SCTE CABLE-TEC EXPO 22
SEPTEMBER 19-22 • PHILADELPHIA, PA

2022 Fall Technical Forum
SCTE • CABLELABS • NCTA

Challenges, Considerations, and Best Practices for Secure SD-WAN Operationalization for Business Services

A Technical Paper prepared for SCTE by

Xin Huang
Sr. Principal Engineer, Product Development Engineering
Comcast Cable
1800 Bishops Gate Boulevard
Mt. Laurel, NJ 08054
Xin_huang@cable.comcast.com

Joshua Horton
Director, Product Development Engineering
Comcast Cable
1800 Bishops Gate Boulevard
Mt. Laurel, NJ 08054
joshua_horton@cable.comcast.com

Hung Le
Sr. Principal Engineer, Product Development Engineering
Comcast Cable
11951 Freedom Dr., STE 900
Reston, VA 20190
hung_le@comcast.com

Table of Contents

List of Figures

List of Tables

1. Introduction

Network connectivity products such as Software-Defined Wide Area Network (SD-WAN) or cybersecurity are becoming critical enablers of needed connectivity as businesses of all sizes re-configure and consolidate network services and solutions using software-driven and virtualization technologies. Large service providers such as multiple systems operators (MSOs) who want to provide innovative solutions in this space have been developing expertise that can drive customer success. This paper will use insights from real projects to detail the ways in which those wishing to deploy these technologies can be guided by simple principles and industry best practices to kickstart successful networking platform initiatives.

In the past, connectivity providers have been very successful deploying networking gear, operating it at scale, and delivering value by executing well. They may not have created their business engines around a core of software technologies or on large-scale virtualization in quite the same way the largest internet platforms have been driving their businesses. Hardware-based technologies provide very high performance in a reliably fixed and predictable architecture, with key differences being variations in speeds, feeds, protocols, or connectors. The technologies and expertise needed to launch software products, by contrast, can often feature dynamic architectures having unpredictable variations, needing data-driven insights to manage.

Organizations with different strengths/expertise who now wish to adopt technologies that have grown up in the era of large internet platforms must become skilled in techniques tied to the software-based infrastructure which brought those platforms to life. Comcast's launch of software-driven networking services could be considered as one such case study. Luckily, many of the lessons that were learned were related to a few fundamental software best practices, which are very well-documented and to which all modern practitioners should already have access.

In this paper, we will share a few key challenges and lessons learnt through real projects and detail the ways in which those wishing to move ahead in deploying networking software at scale can be guided to successfully kickstart similar products and platform initiatives.

2. Challenges and Solutions

Software-based network services such as software-defined networking/network function virtualization (SDN/NFV) connectivity approaches involve abstracting network functions and services out of silicon and into software, separating the connectivity service into administration plane, control plane, and data plane vectors, each of which is coordinated and controlled via software components operated as a platform.

The data plane is transported physically over white-box devices with all the logic for routing and services instantiated across the platform. Control plane services allow update to the configurations and changing the behaviors of the data plane, including adding software-based network functions, such as firewall, traffic steering, or anti-virus in line with packet processing (in a process called "service chaining"). Customers and operators manage the control plane services via the admin plane, exposed via application programming interface (API) or graphical user interface (GUI) into complex orchestration software.

In hardware-based network services, all aspects of the service are well-defined and fixed into the design of the devices being deployed. The ways in which the devices can be configured is therefore more or less pre-determined by the vendors. If more capacity is needed, then new hardware is purchased and deployed.

In software-based network services, all aspects of the service might be distributed across multiple software components, each of which could be instantiated on a variety of different hardware options. Each choice in architecture and configuration results in a potentially wide range of capabilities and trade-offs that must be evaluated and carefully calibrated. Different layers of software abstraction, of operating systems and virtualization layers, and of interoperability between the layers, creates combinatorial numbers of variations which could affect the behaviors of the customer service.

Embracing this complexity and developing techniques to make the problems tractable and in line with the strategies for being handled within hardware was a core part of the challenge in successfully deploying a software-based networking product.

Below is a list of high-level principles we embrace in design and operations in order to resolve the above challenges:

- Keep it simple: Leverage **cloud-native architecture** and **standard technologies** like edge routers, border gateway protocol (BGP), generic routing encapsulation (GRE) tunneling, proxies, and load balancers (LB) for system integration.
- Emphasis on standardization of configuration in version-control, combined w/ logical inventory in change management database (CMDB) plus strict change control policies to facilitate **automation-first deployment, move/add/change/delete (MACD), & disaster recovery (DR)** for operations to reduce unforced errors
- Embrace **test-driven development** using fully-automated unit and integration tests to ensure version-after-version quality consistency
- Forwarding to data lake, aggregation of time-series data combined with intelligent machine learning, to achieve **observability and data-driven capacity planning**

3. Keep-It-Simple

When introducing the SD-WAN product, our main goal is to integrate vendor solutions seamlessly with our existing eco-systems and business strategies.

Nowadays cloud infrastructure virtualization has become a dominating technology because of the set of benefits it brings. These include a wide range of hardware selections, improved economies of scale, reduced costs to resource efficiencies, operational flexibility, and faster time-to-market, etc. To keep our product competitive in the market, we embraced the “keep-it-simple” design principle and make use of industry standard technologies and best practices. For example, we adopted cloud-native architecture design, deployed our platform services in geographical-redundant data centers (DCs), and utilized standard networking technologies to facilitate communications between DCs and to the Internet. Figure 1 below illustrates the high-level architecture design of our cloud-based SD-WAN platform.

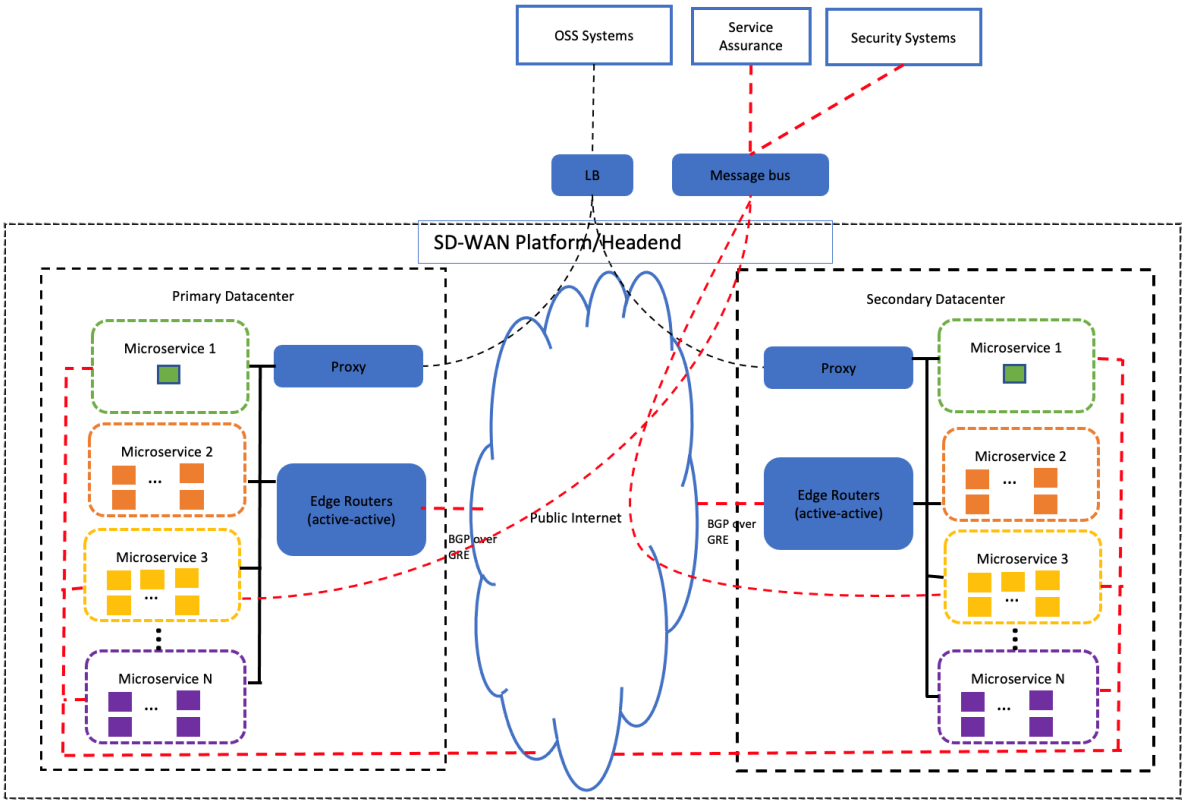

Figure 1 – SD-WAN Platform Architecture Design

Microservices are the core of cloud-native architecture design. The complex SD-WAN control plane and management plane functionality is broken down into multiple microservices, each of which serves a specific function and could scale in/out independently based on its workload. We take advantage of microservices because they support DevOps, and improve scalability, while also allowing flexibility with respect to infrastructure growth. Within the same data center, microservices are interconnected with one another via traditional technologies, e.g., application programming interfaces (APIs), load balancers (LBs), etc.

To achieve SD-WAN platform high-availability (HA) and guarantee business continuity, we deploy our platform (including microservices and data) across geographically diverse DCs (i.e., located in different regions of the country). This geographical redundancy approach is an industry standard best practice that provides business resiliency against natural disasters and catastrophic events which might bring a DC down for certain period of time. Even when disaster happens and one of the DCs is down, our platform remains available since services are still running in the other DC. Once the impacted DC is recovered, everything returns to normal. Different microservices in Figure 1 have different HA designs (e.g., active-backup, active-active, or cluster-based) depending on the nature of the functionality and requirements.

To keep the design simple but efficient, we also adopted standard network technologies to facilitate the inter-connectivity between data centers and the communication between the SD-WAN platform and applications/devices from the Internet. As shown in Figure 1, edge routers and Border Gateway Protocol (BGP) over Generic Routing Encapsulation (GRE) are used to provide HA and dynamic traffic steering for components to communicate with each other between DCs. Standard proxy and load balancer

technologies are adopted to facilitate the communication between upstream systems and components in our platform. Message buses are used to distribute platform telemetry data to service assurance systems and security monitoring systems.

With these high-level design principles in mind, we keep refining and optimizing our platform architecture to make it more scalable. For example, we observed that breaking down big microservices into smaller microservices is an effective way to reduce per-platform cloud footprint. Figure 2 and Table 1 show that our optimization could successfully reduce the per-platform cloud footprint by 9%, 9%, and 62%, respectively, for the best case scenario, the average case scenario, and the worst case scenario. This benefit grows with the platform capacity. When the platform capacity doubles, per-platform cloud footprint could be further reduced by19%, 19%, and 82%, respectively, for the best case scenario, the average case scenario, and the worst case scenario.

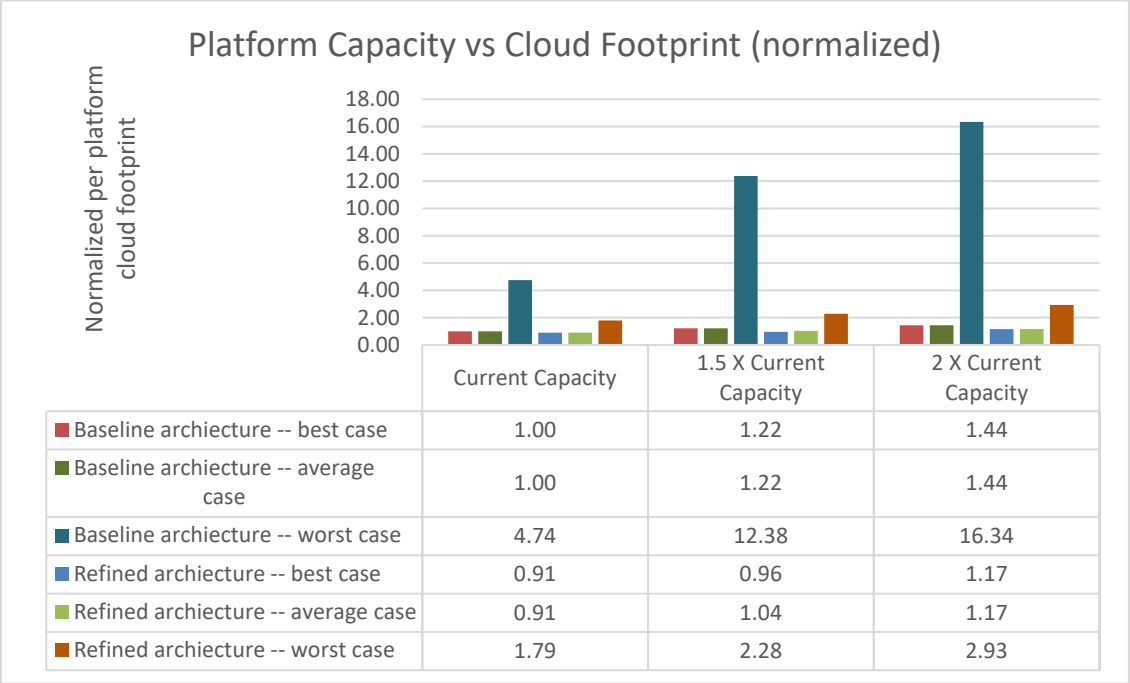

Figure 2 – SD-WAN Platform Architecture Optimization

Table 1 – Per Platform Cloud Footprint Reduction for Different Scenarios

Scenarios	Current Capacity	1.5 X Current Capacity	2 X Current Capacity
Best case	9%	21%	19%
Average case	9%	15%	19%
Worst case	62%	82%	82%

4. Automation-First Lifecycle and Test-Driven Development

When silicon-based network capabilities get implemented as software distributed across multiple components, whether or not on a cloud platform, the pattern is still microservices, and each software component provides a subset of overall system functionality. An orchestration function is then frequently placed between the components to coordinate the components into more advanced processes, such as enabling a new network feature by updating the customer configuration. In some cases, each microservices component might be a separate software product in itself, with its own behaviors and release schedule. This distributed structure provides maximum flexibility and reuse, and can allow for simplification and different optimizations for the operator, at the possible cost of complexity of implementation.

In our case, service reliability was of paramount concern. Given the wide range of network features and functionality being launched, the tight integration between the administration plane and the control/data plane behaviors demanded a comprehensive validation of all capabilities for backwards compatibility, to prove the proper working of all services before moving any new software code to production. This was a non-negotiable requirement, in order to preserve the confidence of customers and operations that software changes would not be disruptive to their experience. But we soon found that traditional approaches to bench testing would not alone be enough to capture sufficient details regarding the individual health and wellness of each component independently, let alone to build a comprehensive picture of overall service reliability.

While we understood that minimizing any risk of disruption would necessitate comprehensive regression before every significant change, we also knew that the tedious manual testing exercises of the early development phase would not suit the needs of our customers. Our approach shifted towards development of a custom, reconfigurable testing platform, integrated with our continuous integration/continuous deployment (CI/CD) pipeline. This high-level framework is depicted in Figure 3. It resulted in reliably repeatable validation cycles, covering an ever-growing set of test cases across all components. This switch to automated testing added new development in the sense of coding test cases, but eventually test design and coding merged into the same practice. Overall, it cut our testing cycles by multiple orders of magnitude, allowing us the flexibility to increase velocity of deploying the latest code, resulting in faster improvement of reliability and features releases to our customers.

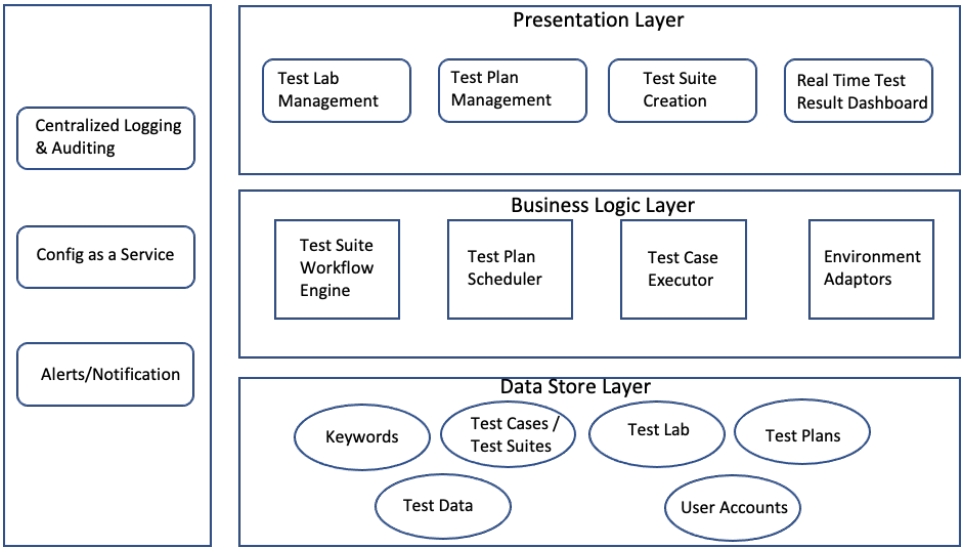

Figure 3 – Test Automation Framework

We also addressed several important challenges. For one, we could not always rely on a fixed set of software versions in production; that is, in the field, due to many different teams operating over time and different field requirements or business realities, there could be different versions of each component running, other than what got initially deployed. All these different variations would need to be supported uniformly from a feature perspective. Further, operations teams maintaining microservices-based systems, having numerous components with different behaviors distributed geographically, would face the tedious exercise of having to manage extreme amounts of detail during maintenance windows, requiring large teams of highly-skilled engineers maintaining superhuman focus for hours at a stretch, attending to every detail when performing upgrades.

The most crucial aspect of successfully managing these details turned out to be perhaps one of the most difficult to achieve in practice: configuration standardization, such that the configurations being tested and deployed have known behaviors that can be used as baselines when our teams are trying to resolve something that isn't behaving as expected in the wild. This is something done very well in software, but very difficult to achieve manually. These system complexities made clear that manual administration of even a small number of environments would be untenable over time. As has been discovered by other software-driven organizations, we resolved that an automation-first strategy was required to make even simple administration tenable.

Another early indicator that tipped the scales towards platform automation was the realization that there would be numerous instantiations of the fundamental datacenter software stacks which powered our service – so many, in fact, as to make manual administration of all those system instances impossible in practice. There could never be enough skilled engineers to manually log in and take care of all the many traditional Day 2 activities which invariably would arise when running complex software systems – password changes, template updates, patches, even disaster recovery. Neither could these systems reliably and repeatably be deployed, day after day, week after week, retaining the same level of quality with the 15th as with the first; nor could they reliably be restored after a disaster using a manual checklist alone.

We thus adopted a platform strategy for our systems lifecycle applications, a technique that is also widespread in the software industry. Starting with a common framework of basic services, such as data, API, GUI, communications, and logging, we ensured that all applications participating in this platform would also enjoy common performance and availability optimizations, such as blue-green deployment for live upgrades, and site-diversity for fault tolerance. On top of this framework was developed a portal, into which bits of functionality could be dropped. Initially just a wrapper for some crude management utilities, it has become the one-stop-shop for platform operations teams, who leverage automation at every step in the lifecycle of our production systems. The portal's extensibility enables it to be used not only for large milestones such as deployment, upgrades, or disaster recovery, but also to perform more routine tasks such as license management, password rotation, and security patches.

The most important benefits from the repeatability and reliability of this standardized approach to operations are clear and have proven value from the start; many serious issues that could typically have resulted from hand-crafted configurations, varying from environment to environment, have been completely avoided. Taken as a whole, the benefits due to the automation are irrefutable, with time-in-motion improvements typically measured in (sometimes multiple) orders of magnitude, as illustrated in Table 2. To paraphrase computing legend Larry Wall, it "makes hard tasks easy, and impossible tasks possible."

Table 2 – Platform Lifecycle Automation Benefits

Platform Lifecycle Category	Execution Timeline Improvement with Automation
Regression Testing	>99%
Production VM Build / Software Deployment	92%
Disaster Recovery / High Availability Testing	>70%

5. Data-Driven Proactive Monitoring

Modern operational visibility has expanded beyond sysadmins and ITOps analysts. It is required not only to monitor the status and performance of running applications/services/systems and to detect issues in real-time but also to understand why, project the trends, and provide feedback to DevOps teams and customers. Additionally, the nature of cloud technology and SD-WAN technology, namely the separation of data plane, control plane, management plane, virtual resource, and physical resources, adds more complexity to the platform monitoring and data analysis.

Following the industry standard, we embrace data-drive proactive monitoring approaches and cross-layer correlation to achieve observability at all layers. It is a straightforward architecture pattern which allows us great flexibility in adding or changing features and service. Figure 4 shows our high-level data-driven monitoring architecture design.

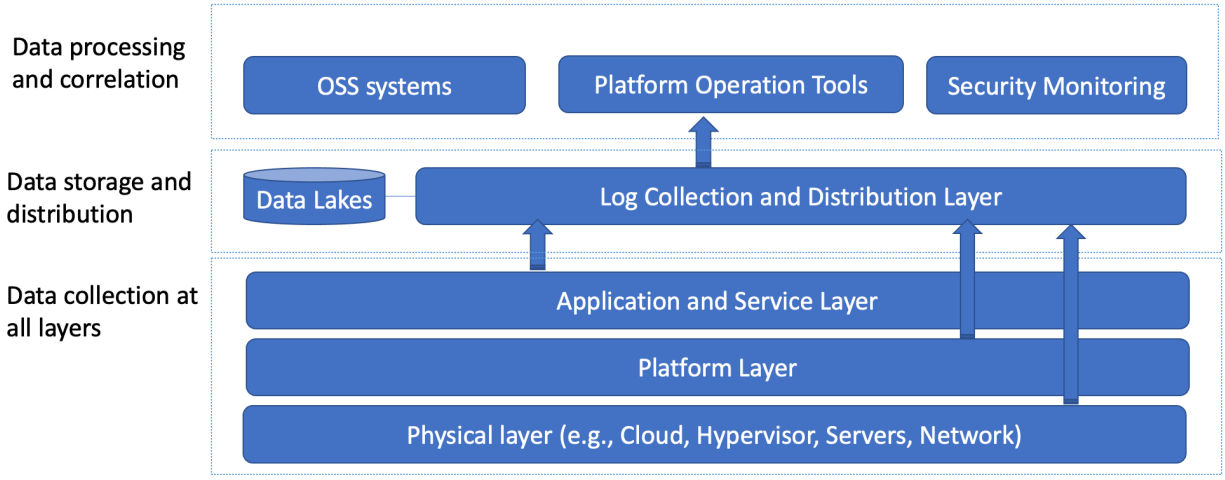

Figure 4 – SD-WAN Platform Data-Driven Monitoring

The data collection layer utilizes industry standard tools or vendor supported features to collect data and provide observability from all layers, including the physical layer (e.g., cloud, hypervisor, servers, hosts, network, etc.), the platform layer, and the application/service layer. The data collected includes all three pillars – logs, metrics, and traces – that are needed for observability.

The data storage and distribution layer uses industry standard technologies and shared platforms to store and distribute telemetry data to the upstream systems. The volume and velocity of the data needed for observability is huge. Thus, our design requirement on systems and platforms used at this layer mainly focuses on scalability, performance, and HA.

The data processing and correlation layer consists of multiple systems that are designed and developed to provide visibility from different perspectives. For example,

- Customer portal: provides the overall health status at the customer service level.
- Operation team tools and portal: provides in-depth health status of all layers from an engineering perspective.
- Security monitoring portal: provides in-depth telemetry data from a security perspective.

Our data-driven monitoring infrastructure has become a critical piece in the entire product ecosystem. It provides insightful information and feedback from many product perspectives, as in Figure 5 (a) below.

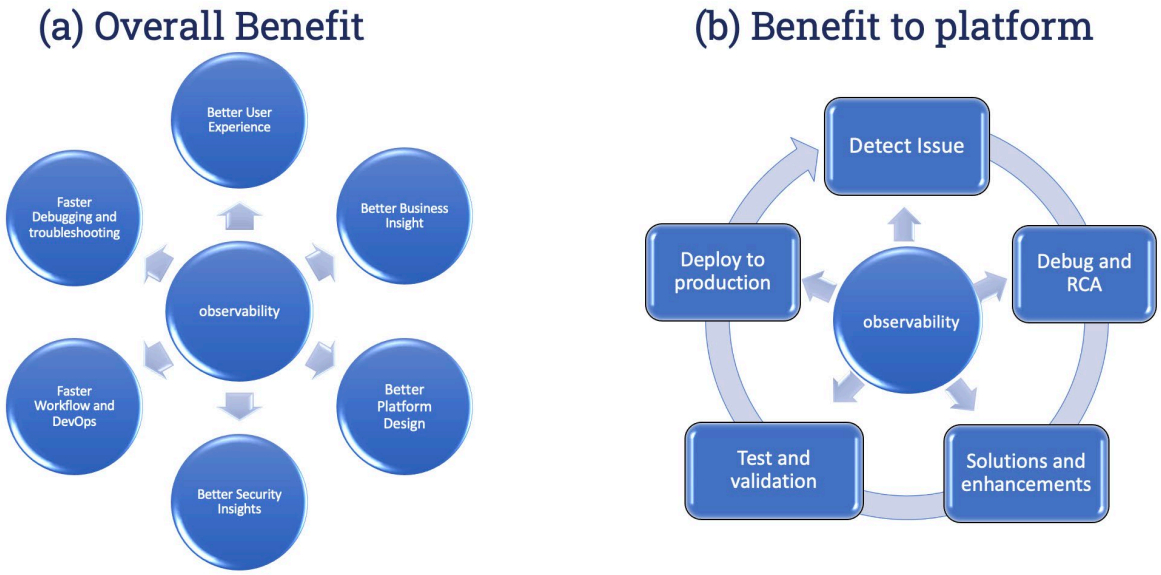

Figure 5 – Benefits of Observability

From a platform perspective the main benefits include but not limited to:

- Fast operation reaction to issues: we are collecting health metrics from all layers and triggering notifications to our operation teams to react. The correlation of collected data from all layers also assists in troubleshooting and debugging process, helping operation teams and platform architecture teams to identify the root causes and fix issues even before customers report them (as illustrated in Figure 5 (b).
- Health metrics collected from the physical layer and the platform layer help us to look for signs that indicate resources may soon run out of capacity, enable us to predict the growth and trend, perform capacity planning, and trigger operation teams to scale out platform components. Figure 6 illustrates the microservices growth projection with increasing platform capacity. The calculation is based on the observability data collected in production. As shown in the diagram, with the increasing platform capacity, different microservices need to be scaled out differently depending on the projected workload. Some microservices do not require to be scaled out even when we plan to double the platform capacity.

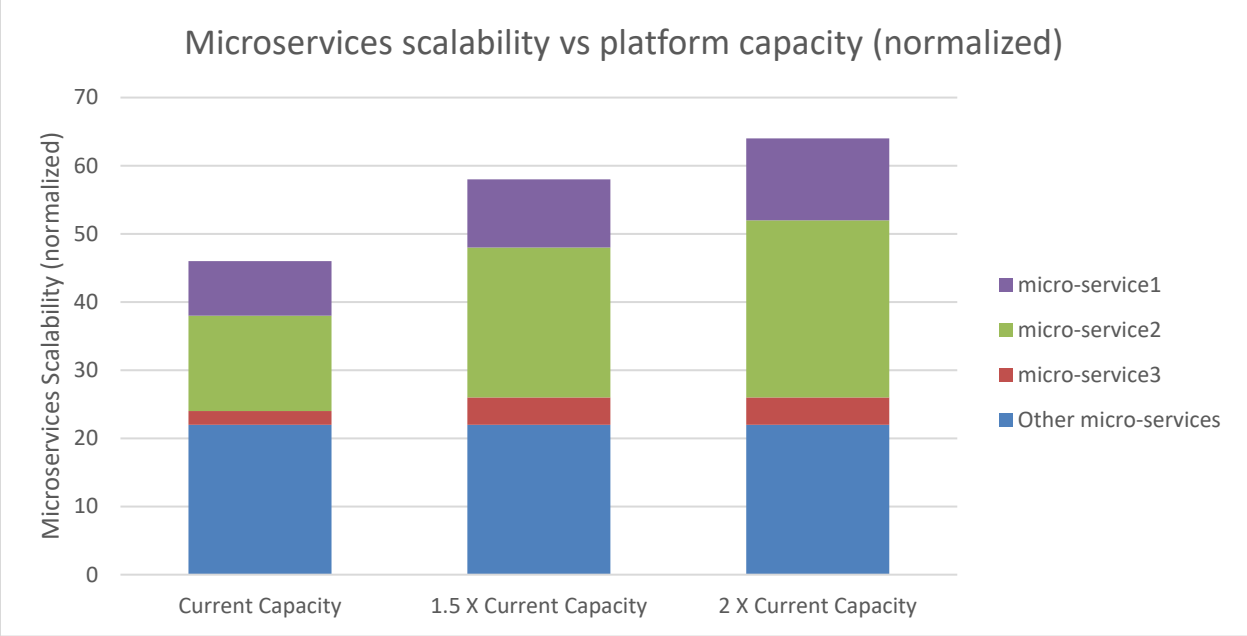

Figure 6 – Microservices Growth vs Platform Capacity

In addition, using standard technologies in design and developing this data-driven proactive monitoring infrastructure helps to reduce development cost, to achieve required scalability and reliability, and to hire talent to maintain and operate the platforms.

6. Conclusions

It was the intention of this paper to detail the ways in which we have found that simple software industry best practices could be implemented to great effect as part of the operationalization of networking services. These include leveraging standard networking protocols for implementing core availability behaviors, standardizing configurations in order to apply an automation-first approach to change management, embracing test-driven development to validate changes as quickly as needed by the business, and employing insights collected using modern data management approaches to forecast growth and anticipate changes. Although common among many industries, these techniques differ from hardware-based approaches due to their inherent flexibility in relation to dynamic virtual and distributed software-based systems, allowing greater reliability and availability to be offered.

Abbreviations

API	application programming interface
BGP	Border Gateway Protocol
CI/CD	continuous integration/continuous deployment
CMDB	change management database
DC	data center
DR	disaster recovery

GRE	Generic Routing Encapsulation
GUI	graphical user interface
HA	high availability
MACD	move/add/change/delete
MSO	multiple system operator
SDN/NFV	software-defined networking/network function virtualization
SD-WAN	software-defined wide area network

Bibliography & References

Programming Perl, 2nd Edition (1996), Tom Christiansen, Randal L. Schwartz, and L. Wall; **ISBN-13:** 978-1565921498, **ISBN-10:** 1565921496; O'Reilly Media.

M. Casado, N. McKeown, and S. Shenker, "From ethane to SDN and beyond", in ACM SIGCOMM Computer Communication Review, vol. 49, issue 5, Oct. 2019, pp 92-95.

L. L. Peterson, C. Cascone, and B.S. Davie, "Software-Defined Networks: A System Approach"; System Approach, LLC.

"Network Functions Virtualisation – Introductory White Paper"; ETSI. Oct. 2012. Retrieved June 2013.
G. Fellows, "High-Performance Client/Server: A Guide to Building and Managing Robust Distributed Systems", in Internet Research, vol. 8, issue 5, Dec. 1998.

Observability Engineering, C. Majors, L. Fong-Jones, and G. Miranda ; ISBN: 9781492076445 ; Released May 2022 ; O'Reilly Media.

Observability in Google Cloud ; Google Cloud DevOps Research and Assessment (DORA) research.

N. Kumar, A. Leventer and A. Matatyaou, "Monitoring and Troubleshooting at Scale with Advanced Analytics", SCTE Cable-Tec Expo 2021.

A. Mohan and X. Huang, "Robust and Resilient Service Assurance System Design with Observability to Improve Enterprise Customer Experience", SCTE Cable-Tec Expo 2022.

