruption: when changes are performed automatically tests to measure the change success can be automated with pass and fail criteria that can result in an automated rollback.

- Reduce SLA breaches: automatic tests for each element of a service can be scheduling to run periodically for an indefinite period. When a testing threshold is breached the required recovery actions can be applied automatically.

- Reduce new hardware/software onboarding: testing of new vendor hardware or software features can be performed automatically to reduce the time required to approve upgrades or new deployments.

- Reduce unexpected opex: a real time estate inventory can be used to produce reports on devices reaching the end of their supplier support contract, devices nearing vendor EoS/EoL announcements and audit against security and bug releases.

Reducing CapEx:

- Reduce vendor software lock-in: by abstracting the control plane and management plane of network devices a vendor agnostic management and monitoring approach can be taken meaning vendor specific tooling is no longer required.

- Reduce vendor hardware lock-in: by abstracting the control and management plane of network devices any vendor can be used which provides the required features for the business at the best price point (there is no need to factor in staff training for specific vendor technologies or learning new vendor configuration syntax).

In addition to reducing costs, by standardising as many operational processes as possible and automating them to remove any variance in the execution of those processes, the exact costs of each operation can be quantified to provide precise forecasting.

Automation is primarily built upon a company having clearly defined operating processes and having all company's operating data stored in a central authoritative system (the "Single Source of Truth"). By taking a company through the process of migrating to an agile and automated operating model the following improvements are achieved:

- Creating a uniform service catalog; to automate service delivery the company must know exactly what services they have sold and are currently selling, a service audit helps to record any bespoke services and reduce the service catalogue deviation.

- Delivering assured services: to deliver any service automatically at scale the service must be defined in detail including any constituent service components so that it can be templated and repeatedly delivered with minimal variation in provisioning and operation. This increases technical consistency (device configuration, naming conventions, software versions etc.) which guarantees that the services being automatically delivered can be supported.

- Reduce variance in process execution; by standardising and automating operational processes to provide deterministic results there is a reduction in variance when projecting deadlines or providing customer quotations and adhering to them.

- Service compatibility; abstraction allows for devices to be queried in a vendor-agnostic manner to ensure they support a proposed new service or change to an existing service, before that action is processed. This also ensures at the point of sale that services are being sold where they can definately be delivered. If there is gap between the current infrastructure and the required infrastructure to deliver a new service this can be exactly identified (and reported in the form of a BoM for example).

- Deliver direct to service handover; by using a templated service delivery process and a templated service testing process, new services can provisioned, tested and brought into operational acceptance for customer handover all in a single automated action. This reduces DoA service deliveries and brings the service acceptance testing coverage closer to 100%.

- Accurate service impact assessments: by automating changes funnelled through a central source of truth, precise service impact assessments can be generated using real-time service deployment stats for planned engineering and change works.

- De-risk changes: network maintenance operations and BAU activities that are templated and automated can de-risk the human error factor. In addition to this automatic staging of changes can more accurately gauge the expected impact of a change.

Network abstraction and programmability can also be used to achieve the following technical advantages:

- Supportability; by only delivering templated services and through a central provisioning system there is a reduction in bespoke and poorly documented solutions meaning the supportability of the live service estate moves closer to 100%.

- Mean time to deliver; automating each step of the provisioning process means (near-)zero touch provisioning can be implemented to scale up service delivery with a reduce overhead.

- Reduce time to market; agile operational processes that use automation and abstraction reduce the time it takes to bring a new service to market by supporting the additional of a new service without needing to extend the existing framework or any processes.

- Mean time to fix; by defining structured services the troubleshooting process can be refined and automated to provide rapid fault detection and resolution against externally sourced impacts to service.

- Mean time to recover; by abstracting device configuration from operating state (off-device if required), automatic configuration checks and rollbacks can be implemented on devices with no native support, reducing the time to recover from internally sourced service impacts.

- Mean time between errors: by abstracting network device configuration to a single vendor agnostic format built from service templates, there is no need to write complex multi-vendor device configurations which increase the risk of mistakes. In addition to this devices that do not support candidate configurations and syntax checking natively, this can be achieved off-device.

- On-time capacity planning: by using a central authoritative system to deploy and manage network services (near-)real time capacity planning can be achieved to more accurately manage capacity and failure requirements.

- Guarantee network state: changes made through automation using transactional change management reduce the level of unknown state in the network ensuring the “single source of truth” is accurate.

- Recurring events: network change actions such as ACL updates or peering updates can be scheduled to run repeatedly and unattended.

- Reduce resource constraints: customer orders or BAU changes can be scheduled for automatic unattended completion at times that satisfy customer maintenance agreements or operational “quiet times” without the need for available human resource or CAB.

Strategy and governance of the network can be formulated by using network abstraction and automation to gather operational intelligence:

- Continual accreditation compliance: configuration reports can be generated automatically on a recurring schedule to check devices for compliance breaches. Any remedial actions can also be implemented automatically.

- Business performance: metrics of operations can be automatically created, gathered and analysed. These can be used to report on service performance or find imperfections in the order-to-handover cycle and assist in setting and meeting realistic improvement targets.

- Customer and supplier integration: having a high level of data accuracy in an organised hierarchy means that both customers and suppliers can integrate by exposing an API to them.

3 Methodology Overview

It is outside the scope of this document however it is implied that a clear service catalogue of network services exists and the data requirements specific to the business to deliver those service catalogue items are already known.

3.1 Resource Data Management

It must be known in advance what data is required for the day to day operations of the network and what the upper and lower bound limits of those data sets are. This allows for a central system to be designed and implemented which can be used to store, access and update the data in real-time and provide accurate capacity analysis of all assets and resources.

Even when using tightly controlled processes it can still be difficult to ensure that the data stored regarding logical and physical resource usage and assignment is accurate, current and not contributing to data duplication. Probably the most ideal approach is the creation of a schema unique to the business that supports its operational data requirements which is then updated by indefinitely running automated task which scans the live estate and resource database and correcting any discrepancies.

Below are some examples of the required data sets the logical and physical resources manager could track:

- Resources (IPAM): Ideally in any modern network this should include many resources beyond basic IP addresses, any logical resource with a finite availability that needs to be assigned for a service needs to be checked for availability and then reserved. These are typical global logical resources, for example;
    - AS Numbers
    - IPv4 & IPv6 addresses
    - VRFs (potentially using the Assigned Number subfield as the key)
    - Route Targets
    - VLANs (QinQinQ... support)
    - Bridge/Broadcast Domains
    - EVC / EFP ID
    - VFI / VSI / EVI ID (for VPLS / EVPN / PBB)
    - Virtual Circuit IDs (pseudowire IDs)
    - Static MPLS Labels
    - Global SIDs (for Segment Routing)
    - Logical interface IDs (tunnel IDs / subinterface number)
    - Frequency / wavelength

- Assets: As with the “IPAM” any physical resource that needs to be reserved must be accounted for so that availability of physical resources can be tracked and forecasted. It might make sense to track these physical resources separately to the logical resources above as these likely change less frequently or reach capacity as quickly as logical resources (devices are rarely deployed at 90% capacity for example).

  Ideally this would also include logical resources that are platform dependant and not global;
  - Chassis slots / MPCs / MPAs / SPAs / expansion modules
  - Physical ports / plugables / patch panel ports / PDU sockets
  - Rackspace / floorspace
  - Power / UPS capacity
  - Airflow

3.2 State Management

The operating state of a network device is usually derived from configurational state data being stored on it. By abstracting vendor specific configuration syntax into a vendor neutral format the configurational data for the network estate can be automatically and dynamically generated by data that is pushed/pulled from central BSS/OSS systems. This can then be translated to the a vendor specific syntax if required, before being applied to a device.

No operational state changes should be made without configurational state changes to infer that operational state. By having a separation between configuration state data and operating state data any proposed operational state change which will be derived from new configuration data can first be validated before being implemented. Operating state can be changed without a config change though. If this introduces a problem or the desired operating state is simply not achieved, then there might not be an audit trail of how the current operating state was reached or what needs to be reversed to revert back to desired operating state.

For example an ordering system API can feed data into an operational API to generate device configuration state. In this case the central BSS systems could provide the central source of truth for the network operations. Device configurational state derived from the BSS systems can be stored in a central atomic change database. Configuration data in this central store can be compared against the running configuration of network devices and pushed to the running configuration if discrepancies are found (a new “service” is added to the device). Finally the operational state of devices can be checked to confirm if the desired operating state has been implemented based on the configuration data applied to the device (ultimately changes are being sourced from the BSS system in this example which might not be ideal).

A finite state machine can be used to manage the process of configuration state updates, publications and implementations. This can be coupled with a tight control loop used to move through state machine but also allow the FSM to be restarted and resume from where it left off without issue (meaning that network configuration change events idempotent and the device stats are atomic).

As an example of this, if the BSS/OSS systems provide the central south of truth for the network estate and a change in the central system is made “provision interface X on device Y for customer Z”, this could result in a push to an automation engine API. The engine could then update a centrally stored vendor neutral configuration file in an atomic database using the information parsed from the BSS/OSS system. The configuration changes (either added due to a new interface being used, or altered due to using an preconfigured interface) have the intention of changing the devices operational state to that which matches service catalogue item N (ideally templates exist for all service catalogue items so that the outcome is predictable). This means the central configuration file is build from the BSS/OSS systems passing service specific details to the engine which can fill out service catalogue templates creating a whole device configuration file that is simply a collection of service templates. The central automation engine can then either push the configuration to the device in a vendor neutral form if supported or use conversation templates if required to produce vendor specific configuration.

When a change is made to the configuration of a device the change should idempotent. Continuing the example of “provisioning interface X on device Y”, generating the configuration for interface X and pushing the configuration to device Y might work for an interface in a default state or known prior state and the generated configuration will transition the interface to the desired operational state. If the interface state is unknown or transitioning from a non-default state either the current running interface configuration must be factored in and the difference between the current configuration state and desired state must be used to produce a transitional config snippet to be applied, or the config generated to provision interface X must replace the existing configuration on the interface in its entirety. In the case that the currently running configuration is factored in this would ideally come not from the live device but from the central configuration database. By always altering this configuration and pushing it to a device, from this central location the entire estate configuration can be searched, updated and validated centrally. This would require that changes made directly to devices are disabled so that all configuration changes come from the central system only so that it is always an accurate reflection of the entire estate operations.

If the method of pulling the running configuration from the device is used to then generate a transitional configuration some logical is required that is aware of vendor specific configuration syntax for each vendor deployed on the network, to calculate the required transition. In the case a vendor agnostic configuration syntax is being used then “only” a single interpretation function would be required (hopefully, there is the possibility that different device generations of devices will use different interpretations of the same vendor agnostic syntax). If the central configuration database is updated the entire device configuration can then be replaced and all that should have changed is the configuration for interface X. A diff can be generate to check this. By pushing the entire device configuration using a replace and not a merge operation the device state is enforced and the change becomes idempotent. This can also be wrapped into an atomic change action to ensure it does happen.

3.3 Abstracting Network State

To create a programmable network the configurational state of the network must be abstracted and preferably centralised. Network configuration can be split into phases:

1. The intended state of the network can be derived from service templates after they have been populated with the service variables from the BSS systems. This is then pushed to the network devices to actualise the desired services.

2. The operational state of a network device must then be verified against the intended configuration to check for discrepancies.

Some device vendors support vendor agnostic configuration formats and some do not. Also some devices support different methods of device configuration. In order to provide a method of creating intended device configuration that is agnostic of any vendor specific syntax, and to apply and verify that configuration over a variety of technologies, abstracting those processes through translation to a neutral format and action set allows for them to be simplified and standardised for programmability.

CHECKING: This can happen in two main methods. Firstly the proposed configuration changes can be applied to a staging environment so that the operational changes inferred can be checked that they match the desired operational state changes that lead to the new configuration data being generated. This provides explicit test of the proposed config changes. Secondly a syntax check can be performed. By applying the configuration to a staging environment the syntax of the change is checked (not just any vendor specific syntax such as CLI commands but also that the data supplied to correct, no IPv4 address with decimal 300 in an octet it or a VLAN ID over 4096 etc). By using some more structure data models such as YANG modules the configurational data can also be validated to the syntactically correct before it is sent to the production devices without requiring a staging device for every device type deployed on the network.

3.4 Abstracting Network Telemetry

After a network devices has been correctly configured to deliver a service the device and service should now be monitored. As with network configuration the monitoring and telemetry of a network device or service can be abstracted to provide a standardised method that is vendor agnostic and programmable.

4 Technology Overview

The following series of tables provide an overview of the different technologies currently available the can be used to provide the methods described in Section 3.

4.1 Device Management Protocols

System/ Vendor	**Min[*1] Version for YANG**	**Config and State Protocols and Transport Methods**	**Notes**
Alcatel Lucent		● CLI over SSH ● SNMPv3 ● XML over NETCONF over SSH	
Brocade		● CLI over SSH ● SNMPv3 ● XML over NETCONF over SSH	
IOS-XR (Cisco)	6.0.0	● CLI over SSH ● SNMPv3 ● XML over NETCONF over SSH ● XML over RESTCONF over HTTPS/1.1 ● JSON over RESTCONF over HTTPS/1.1 ● JSON over gRPC over HTTPS/2 ● CLI over gRPC over HTTPS/2	ASR9000s and NCS5000s: Actually YANG support started in 5.3.0 however 6.0.0 supports 150+ Cisco YANG models, pre- 6.0 the support is poor, 6.1.1 brings OpenConfig support.
IOS-XE (Cisco)	16.4	● CLI over SSH ● SNMPv3 ● XML over RESTCONF ● XML over REST API on CRV1000V only?I	Starts in 16.3 but 16.4 covers more devices.
IOS (Cisco)	?	● CLI over SSH ● SNMPv3 ● XML over NETCONF over SSH ● XML over NETCONF over BEEP	Manual testing on a 1941 running 15.3M shows some basic support is present for Cisco YANG models but no documentation online and it was very buggy.
NX-OS	7.x	● CLI over SSH	This is for Nexus 9Ks and

(Cisco)		● SNMPv3 ● XML over NETCONF over SSH ● XML over RESTCONF over HTTPS/1.1 ● JSON over RESTCONF over HTTPS/1.1 ● GPB over gRPC over HTTPS/2	3Ks, the 5Ks and 7Ks are still pending.
Junos (Juniper)	16.1	● CLI over SSH ● SNMPv3 ● XML RPC over NETCONF over SSH ● JSON over REST API	Actually YANG support started 14.1 but configuration only until 16.1: http://www.juniper.net/documentation/en_US/junos16.1/topics/task/operational/netconf-yang-module-obtaining-and-importing.html 15.1 for REST API support.

*1 NETCONF has been supported on many platform before vendors started producing their vendor specific YANG models. So this is minimum version that supports both NETCONF and YANG.

NMS and Streaming Telemetry

Operations Feature		**Transport**	**Consumer**
Streaming Telemetry		GPB over UDP	
		GBP over gRPC over HTTPS/2	
		JSON compressed over TCP	

Device Vendor	**Southbound API Driver**	**RPC Method**	**Transport**	**Homepage**	**Vendor Support**	**Notes**
Any	NCClient	XML	NETCONF	https://github.com/ncclient/ncclient	3rd Party Open Source	Vendor agnostic XML RPC

						over NETCONF
Juniper (Junos)	pyEz	XML	NETCONF	https://github.com/Juniper/py-junos-eznc	Juniper	Vendor specific Python classes
Cisco IOS-XR	iosxr-eznc	XML	NETCONF	https://github.com/mirceaulinic/iosxr-eznc and https://github.com/mirceaulinic/napalm-iosxr-rpc	3rd Party Open Source	Still under development, not usabel yet, ncclient in the background
Cisco IOS-XR	pyIOSXR	CLI	SSH	https://github.com/fooelisa/pyiosxr	3rd Party Open Source	Wrapper for CLI access, can send individual XML RCP reuests, uses Netmiko for transport
Cisco IOS	Netmiko	CLI	SSH, Telnet	https://github.com/ktbyers/netmiko	3rd Party Open Source	Wrapper for CLI, Paramiko in the background
		RESTConf	HTTP/S			
			gRPC			

Data Libraries

Application/Library	Description	Homepage	Vendor Support	Notes
Jinja2	Templating engine for Python	http://jinja.pocoo.org/	3rd Party Open Source	
YAML				Python can natively import YAML files as dictionaries
Pyang	Validates YANG models and converts them to tree/XML/JSON/etc	https://github.com/mbj4668/pyang	3rd Party Open Source	
PyangBind	Extends Pyang to generate Python classes from YANG models	https://github.com/robshakir/pyangbind	3rd Party Open Source	
YDK-gen	Generates an API from YANG models (using Pyang)	https://github.com/CiscoDevNet/ydk-gen	Cisco	
YDK-Py	Generates Python classes from YDK-gen API output	https://github.com/CiscoDevNet/ydk-py Samples: https://github.com/CiscoDevNet/ydk-py-samples	Cisco	Provide client side validation of configuration against the YANG model
Cisco YANG Models (IOS-XR/IOS-XE/NX-OS)		https://github.com/YangModels/yang/tree/master/vendor/cisco		

UI YANG Browsers:
https://github.com/CiscoDevNet/yang-explorer
https://github.com/CiscoDevNet/yangman

Add references for…

OpenContrail
OpenDaylight Controller

IPAMs;
NIPAP
NetBox
NSoT https://github.com/dropbox/nsot
phpIPAM

Git: maintaining and enforcing state drift in an idempotent way
GitLab

Maintenance portal