MANIPAL
ACADEMY of HIGHER EDUCATION
(Institution of Eminence Deemed to be University)

MANIPAL SCHOOL OF INFORMATION SCIENCES
(A Constituent unit of MAHE, Manipal)

Analysis of Marvis Minis Connectivity Test Results

Juniper Networks

A Postgraduate Project Report submitted to Manipal Academy of Higher Education in partial fulfilment of the requirement for the award of the degree
Of

Master of Engineering
ME (Big Data Analytics)

Submitted by
NIDHI S
231058001

Project Start Date: 10/06/2024

Industry Guide:
Shirley Wu
Software Engineering Senior Director
Juniper Networks
Bangalore

Internal Guide:
Dr Arockiaraj S
Associate Professor
MSIS
MAHE, Manipal

Juniper Business Use Only

MANIPAL
ACADEMY of HIGHER EDUCATION
(Institution of Eminence Deemed to be University)

MANIPAL SCHOOL OF INFORMATION SCIENCES
(A Constituent unit of MAHE, Manipal)

CERTIFICATE

This is to certify that the project titled
Analysis of Marvis Minis Connectivity Test Results

is a record of the Bonafide work done by

Nidhi S
Reg. No: 231058001

submitted in partial fulfillment of the requirements for the award of the degree of **Master of Engineering - ME (Big Data analytics)** of Manipal School of Information Sciences, Manipal, Karnataka (A Constituent Unit of Manipal Academy of Higher Education), during the academic year 2024- 25, and the same has not been submitted elsewhere for the award of any other degree. The project report does not contain any part or chapter plagiarized from other sources.

Shirley Wu
Software Engineering Senior Director
Juniper Networks
Bangalore

Dr Arockiaraj S
Associate Professor
MSIS
MAHE, Manipal

Dr. Keerthana Prasad
Director
Manipal School of Information Sciences

Juniper Business Use Only

Date: 15th April 2025
Intern Name: Nidhi S
Employee ID: 118658

TO WHOMSOEVER IT MAY CONCERN

This is to certify that **Nidhi S, 231058001** from **Manipal School of Information Sciences (MSIS) Manipal** is undergoing his/her internship at Juniper Networks India Pvt Ltd., Bengaluru, from **6/10/2024** and scheduled to complete on **5/30/2025** under the guidance of **Shirley wu**.

We will be issuing the **Release and Experience Certificate** once the internship ends with juniper Networks.

Yours sincerely

M.N. Kakade

Manohar Kakade
Talent Acquisition Sr Director
Juniper Networks India Pvt Ltd.

JUNIPER NETWORKS

Juniper Networks, India Pvt. Ltd.
Elnath, Survey # 111/1 to 115/4, Wing A&B
Amane Belandur Khane Village
Exora Business Park, Marathahalli,
Sarjapur Outer Ring Road
Bengaluru 560 103 , India.
o: +91 80 6121 9999 f : +91 80 6121 9998
CIN : U72200MH2000FTC126336
Email : India-support@juniper.net

Juniper Business Use Only

Registered Office :
Juniper Networks, India Pvt. Ltd.
Unit # 103, 1st Floor, Platina
Plot # C-59, 'G' Block, Bandra Kurla Complex
Bandra East, Mumbai - 400051, India.
o +91 22 6121 3700 f +91 22 6121 3709

www.juniper.net

Acknowledgment

First and foremost, I would like to extend my heartfelt gratitude to **Juniper Networks**, my workplace, for granting me the incredible opportunity to work on cutting-edge technologies. The resources, support, and enriching environment provided by the organization have been instrumental in my professional growth and learning.

I am sincerely grateful to my manager, **Ms. Shirley Wu**, for her steadfast support and thoughtful direction throughout the course of this project. Her leadership and vision have significantly enhanced my learning experience.

I would especially like to thank my mentor, **Mr. Rinoob Babu** and **Mr. Prathamesh Dnyanesh Kumkar**, for his invaluable guidance, insightful feedback, and unwavering encouragement. His mentorship has played a pivotal role in shaping my understanding and approach.

A special note of thanks to my colleagues, **Dharun Bharathi** and **Varun M**, for their collaboration and camaraderie. Their enthusiasm and teamwork made this journey both productive and enjoyable.

I would also like to express my sincere appreciation to **Dr. Keerthana Prasad**, our director, for fostering a culture of excellence and providing an inspiring environment to grow and learn.

My heartfelt thanks go to my academic guide, **Dr. Arockiaraj S**, for his constant support, mentorship, and belief in my capabilities. His guidance has been invaluable throughout this endeavour.

Finally, I am profoundly thankful to the **Manipal School of Information Sciences** for offering the academic framework, resources, and opportunities that have allowed me to thrive in my field of study.

Thank you all for your unwavering support, encouragement and belief in me.

Juniper Business Use Only

Abstract

Marvis Minis is a cloud-based service that enables proactive detection and fixing of network infrastructure problems by coordinating synthetic testing on network devices. Marvis Minis is a network digital twin that evaluates the network's connection and service reachability using the network architecture. Through proactive user connection simulation via an access point (AP), Marvis Minis can assist in identifying and fixing problems before they affect users. Marvis Minis automatically generates the tests to perform after learning about all the switches, APs, WLANs and active VLANs in a site. For any new addition or modification to the site, such as the addition of new APs, WLANs or VLANs, Marvis Minis creates and updates its testing scopes. The analysis of the marvis-minis connectivity tests explores the crucial elements of failure and timeout situations in the Marvis Minis framework. We seek to identify the underlying causes of these problems, assess the root cause of the timeout mechanisms in place and suggest potential remedies by looking at unsuccessful and timed-out test results.

The results of this analysis will help improve the overall performance and dependability of Marvis Minis connectivity testing, allowing network administrators to proactively handle possible problems and guarantee users best possible network performance. We can increase the precision and efficacy of the Marvis Minis platform, resulting in quicker problem solving, less downtime and a better user experience, by locating and addressing the underlying causes of timeouts and failures.

Juniper Business Use Only

Index

Juniper Business Use Only

List of Figures

Juniper Business Use Only

1.Introduction

1.1.About Juniper Networks

Juniper Networks is a leading multinational corporation that specializes in networking and cybersecurity solutions for enterprise, cloud, and service provider markets. Founded in 1996 and headquartered in Sunnyvale, California, Juniper emerged with a bold mission to revolutionize the performance and scalability of the internet. The company gained early prominence with its high-performance routers that challenged the dominance of established players and set new standards for network efficiency and reliability.

Over the years, Juniper has expanded its portfolio to include switches, firewalls, and software-defined networking (SDN) solutions. Its flagship products include the MX Series routers, QFX Series switches, and SRX Series firewalls, all of which are widely used in data centers and service provider networks. At the core of Juniper's product innovation is its proprietary operating system, Junos OS, which delivers consistency across the company's hardware platforms and simplifies network management.

Juniper is also recognized for its focus on automation and artificial intelligence. Its Mist AI platform, which combines cloud-based management with machine learning, enables proactive network operations and exceptional user experiences. This emphasis on intelligent automation is part of Juniper's broader vision for Experience-First Networking — a strategy aimed at delivering predictable, secure, and measurable performance to users and devices.

In addition to its technological strengths, Juniper is known for its commitment to open standards, interoperability, and collaboration with the broader networking community. It maintains a strong global presence and continues to invest in research and development to stay at the forefront of networking innovation.

As digital transformation accelerates across industries, Juniper Networks remains a key enabler of scalable, secure, and smart connectivity — powering the networks that drive today's digital economy.

Juniper Business Use Only

1.2.About Mist

As a member of the Mist team at **Juniper Networks**, I am proud to be part of a pioneering effort that is transforming enterprise networking through artificial intelligence and cloud-native technologies. Mist, a Juniper company, is at the forefront of delivering **Experience-First Networking** — where the focus shifts from managing network infrastructure to ensuring outstanding user experiences.

Mist Technologies revolutionizes IT operations by combining cloud agility, AI-driven automation, and rich analytics. The Mist platform is built on a **modern microservices-based cloud architecture**, which allows for faster innovation, seamless scalability, and improved network resiliency. This architectural shift sets Mist apart from legacy controller-based solutions, enabling a highly responsive and flexible network management model.

One of Mist's most groundbreaking innovations is **Marvis**, the industry's first AI-powered Virtual Network Assistant. Marvis uses machine learning and natural language understanding to assist IT teams with proactive troubleshooting, root cause analysis, and even intent-based network queries. By delivering actionable insights and resolving issues before users are affected, Marvis significantly reduces operational overhead and mean time to resolution (MTTR).

Mist also introduces a user-centric approach to network performance with **Service Level Expectations (SLEs)** — intuitive metrics that reflect the actual experience of users on the network. These include measurements for throughput, coverage, latency, and successful connection rates, giving IT teams clear and meaningful targets to optimize.

The platform supports wireless (Wi-Fi), wired, and SD-WAN services through a single, cloud-based dashboard. Its **open APIs** and rich telemetry make Mist highly programmable and analytics-friendly, empowering organizations to integrate it easily into their IT ecosystems.

As part of the team driving this innovation, I witness firsthand how Mist is enabling enterprises to build networks that are smarter, more secure, and easier to operate — perfectly aligned with the demands of today's digital workplaces and hybrid work environments.

Juniper Business Use Only

1.3.Marvis Minis

Juniper Networks has been at the forefront of building intelligent network infrastructure. With the integration of Mist AI, the company has introduced several innovations to simplify and automate network operations. Among these is **Marvis Minis**, a feature designed to simulate user connectivity and validate the end-to-end health of the network. In modern enterprise environments, maintaining consistent connectivity, application performance, and rapid issue resolution is paramount. Marvis Minis offers an automated, scalable, and intelligent approach to proactively identifying and resolving network issues.

Marvis Minis is part of the larger Marvis Virtual Network Assistant (VNA) ecosystem. It leverages synthetic testing and AI-driven orchestration to validate network performance, monitor key services like DHCP, DNS, and ARP, and ensure application reachability—all without manual intervention. This document explores its architecture, test flow, customization features, benefits, and real-world impact.

Marvis Minis, a cloud service that orchestrates synthetic tests on network devices like Access Points (APs), switches and gateways. Validations are automatically performed by Marvis Minis on a regular basis. If Marvis Minis detects a failure, it revalidates the issue and extends its validation to further switches and APs. Marvis Minis can determine the total impact of the problem that is, whether it affects a particular switch, WLAN, VLAN, AP or the entire site by broadening the validation scope. Any changes pertaining to APs, switches such as the addition of new devices or modifications to configuration are automatically scoped and validated by Marvis Minis.

At its core, Marvis Minis is a cloud-based service embedded within the Mist AI platform. It interacts with the wireless access points (APs), switches, and gateways deployed in a customer's environment. The key architectural components include:

- **Mist Cloud**: Orchestrates the synthetic tests and gathers results.
- **Marvis Engine**: AI-driven decision-making logic that determines test scope and interprets results.
- **Access Points**: Act as testing agents to simulate client behavior.
- **Network Devices**: Includes switches and gateways involved in routing and service delivery.

Each AP can be instructed to simulate a user device, initiating DHCP requests, ARP probes, DNS queries, and HTTP/HTTPS requests to evaluate the performance and availability of critical services. The data collected is sent back to Mist Cloud for analysis and is displayed in an intuitive dashboard.

Marvis Minis can run in two modes: automated and manual.

Juniper Business Use Only

- **Automated Tests**: Scheduled to run hourly or at specified intervals. Marvis Minis uses AI to determine which APs and VLANs to test based on recent changes, past issues, or patterns in network behaviour.
- **Manual Tests**: Initiated by administrators for specific scenarios such as post-maintenance validation or when issues are suspected in a given site or VLAN.

To minimize network load, Marvis Minis does not test all APs simultaneously. It selects a representative subset and expands the test scope only when issues are detected. For instance, in a large site with 2000 APs, only about 10% might be tested initially.

Marvis Minis can perform the validation on a single site or on several sites within an organization. Every site's active APs, VLANs and running applications are automatically discovered by Marvis. This feature enables Marvis Minis to validate individual APs and all user VLANs without requiring validation of all APs. Take a site with 2000 APs connected to 200 switches, for instance. About 200 APs are triggered by Marvis Minis. If required, Marvis Minis extends the validation scope to further APs based on the failure it sees. This feature makes sure that there isn't any extra strain on the network services.

The validation findings are visible on the Marvis Minis dashboard. It provides a graphical depiction of all the validations that were carried out.

The dashboard displays these particulars.

1. Failed Sites: The total number of sites that failed the validations.
2. Live Minis Tests: The quantity of validations being carried out at the moment.
3. Active Marvis Actions: The quantity of organization-level actions identified by Marvis Minis.

Every hour, the automated validations are executed. Every hour, Marvis Minis changes the scope according to the client's current VLAN information. To make sure the site is functioning,
Marvis Minis verifies the following network services:

- DHCP Protocol
- ARP Protocol
- DNS Protocol
- Application Reachability

Marvis Minis simulates a user connection on active user VLANs and validates the connectivity process using the following steps:

Juniper Business Use Only

DHCP Protocol:

- Marvis Minis initiates the process by simulating a DHCP (Dynamic Host Configuration Protocol) request on the target VLAN. This emulates a device seeking an IP address from the DHCP server within the network.
- The test then records whether the VLAN successfully obtains a valid IP address from the DHCP server. If the DHCP request fails, it could indicate issues with the DHCP server configuration, insufficient IP address pool availability or problems with the VLAN itself.

Gateway Validation via ARP:

- Once an IP address is acquired, Marvis Minis transmits an ARP (Address Resolution Protocol). ARP is a vital network protocol that translates IP addresses into corresponding Media Access Control (MAC) addresses, essential for communication across the network.
- A successful ARP response indicates that the target VLAN can effectively communicate with the gateway device, which is typically a router or firewall. Conversely, a failed ARP response suggests potential problems with the gateway configuration, reachability issues or misconfigurations within the VLAN itself.

DNS Resolution Testing:

- Following successful gateway validation, Marvis Minis simulates DNS (Domain Name System) queries directed towards all DNS server IP addresses obtained during the initial DHCP offer.
- By resolving these queries, Marvis Minis verifies the functionality of the DNS servers and the VLAN's ability to translate domain names into their corresponding IP addresses. Issues at this stage could point towards problems with the DNS server configuration or reachability issues between the VLAN and the DNS servers.

Internet Connectivity and Application Reachability:

To comprehensively assess internet connectivity, Marvis Minis attempts to access pre-defined URLs like captive.apple.com, connectivitycheck.gstatic.com, office.com and teams.microsoft.com.

Juniper Business Use Only

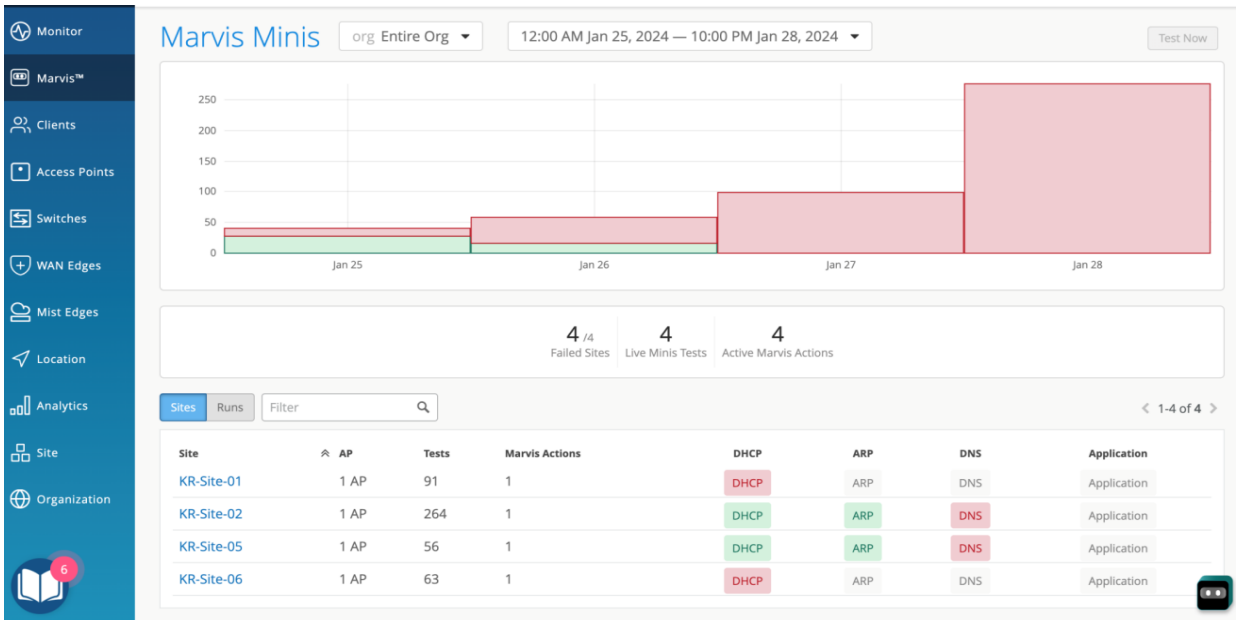

Fig 1. Org Level Marvis Minis Dashboard

The Sites tab displays all the sites in the organization. The table includes:

- Site: The name of the site where the validation was run.
- AP: APs on which Marvis Minis validation is triggered.
- Tests: The number of times the validation was run on the site for the selected timeline (automated and triggered).
- Marvis Actions: Lists the number of Marvis Actions detected by Marvis Minis for the site.
- Network and application services: Marvis Minis provides the validation results for a site for the following network and application services:

We can view the details of each validation run on a site by clicking the site name. In this example, we can see the validations run on a site.

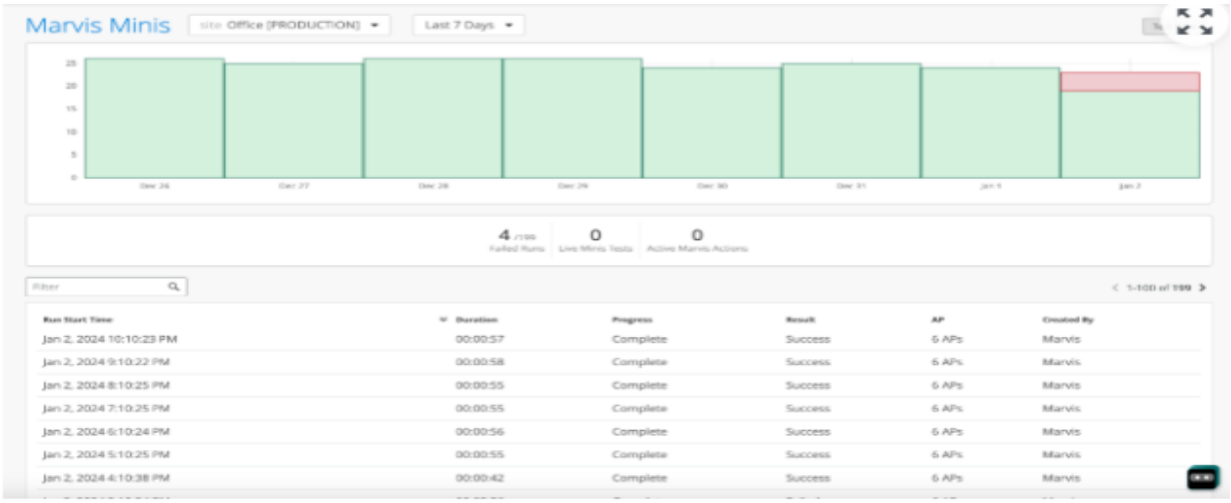

Fig 2. Site Level Dashboard

Juniper Business Use Only

The Created By column indicates who initiated the validation:

1. Marvis: Indicates that Marvis initiated the validation automatically
2. User: Indicates that a user initiated the validation manually

We can also use the Filter option to view specific validations. In the following example, we can see the filtered results for tests run on a specific date.

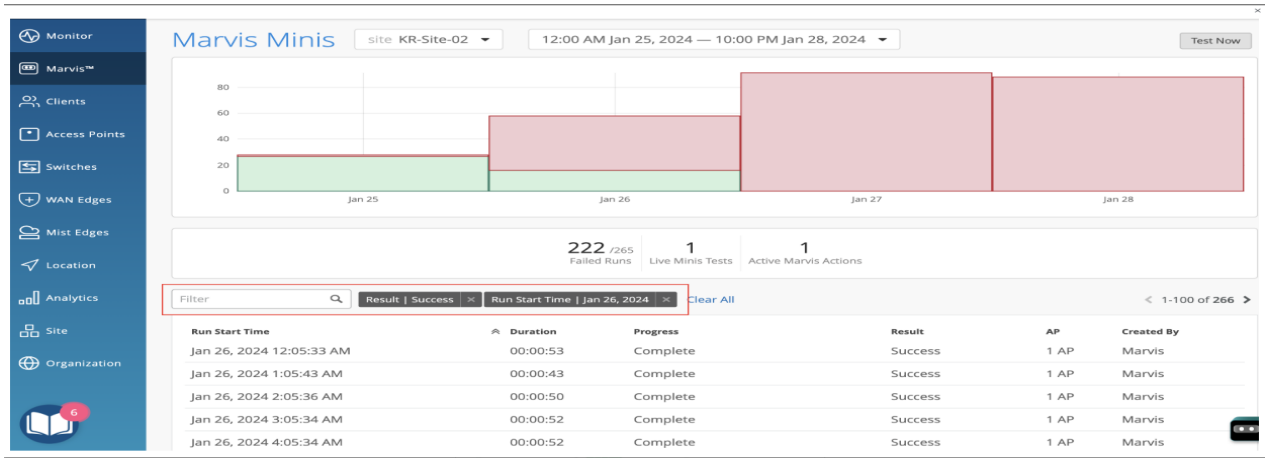

Fig 3. Filtered results for tests run on a specific date Dashboard

To view more information about each validation, click on each row. We will see the details for a validation. The table lists all the APs at the site, the switch to which each AP is connected, VLANs, LLDP port information and the status for DHCP, ARP, DNS and application connectivity.

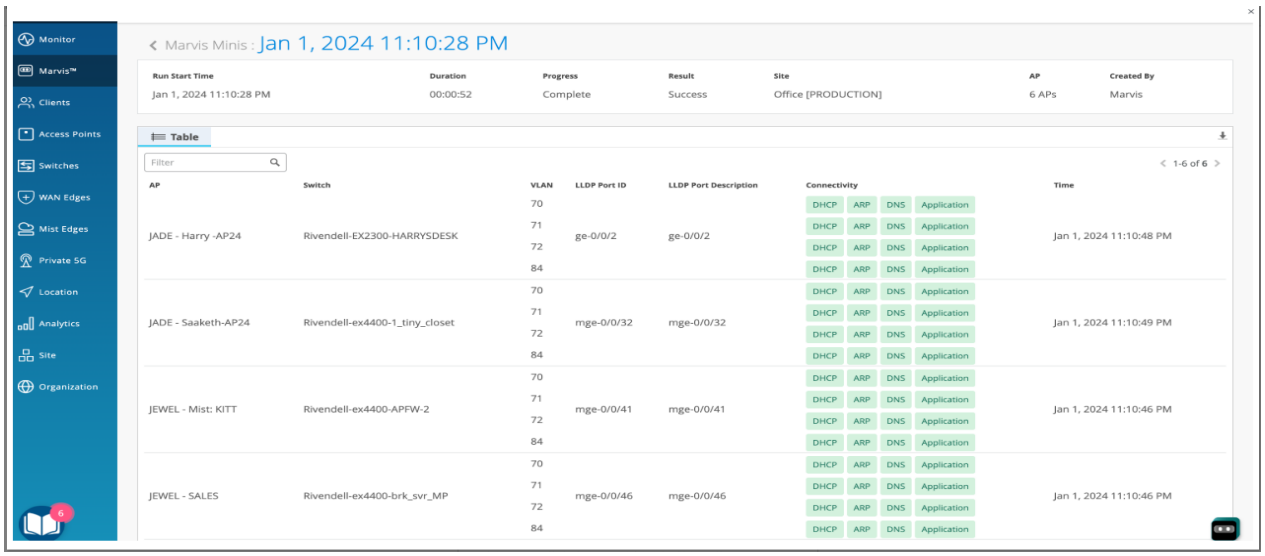

Fig 4. Information on each validation Dashboard

Marvis Minis retests each failure for confirmation. It also expands the scope to additional APs to identify whether the failure is limited to a specific VLAN, AP or switch or whether it is a site-wide issue.

Juniper Business Use Only

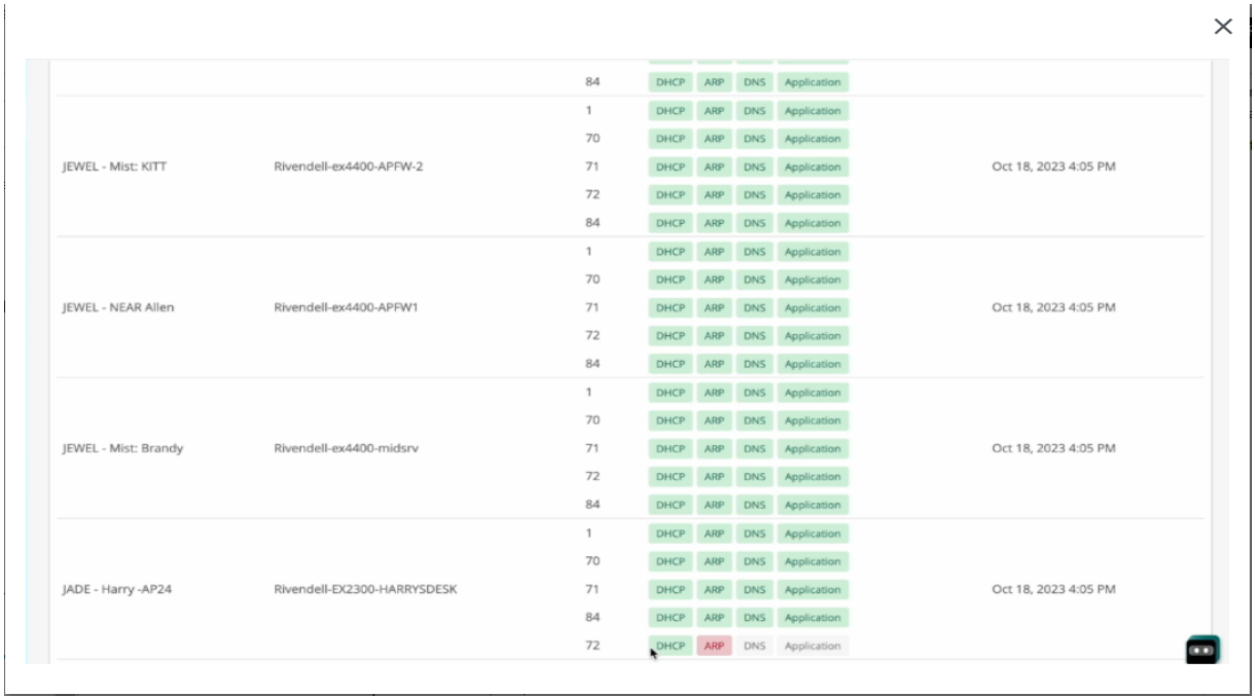

Fig 5. Example of a validation that detected an ARP failure on one of the Aps Dashboard

Marvis Minis is a feature designed to proactively validate the reachability and functionality of application URLs, ensuring network readiness and reliability. By default, Marvis Minis performs validation checks on a predefined set of URLs to confirm that these applications are accessible and functioning as expected from a network perspective. This makes it a valuable tool for maintaining the performance of critical services.

As an administrator, we have the flexibility to configure Marvis Minis to suit our organization's needs. One of the key features is the ability to add custom workload application URLs hosted on Amazon Web Services (AWS) or Microsoft Azure. This functionality is particularly useful for organizations that rely on specific cloud-hosted applications or services. For instance, if an organization uses Azure-hosted applications such as Dynamics 365 or AWS services like an S3 bucket, they can include their URLs in the Marvis Minis validation process. This ensures these applications are continuously monitored for reachability and functionality, reducing downtime and enhancing operational efficiency. Administrators can achieve this by navigating to the Marvis Minis configuration panel, inputting the required URLs, and saving the changes. These URLs are then incorporated into the routine validation checks, ensuring they are as closely monitored as the default set of URLs.

Another significant feature is the ability to exclude certain VLANs from application reachability checks. This is particularly useful for environments where certain VLANs are designated for specific purposes, where application validation checks might not be necessary. For instance, VLANs used for isolated testing or non-critical purposes can be excluded from these checks to avoid unnecessary alerts and to focus monitoring efforts on areas that matter most. This exclusion ensures that the tool only focuses on relevant parts of the network.

Juniper Business Use Only

The validation checks performed by Marvis Minis are designed to ensure the availability and performance of applications. These checks include reachability tests, such as DNS resolution, HTTP/HTTPS connectivity, and latency measurement. If a URL becomes unreachable or experiences unacceptable latency, Marvis Minis logs the issue and provides detailed error reporting, allowing administrators to troubleshoot and resolve the problem proactively. These automated checks run periodically, minimizing the need for manual intervention while providing insights that are essential for maintaining a reliable network.

The ability to customize Marvis Minis by adding specific URLs and excluding VLANs provides several advantages. It allows monitoring to be tailored to an organization's critical applications and infrastructure, ensuring that essential services are prioritized. By excluding irrelevant VLANs, administrators can reduce unnecessary alerts and streamline monitoring efforts. Furthermore, the proactive detection of potential issues helps improve overall network performance and reliability, ensuring that users experience minimal disruptions.

Marvis Minis offers a robust and customizable framework for application reachability validation. By enabling administrators to add specific AWS and Azure URLs and exclude VLANs as needed, it ensures comprehensive and targeted monitoring that aligns with the unique requirements of the organization. This customization enhances the overall functionality of the tool, providing a proactive approach to network management and ensuring that critical applications remain accessible and reliable.

By meticulously simulating these steps, Marvis Minis offers a comprehensive assessment of network connectivity within VLANs. The test results provide valuable insights into potential issues with DHCP servers, gateways, DNS resolution, internet connectivity and application reachability. This information empowers network administrators to proactively identify and troubleshoot network problems before they can disrupt user experience.

Marvis Minis provides a meticulous simulation of various network processes to deliver a comprehensive assessment of network connectivity within VLANs. By simulating key steps such as DHCP server interactions, gateway accessibility, DNS resolution, and application reachability, it creates an in-depth diagnostic framework to evaluate network performance. This thorough approach ensures that every aspect of network functionality, from obtaining an IP address to reaching specific applications, is examined in detail. These simulations help uncover any misconfigurations, bottlenecks or failures in the network infrastructure that could otherwise go unnoticed.

The test results generated by Marvis Minis offer valuable insights into the health and reliability of the network. For instance, it can identify issues such as DHCP server delays, gateway inaccessibility, DNS lookup failures, internet connectivity problems or unreachable applications. By pinpointing the exact source of these problems, network administrators are equipped with actionable data to address and resolve these issues quickly. These insights not only help diagnose immediate network challenges but also provide trends and patterns that can inform long-term improvements in network design and management.

Juniper Business Use Only

With this proactive diagnostic capability, Marvis Minis empowers network administrators to take preemptive action against potential disruptions. By identifying and addressing problems before they escalate, administrators can ensure a seamless user experience and maintain high levels of network performance. This proactive approach reduces downtime, prevents service interruptions and enhances the overall reliability of the network. Ultimately, Marvis Minis acts as a critical tool in maintaining a well-functioning network ecosystem, ensuring that users can access the resources and applications they need without interruption.

In addition to its diagnostic capabilities, Marvis Minis also aids in building confidence in network readiness and scalability. By continuously monitoring and validating critical network processes, it ensures that the infrastructure can handle current demands and adapt to future growth. This makes it an invaluable tool for organizations looking to implement new services, migrate workloads or scale their operations without compromising connectivity or performance. With its ability to simulate real-world scenarios and provide actionable insights, Marvis Minis not only resolves existing issues but also helps future-proof the network, enabling administrators to deliver a reliable and seamless experience for users in an ever-evolving digital landscape.

Beyond the foundational validation capabilities of Marvis Minis, its application reachability tests are uniquely detailed and sophisticated. Unlike traditional tools that rely on basic ICMP pings or DNS checks, Marvis Minis conducts deep HTTP/HTTPS simulations. These simulations include SSL/TLS certificate validation—ensuring that certificates are correctly chained, valid, not expired, and appropriately matched with the target hostname. This is critical for identifying issues that could cause browsers or applications to display certificate errors. Moreover, the tool captures detailed latency profiles for each application URL it checks. This includes DNS resolution time, TCP handshake duration, SSL negotiation latency, and the total page load time, offering granular insights into where bottlenecks might be occurring. In some cases, the system also performs response body pattern matching or HTTP status code verification to confirm that the application is not only reachable but also functioning as intended. These capabilities elevate Marvis Minis beyond simple connectivity validation into the realm of SLA compliance and user-experience assurance.

Marvis Minis is deeply integrated with the broader Mist ecosystem, creating a feedback loop that significantly enhances automated operations. When connectivity or application reachability tests fail, Marvis generates Marvis Actions—intelligent, context-aware recommendations for remediation. These can include suggestions like rebooting specific APs, expanding the DHCP address pool, or updating DNS settings. The intelligence doesn’t stop there. Minis collaborates with Mist’s event correlation engine to uncover the bigger picture. For example, a DHCP failure might be linked to a recent configuration change on a core switch or an upstream firmware upgrade. Such correlations help prevent wild goose chases during troubleshooting. Furthermore, the results of Minis’ synthetic tests can be compared with actual client experience data collected through the Mist SLE (Service Level Expectation) framework. If Minis reports a failure but clients are unaffected, administrators can infer that the issue is either isolated or transient, leading to more informed and less reactive decisions. This integration forms a powerful diagnostic matrix where synthetic and real data amplify each other.

Juniper Business Use Only

In today's world of hybrid and multi-cloud deployments, application access is often distributed across internal data centers, public clouds, and SaaS platforms. Marvis Minis is built to support these complex environments with intelligent cloud-aware testing. Enterprises leveraging federated DNS where internal DNS servers forward requests to cloud-based resolvers like AWS Route 53 or Azure DNS can use Minis to validate the end-to-end resolution path. This includes checks for resolution time and IP address consistency, which are vital for debugging intermittent resolution failures. For content-heavy applications delivered via CDNs, Minis can validate application availability from multiple locations to ensure geo-routing is functioning correctly. Similarly, organizations using site-to-site VPNs or SD-WAN to connect branch sites with private cloud-hosted apps can use Minis to verify whether those apps are reachable over encrypted tunnels. By allowing administrators to add custom application endpoints—internal, SaaS, or cloud-hosted—Marvis Minis becomes a centralized visibility point across all critical services, regardless of their hosting environment.

The wealth of data generated by continuous synthetic testing becomes a treasure trove for data-driven optimization. Over time, recurring DHCP or DNS failures in specific VLANs or APs can reveal deeper architectural issues, such as misconfigured VLANs, overloaded DHCP servers, or suboptimal DNS hierarchy. Marvis Minis captures and trends this data, making it easier for network engineers to conduct root cause analysis and implement permanent fixes. Moreover, Mist's AI engine can analyze historical patterns to deliver predictive insights. For example, if a particular VLAN shows increased latency trends or intermittent failures during peak hours, the system may proactively alert administrators to investigate capacity or policy-related issues. These insights can guide configuration changes like increasing lease duration, reallocating IP scopes, or modifying QoS settings. In this way, Marvis Minis not only helps diagnose current issues but also acts as a long-term optimization and planning tool, supporting continuous improvement efforts.

While Marvis Minis is primarily designed for network health validation, its indirect contributions to security should not be overlooked. The tool's ability to detect anomalies in DNS resolution, ARP behavior, or certificate chains provides a lightweight but effective early warning system for potential security incidents. For example, if a DNS request for a known application resolves to an unexpected IP address, this could indicate DNS hijacking or unauthorized DNS entry manipulation. Similarly, if an ARP request for a gateway returns multiple MAC addresses, it might point to a man-in-the-middle attack or a misconfigured switch broadcasting improper ARP replies. Additionally, the TLS/SSL checks performed by Minis can identify weak encryption configurations, expired certificates, or the presence of interception proxies that modify certificate chains—issues that could degrade trust and expose the organization to risks. While Marvis Minis is not a substitute for a dedicated security platform, its continuous, automated tests significantly enhance an organization's overall security posture by identifying vulnerabilities that might otherwise go unnoticed.

Looking forward, the roadmap for Marvis Minis includes a host of new capabilities that promise to expand its value further. One major area of enhancement is synthetic VoIP testing. By simulating SIP call setup, media negotiation, and jitter/loss measurements, Minis would be able to validate the readiness of a network for real-time communications. This is particularly useful for enterprises using UCaaS platforms like Microsoft Teams or Zoom. Another future feature under consideration is the

Juniper Business Use Only

development of application-specific dashboards. Instead of viewing aggregate performance across all URLs, administrators could access tailored views for key applications like Salesforce, Google Workspace, or Slack. These dashboards would provide deep insight into URL-specific performance trends, latency breakdowns, and failure root causes. Even more ambitious is the planned support for full user journey simulation. Rather than testing simple reachability, Marvis Minis would emulate complex workflows—logging into a portal, performing searches, and submitting forms—mimicking how real users interact with web applications. These enhancements will push Marvis Minis from a validation tool into a full-fledged digital experience simulator, helping organizations ensure not just network readiness, but also true service usability.

Juniper Business Use Only

2.Problem statement

Enterprise networks are growing exponentially in complexity, incorporating an evolving mix of wired and wireless infrastructure, cloud-hosted workloads, hybrid IT environments, and an ever-increasing number of endpoints. The demand for uninterrupted connectivity, low-latency application access, and seamless user experiences is at an all-time high. However, as network infrastructure becomes more distributed and dynamic, diagnosing issues in real-time has grown into a considerable operational challenge.

Traditional monitoring methods rely heavily on passive telemetry or client-side feedback, which often fails to capture transient issues, misconfigurations, or localized outages. Moreover, failures that occur in the early stages of connectivity—such as during DHCP negotiations, DNS resolution, or ARP replies—can go unnoticed until a critical threshold is crossed. This can result in helpdesk tickets, frustrated users, and prolonged downtime before root causes are identified and addressed.

The motivation behind this project stems from these operational gaps. The core aim is to improve visibility into the network validation process and leverage synthetic testing to detect, diagnose, and resolve connectivity failures more effectively. Specifically, the objectives are as follows:

To optimize test coverage while minimizing network strain: Testing every access point (AP) or every VLAN continuously is not feasible in large-scale environments. This project focuses on intelligent test orchestration—selecting the most representative APs and VLANs for validation while avoiding unnecessary load on the infrastructure.

To identify root causes behind failed or timed-out tests: Failures in DHCP, ARP, DNS, or HTTP stages often result in incomplete logs or ambiguous messages. The project aims to break down failure types, enrich them with context, and surface actionable root causes using correlation and pattern analysis.

To analyze latency behavior across protocols, APs, and sites: Beyond pass/fail results, response latency holds critical information. This includes DHCP lease time, ARP response duration, DNS lookup latency, and HTTP roundtrip time. Aggregating these across APs and visualizing deviations helps surface deeper performance bottlenecks.

To improve reliability, reduce MTTR (Mean Time to Resolution), and enhance user experience: Rapid issue detection and fault isolation are key drivers. By automating test execution, logging, and visualization, this project seeks to accelerate problem resolution and ensure network performance meets user expectations.

Despite the sophistication of the Mist AI platform and its Marvis ecosystem, field data reveals several persistent issues that compromise network performance or increase operational overhead:

Tests failing or timing out without actionable logs: Sometimes, synthetic tests (e.g., a CURL request to a known application URL) fail, but without detailed error logs.

Juniper Business Use Only

This makes troubleshooting dependent on external logs or packet captures, increasing MTTR.

APs failing silently or inconsistently: Certain APs intermittently fail to complete tests, but without raising alarms. These silent failures can be due to firmware anomalies, misconfigured VLAN tagging, or network congestion, and often go unnoticed without active monitoring.

Slow DHCP/DNS responses causing delays in real connectivity: Even if tests do not fail, elevated DHCP or DNS latencies can degrade the user experience significantly—causing long connection times, application lag, or failed captive portal redirections. Identifying the source of latency (e.g., congested relay paths or overburdened servers) is essential.

Repeated CURL failures for specific URLs or endpoints: In many cases, application-specific endpoints (e.g., connectivitycheck.gstatic.com or teams.microsoft.com) fail due to cloud-side issues, local DNS forwarding failures, or ISP-specific blocks. Without trend analysis or error pattern recognition, these failures are difficult to diagnose.

Addressing the Challenge: Project Scope and Objectives

This project targets a methodical analysis of failure scenarios encountered in synthetic connectivity validation, with a strong emphasis on Marvis Minis data. The goal is to dissect failure types, correlate them across APs and sites, and design a framework for scalable trend analysis.

1. Analysis of Failure Cases

A foundational component of this project is the categorization and analysis of failed synthetic tests. By dissecting the test flow—starting from DHCP negotiation, followed by ARP gateway resolution, DNS queries, and finally CURL requests—the project identifies failure modes such as:

- **DHCP Lease Denied**: Due to pool exhaustion, rogue DHCP servers, or misconfigured scopes.
- **Gateway ARP Timeout**: Triggered by incorrect subnet configurations, layer-2 isolation, or absent MAC address resolution.
- **DNS Query Timeout**: Caused by unreachable resolvers, misconfigured DHCP options, or packet filtering.
- **HTTP Failure**: Including TLS handshake issues, redirect loops, HTTP 500 errors, or unexpected content validation mismatches.

Using logs exported from Marvis Minis and filtered by site, AP, and timestamp, each failure is cross-referenced with test metadata. The aim is to distinguish between transient issues (e.g., brief server downtime) and persistent faults (e.g., misconfigured DNS forwarding).

Juniper Business Use Only

2. Optimization of Test Scope

In a real-world enterprise deployment, running full-scale synthetic tests across thousands of APs is both inefficient and counterproductive. Instead, the Marvis engine intelligently selects a subset of APs and VLANs based on recent changes, past failures, or network topology.

This project enhances that logic by proposing an **AP-VLAN prioritization model** factoring in:

- Historical failure density per AP**.**
- Latency anomalies across specific VLANs**.**
- Recent configuration changes (e.g., firmware upgrades or VLAN reassignments)**.**
- User traffic volume and business criticality**.**

By simulating various scope selection strategies and measuring detection coverage versus resource consumption, the project aims to derive an optimal test scope for different site profiles (small branch, medium campus, large enterprise).

3. Latency and Outlier Analysis Framework

A significant part of the project involves analyzing latency trends across layers and protocols. Using Spark-based aggregation and visualization tools (such as Plotly or seaborn), the project constructs:

- Latency distribution charts per protocol (DHCP, ARP, DNS, CURL).
- Per-VLAN and per-site latency heatmaps.
- Daily and hourly trend lines for spotting degradations.
- Unusually high latency periods (e.g., DHCP taking >3 seconds).
- Discrepancies between APs on the same VLAN (suggesting switch-level issues).
- Protocol-specific lag (e.g., DNS slower only on guest VLANs).

This framework not only identifies performance bottlenecks but also provides input for alert generation, baseline recalibration, and capacity planning.

Strategic Impact

The value of this project extends beyond reactive troubleshooting. By turning synthetic testing into a proactive performance monitoring tool, organizations can:

- Shorten resolution times for high-impact incidents.
- Prevent outages by identifying latent issues early.
- Improve SLA compliance by tracking service availability and latency metrics.
- Empower IT teams with high-fidelity data and AI-driven recommendations.

Juniper Business Use Only

3.Literature Survey

Overview of Synthetic Network Testing

Synthetic network testing involves proactive, programmable methods to simulate network behavior and measure performance under various conditions. Several tools and frameworks have been developed for this purpose, including iPerf, PingMesh, ThousandEyes, and RIPE Atlas, each with varying capabilities in terms of protocol support, scalability, and diagnostic depth.

Active Monitoring vs. Passive Monitoring

Studies such as [B. Trammell et al., 2014] have distinguished between active and passive monitoring techniques. Active monitoring tools like Marvis Minis inject synthetic traffic into the network to assess availability, latency, jitter, and loss metrics, whereas passive monitoring observes existing traffic. Active monitoring is particularly effective for early fault detection in idle network segments.

Cloud-Scale Network Testing Platforms

Google's Espresso [Jain et al., 2013] and Microsoft's PingMesh [Chen et al., 2014] represent early large-scale efforts in synthetic testing across data centers and WANs. These platforms emphasize distributed agents, lightweight probes, and real-time analysis—concepts also reflected in Marvis Minis.

AI-driven Network Insights

Recent research has explored the fusion of synthetic testing with AI/ML-based analytics. For instance, Juniper's Mist AI leverages real-time telemetry and historical trends to trigger synthetic tests proactively [Juniper Networks, 2021]. Similar approaches are seen in Aruba's UXI and Cisco's ThousandEyes, where machine learning assists in root cause analysis.

Test Scope Optimization

The idea of "intelligent test scope selection" is supported by work on dynamic test orchestration. Studies in adaptive testing frameworks suggest that minimizing test footprint while maximizing diagnostic coverage (e.g., selecting representative VLANs or APs) leads to better scalability and performance [Singh et al., 2020].

Application Layer Reachability

Synthetic testing for DNS, HTTP, and cloud app reachability has been addressed in research like "Measuring the Web" [Krishnamurthy & Wills, 2001] and more recently in "Cloud Outage Detection via Active Probing" [Zhang et al., 2019]. Marvis Minis' design aligns with these principles by validating application-specific paths from the network edge.

Gaps in Existing Research

Juniper Business Use Only

While synthetic testing is well-researched, gaps remain:

- **Test Adaptability**: Most tools run static tests; few adapt tests dynamically based on changing network states.
- **Root Cause Attribution**: Many systems detect performance drops but lack granularity to pinpoint specific faults (e.g., AP misbehaviors, VLAN misconfigurations).
- **Integration with Real-Time AI Ops**: Existing literature often separates network telemetry and synthetic testing, missing opportunities for closed-loop automation.

AI-NOC and Future Directions

AI-NOC is an emerging paradigm where autonomous agents perform continuous monitoring, diagnostics, and corrective actions. Mist AI, via Marvis Minis, exemplifies this trend with:

- **Anomaly Detection**: Automated detection of user-impacting issues before tickets are raised.
- **Event Correlation**: Synthesizing passive and active signals for faster MTTR (Mean Time to Resolution).
- **Intent-Based Testing**: Testing only what matters based on contextual clues like SSID usage or client behavior.

Future directions include reinforcement learning for self-optimizing test strategies and integration with SD-WAN controllers for predictive path validation.

Juniper Business Use Only

4.Technology tack

4.1.Software

Python

Python is a high-level, interpreted programming language widely used for data analysis, scripting, automation, and application development. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Due to its simplicity and large standard library, Python has become a preferred language in academic research and industry alike.
In the context of this project, Python was used to coordinate the entire data pipeline—from reading and processing raw test logs to analyzing metrics and visualizing results. Python also serves as the base language for interacting with libraries like PySpark, matplotlib, and Airflow.
Its easy syntax, dynamic typing, and robust ecosystem make Python ideal for iterative development and rapid prototyping. The vast array of third-party packages provides extended functionality across domains such as web development, scientific computing, and artificial intelligence.

Apache Airflow

Apache Airflow is an open-source platform designed to programmatically author, schedule and monitor workflows. It has grown to become one of the most popular workflow orchestration tools used in data engineering, analytics and other automated processes. With its flexibility, scalability and developer-friendly interface, Airflow is an essential tool for managing complex workflows in modern data pipelines.

Apache Airflow is an open-source platform to programmatically author, schedule, and monitor workflows. It uses Directed Acyclic Graphs (DAGs) to represent tasks and their dependencies, allowing for dynamic pipeline creation and execution.

Apache Airflow offers several features that make it a powerful tool for workflow orchestration. One of its most notable features is its Directed Acyclic Graph (DAG) based approach to workflow design. Workflows in Airflow are defined as Python scripts, with tasks organized in a DAG structure, ensuring clear dependencies and a logical execution order. This Python-based configuration allows for high customization and dynamic workflow creation, catering to a wide range of use cases.

Airflow also supports a robust scheduler that orchestrates tasks based on defined dependencies and execution times. Its scheduler uses the metadata database to efficiently manage and queue tasks, ensuring optimal resource utilization. Additionally, Airflow's web-based UI provides an intuitive interface for monitoring and managing workflows. Administrators can view DAGs, inspect task logs and manually trigger or pause workflows as needed, offering complete visibility and control over the system.

Another standout feature of Airflow is its extensibility. It supports numerous operators, which are pre-built task templates for various integrations, including SQL, Bash,

Juniper Business Use Only

Python, and cloud services like AWS, Google Cloud and Azure. Custom operators can also be created to meet specific requirements. Furthermore, its plugin architecture allows developers to add new functionalities, such as custom hooks, executors and UI components.

Apache Airflow is widely used across industries for automating and managing workflows. In data engineering, it orchestrates Extract, Transform, Load (ETL) processes, ensuring data pipelines run efficiently and reliably. For instance, Airflow can manage workflows that extract raw data from various sources, transform it into meaningful formats and load it into a data warehouse for analysis.

In machine learning, Airflow is used to manage end-to-end pipelines, from data preprocessing to model training and deployment. It ensures reproducibility and scalability by handling dependencies and scheduling tasks. Airflow's ability to interface with various cloud providers also makes it ideal for hybrid workflows, where tasks run across on-premises and cloud infrastructure.

Additionally, Airflow is employed in software development for Continuous Integration/Continuous Deployment (CI/CD) pipelines. Developers can use it to automate testing, building and deploying applications, improving efficiency and reducing manual intervention.

Airflow's architecture is highly scalable and designed to handle workloads of varying sizes. Its modular structure, comprising a web server, scheduler, executor and metadata database, allows for horizontal scaling. Executors, such as the CeleryExecutor or KubernetesExecutor, enable distributed task execution across multiple nodes, making it suitable for high-demand environments.

Flexibility is another strength of Airflow. By supporting dynamic DAG generation, it allows workflows to adapt based on runtime conditions or external inputs. This is particularly useful for organizations with rapidly changing requirements or workflows that depend on variable datasets.

Despite its strengths, Apache Airflow has some challenges. Its reliance on Python for workflow configuration can be a barrier for teams unfamiliar with the language. Additionally, managing Airflow at scale requires careful attention to infrastructure, particularly the metadata database and executor configurations. Inefficient DAG design or poor resource allocation can lead to performance bottlenecks. Lastly, while Airflow is highly extensible, the learning curve for customizing it with plugins and custom operators can be steep for new users.

In this project, Airflow was used to automate data ingestion and transformation tasks. Custom DAGs were created to fetch Marvis test logs, initiate PySpark jobs, and schedule summary reports.

Airflow supports retries, logging, alerting, and monitoring out-of-the-box. It also integrates seamlessly with cloud infrastructure, databases, and APIs. Its modular architecture allows for extension via plugins, operators, and hooks, making it ideal for complex data engineering tasks

Juniper Business Use Only

Apache Spark

Apache Spark is a powerful open-source distributed computing system designed for big data processing. Spark is widely used in data engineering and analytics for processing large datasets that cannot be handled by a single machine. One of the core concepts in Spark is the "job," which represents a specific task or set of operations that are executed on distributed data across a cluster. Spark's design ensures faster processing and scalability by dividing jobs into stages and tasks that can be processed in parallel.

A Spark job is any computational task that needs to be performed on a large dataset. When an action is invoked on an RDD (Resilient Distributed Dataset) or a DataFrame, Spark creates a job to execute the required transformations and produce a result. A Spark job typically involves three main steps: loading data from a source, transforming or manipulating the data using Spark APIs like Map, Reduce, or Join, and storing the processed data in an external storage system.

To achieve efficient processing, a Spark job is divided into smaller units called stages, and each stage contains multiple tasks. These tasks operate on partitions—smaller chunks of the dataset—allowing parallel execution across nodes in the cluster. This distributed processing capability ensures high scalability and significantly reduces the time required to process large datasets.

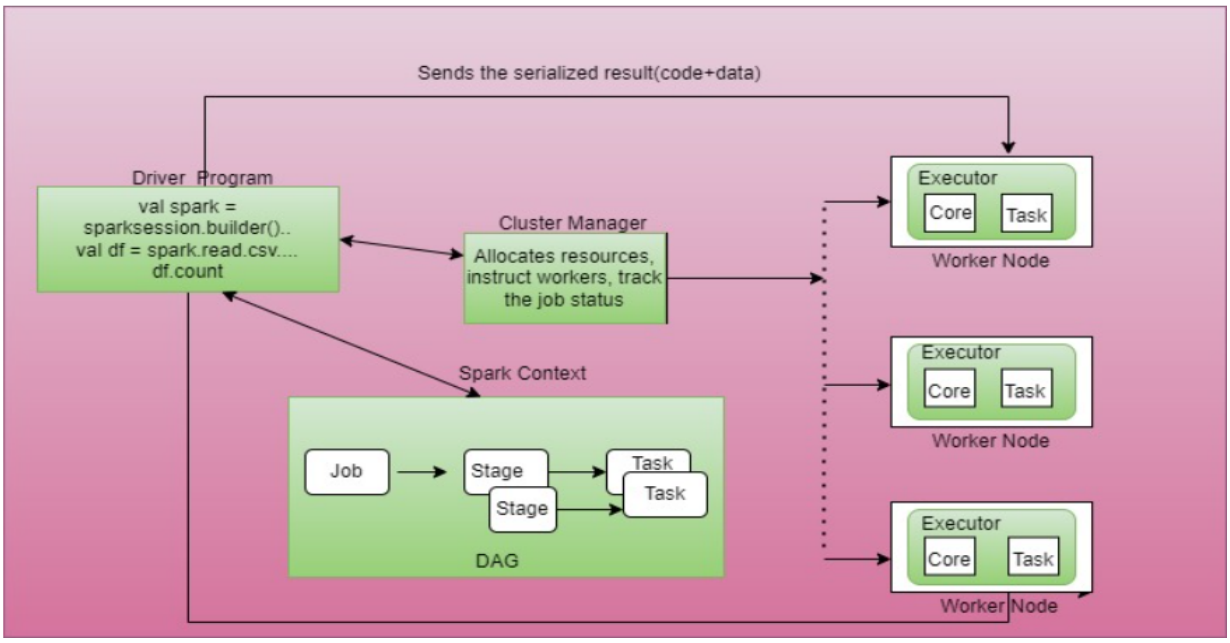

Fig 6. Spark Job Execution

In Spark, a stage is a set of tasks that can be executed in parallel because they share the same dependencies. Stages are created based on the Directed Acyclic Graph (DAG) of transformations that Spark generates when a job is defined. Tasks within a stage operate on individual partitions of the data. By dividing the data into partitions, Spark enables parallelism, where tasks run concurrently on worker nodes in the cluster. The number of tasks in a stage is determined by the number of partitions in the dataset.

Juniper Business Use Only

Spark employs a lazy evaluation model, which means that transformations like map, filter and groupBy do not execute immediately. Instead, they build a logical execution plan called a DAG. The DAG represents the sequence of operations to be performed on the data. When an action, such as count or save, is invoked, Spark examines the DAG and optimizes it before execution. This optimization minimizes data shuffling and improves the efficiency of task execution.

The execution of a Spark job begins with loading data from a source, such as HDFS, a database or cloud storage. The data undergoes a series of transformations defined by the user, such as filtering, aggregation or joining. These transformations build up the DAG. When an action is triggered, Spark schedules the stages and tasks based on the DAG. Tasks are distributed across the worker nodes in the cluster, where they process the data partitions and produce intermediate results. These results are then combined to generate the final output, which is either returned to the driver program or stored in an external system.

Spark jobs can be written in several programming languages, including Java, Scala, Python and R, making it accessible to a wide range of developers. It can be deployed on various platforms, such as Hadoop, Kubernetes and cloud-based services like Amazon EMR, Google Dataproc and Microsoft Azure HDInsight. This versatility makes Spark a popular choice for organizations working with diverse infrastructures.

While Spark provides significant advantages in terms of performance and scalability, managing Spark jobs effectively requires careful attention to certain challenges. Inefficient partitioning or poorly designed transformations can lead to performance bottlenecks and increased execution time. Additionally, shuffling data between nodes can impact performance if not minimized through proper optimization.

To overcome these challenges, developers should design efficient DAGs, minimize data shuffling and use appropriate cluster resources. Partitioning data intelligently and leveraging Spark’s built-in caching mechanisms can also improve performance. Regular monitoring and tuning of Spark jobs using tools like Spark UI or external observability platforms can ensure optimal execution.

Apache Spark’s ability to process large datasets in parallel makes it an essential tool for big data applications. By dividing jobs into stages and tasks, Spark achieves high scalability and performance. Its support for various programming languages and deployment platforms adds to its versatility. Despite its challenges, Spark’s power lies in its ability to handle complex workflows efficiently, making it a critical component of modern data processing pipelines.

PySpark is the Python API for Spark, enabling developers to write Spark applications using Python while leveraging Spark's parallel processing capabilities.
In this project, PySpark was used to handle large datasets generated by Marvis Minis tests. Tasks like filtering connectivity failures, grouping results by access point or VLAN, and aggregating latency metrics were efficiently performed using PySpark DataFrames.

PySpark supports in-memory computing and fault tolerance, making it suitable for big data analytics. It is scalable across hundreds of nodes and integrates well with cloud

Juniper Business Use Only

services like AWS EMR, thus providing flexibility and performance in data-intensive environments.

Jupyter Notebook

Jupyter Notebook is an open-source web application that allows users to create and share documents containing live code, equations, visualizations, and narrative text. It is widely used in data science and research environments for interactive development and experimentation.
During this project, Jupyter Notebooks provided an efficient interface for developing and testing PySpark and visualization code. The ability to see output inline and adjust code iteratively helped in quickly understanding data behaviors and refining the analysis.
Jupyter also supports integration with multiple kernels including Python, R, and Scala, and is compatible with cloud platforms and version control systems, making it a versatile tool for development and collaboration.

AWS EMR

Amazon EMR (Elastic MapReduce) is a cloud-based big data platform provided by AWS. It allows for fast processing of large datasets using frameworks such as Apache Spark, Hadoop, Hive, and HBase.
In this project, EMR was used to run PySpark jobs on cloud clusters, enabling parallel processing of Marvis test logs at scale. This reduced the execution time and allowed analysis across large tenants and time ranges.
AWS EMR supports autoscaling, spot instances, and integration with S3 for storage, making it a cost-effective solution for big data workflows. It abstracts much of the complexity involved in managing Spark clusters, allowing developers to focus on application logic.

Juniper Business Use Only

4.2.Libraries

Matplotlib

Matplotlib is a 2D plotting library for Python that enables the creation of static, animated, and interactive visualizations. It is widely used for its flexibility and compatibility with the Python ecosystem.
In this project, matplotlib was used to plot simple graphs such as failure distributions, bar charts of test statuses, and pie charts showing failure categories. Its fine-grained control over visual elements made it ideal for producing publication-quality plots for the report.
Although it lacks the interactivity of modern web-based plotting libraries, matplotlib remains a reliable choice for generating static images suitable for presentations and documentation.

Plotly

Plotly is a modern graphing library that supports interactive plotting in web-based environments. It enables users to create plots that can zoom, pan, hover, and respond to filters, making it particularly useful for exploratory data analysis.
In this project, plotly was utilized to visualize MTR path fluctuations, hop-to-hop RTTs, and VLAN-wise latencies. The interactivity provided by plotly allowed for better insight into multi-dimensional performance metrics.
plotly also integrates with Dash to build web-based dashboards, making it suitable for real-time monitoring solutions. Its ability to export plots as HTML files provides convenience in sharing insights.

Pandas

Pandas is a Python library providing high-performance data manipulation and analysis tools. It offers data structures like DataFrames, which simplify tasks such as cleaning, merging, and aggregating data.
Although PySpark was used for large-scale processing, pandas was used locally for prototyping logic, inspecting data samples, and preparing inputs for visualization libraries. Its intuitive API allows for operations like filtering, pivoting, and grouping using simple syntax.
pandas is ideal for tasks involving tabular data and integrates well with other Python libraries, such as matplotlib and scikit-learn.

Seaborn

seaborn is a statistical data visualization library built on top of matplotlib. It provides high-level interfaces for drawing attractive and informative statistical graphics.
In this project, seaborn was used occasionally to plot heatmaps and distribution plots for failure rates and latency metrics. Its concise syntax and aesthetically pleasing default styles make it suitable for generating exploratory plots quickly.
Seaborn is especially useful when working with pandas DataFrames and supports complex visualizations like categorical scatter plots, violin plots, and time series line charts.

Juniper Business Use Only

NumPy

NumPy is the fundamental package for numerical computation in Python. It supports multidimensional arrays and provides a wide range of mathematical functions to operate on these arrays.
Although not used directly in large-scale processing, NumPy was crucial in performing small-scale numerical analysis, such as calculating average response times and standard deviations.
It is also used internally by libraries like pandas and scikit-learn, making it an essential building block in the data science ecosystem.

Requests

The 'requests' library is a popular HTTP client library in Python, designed to make web requests simple and more human-friendly. It abstracts the complexities of making HTTP requests behind a simple API, allowing developers to send GET, POST, PUT, DELETE, and other HTTP methods with minimal code.
In the context of network testing and connectivity validation, 'requests' can be used to programmatically verify the reachability of application-layer services such as HTTP servers, APIs, and dashboards. It can also be used to interact with cloud services and fetch real-time data from remote endpoints.
The library provides useful features like automatic content decoding, support for sessions, cookies, headers, and error handling, making it an essential tool for network diagnostic and API-driven test workflows.

Networkx

networkx is a Python library designed for the creation, manipulation, and study of complex networks (graphs). It provides tools to work with both directed and undirected graphs, multigraphs, and weighted networks.
For network testing scenarios, networkx is especially valuable for visualizing hop paths, modeling VLAN connectivity, and analyzing graph-based relationships such as AP-client associations or routing paths. It supports a wide array of algorithms including shortest path, centrality, connectivity, and clustering.
In this project, networkx could be used to construct a visual representation of traceroute paths, helping to identify bottlenecks or path inconsistencies. It also integrates with matplotlib for plotting, enabling both analytical and visual understanding of network structures.

Juniper Business Use Only

5.Development Cycle

The project followed an Agile methodology, emphasizing iterative development, continuous feedback, and collaborative decision-making. This approach was instrumental in refining the analytical models, improving test orchestration strategies, and integrating insights into the Mist AI infrastructure effectively.

Initial Phase

- **Requirement Gathering and Planning**:

The project commenced with planning meetings involving team members from analytics, backend, and platform teams. The focus was to understand Marvis Minis' testarchitecture, data sources, and the problem space—particularly in identifying failure patterns, optimizing test scope, and analyzing latency behaviors.

- **Technology Assessment and Selection**:

Discussions centered around the selection of appropriate tools and frameworks. Given the nature of large-scale test data and the need for distributed processing, Apache Spark (PySpark) was chosen as the core data processing engine. Airflow was designated for orchestrating ETL workflows, and visualization libraries like matplotlib and plotly were shortlisted for generating exploratory plots and dashboards.

- **Infrastructure Setup**:

A dedicated development environment was configured using AWS EMR for scalable Spark jobs. Data access and permissions for Marvis Minis logs were provisioned through internal cloud pipelines, and Jupyter Notebooks were used for prototyping analysis logic interactively.

Iterative Development

- **Cycle-Based Implementation**:

The core development followed bi-weekly sprint cycles. Each cycle involved implementing specific analytical features—for instance, one sprint focused on identifying failure root causes, while another tackled test scope optimization using VLAN clustering.

- **Data Exploration and Validation**:

Every iteration began with data exploration using PySpark and pandas to validate assumptions and detect anomalies in the test results. Sample queries and metrics were run to ensure completeness, quality, and consistency of the dataset.

- **Visualization and Insight Generation**:

plotly and matplotlib were used to generate time series plots, failure heatmaps, and hop-wise latency trends. These visualizations were iteratively refined based on feedback from stakeholders to highlight meaningful patterns.

Juniper Business Use Only

- **Feedback Loops and Reviews**:

Sprint review meetings were held to present interim results and receive feedback from product managers and platform engineers. Suggestions—such as filtering transient failures or grouping by test category—were incorporated into subsequent sprints.

- **Refinement and Integration**:

Insights generated during each cycle were shared with the Mist AI team to align with ongoing diagnostic logic enhancements. Feature requests or data anomalies discovered through this analysis also helped improve test instrumentation itself.

Agile Principles Applied

- **Cross-Functional Collaboration**:

Regular sync-ups were conducted with engineering, data science, and QA teams. The cross-functional nature of these meetings helped align on use cases, address technical blockers early, and share actionable insights.

- **Continuous Improvement**:

Retrospectives were conducted bi-weekly to identify process gaps and opportunities for improvement. For example, early retrospectives revealed the need for better test log parsing, which led to the creation of reusable PySpark modules for log enrichment.

- **Scalability and Modularity**:

Modular design principles were followed in developing reusable functions for log filtering, hop parsing, and latency grouping. These modules were version-controlled and reused across multiple analyses, enabling faster experimentation in future sprints.

- **Documentation and Traceability**:

All analyses, queries, and results were documented in shared Jupyter Notebooks and Confluence pages. This not only ensured traceability but also enabled other teams to build upon the existing analysis framework.

Juniper Business Use Only

6.Methodology

6.1.Minions Service Overview

Overview of the Architecture

The Marvis Minis architecture is designed for efficient, scalable, and intelligent synthetic testing at the site level. It coordinates between different modules to handle synthetic test scope determination, execution, and result processing. The architecture supports proactive diagnostics and rapid identification of network issues through streamlined workflows and modular design.

Minions Service

The Minion Service is a Python Flask-based web application acting as the central communication layer. It handles requests from the UI and Apache Airflow, manages test scopes, and orchestrates test initiation. It is responsible for ensuring no overlapping tests occur, handling concurrency and site-specific execution policies. The service leverages Flask's lightweight design to scale efficiently while offering robust API support, error handling, and workflow orchestration.The Minions Service is a Python Flask-based web service that acts as the backbone of communication within the system, handling API requests originating from both the user interface (UI) and the Apache Airflow scheduler. This service functions as the central interface, ensuring seamless and efficient communication between the frontend and backend systems. It is designed to manage and process requests efficiently, enabling various components of the system to operate cohesively. One of its primary responsibilities is determining the site-level test scope by analysing synthetic test configurations for each site, ensuring that the appropriate tests are assigned and executed based on predefined or dynamic conditions. The service also plays a pivotal role in facilitating dynamic workflows by responding to triggers and providing up-to-date instructions to Airflow tasks, ensuring workflows are executed accurately. Leveraging Python Flask's lightweight and modular architecture, the Minions Service is built to handle high volumes of concurrent requests and scale horizontally as the system expands. Additionally, it provides robust error handling, logging and validation mechanisms to ensure data integrity and system reliability. By centralizing communication and simplifying workflows, the Minions Service ensures that all system components work together seamlessly, making it a critical element in the overall architecture.

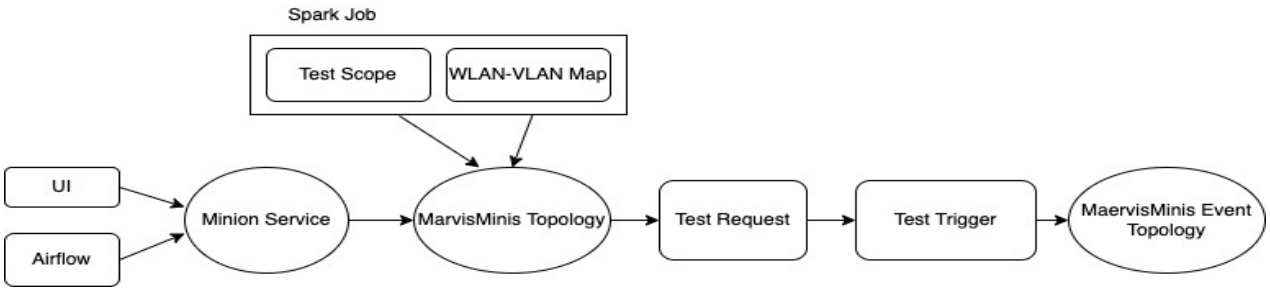

Fig 7. Minis Architecture Diagram

Juniper Business Use Only

MarvisMini Topology Module

This module manages test initiation at the AP level, ensuring that no concurrent tests run on the same AP. It handles queuing, concurrency, and prevents conflicts. The topology module plays a critical role in smooth execution of site-wide tests by monitoring test states and coordinating with the Minion Service to allocate resources.The MarvisMini Topology module is integral to orchestrating synthetic tests for access points (APs) at any given site. It is responsible for initiating tests while ensuring no ongoing tests are running on the same AP to prevent resource conflicts. Before triggering new tests, this module enforces concurrency controls, enabling efficient resource management. By managing these processes, MarvisMini Topology ensures that testing is conducted smoothly without overlapping or interference.

Event Topology Module

Post-test execution, this module collects, processes, and publishes results. It handles partial test failures with a retry mechanism, expanding test scopes if necessary. Final results are published to storage systems for further analysis, and are visualized through the UI to enable fast debugging and resolution. The Event Topology module focuses on handling and processing test results after they are generated. Once synthetic tests are complete, this module processes the results and publishes them to storage systems, making the data accessible for the UI and other tools. For tests that partially fail, it initiates retries with an expanded test scope, which involves testing additional APs or configurations to identify the root cause. If the test completely fails even after retries, the module publishes the data for further analysis, providing administrators with detailed insights for troubleshooting and resolution.

Spark Job for Test Scope Optimization

The Spark job analyzes AP-switch connectivity, WLAN-VLAN mappings, and prior test data to dynamically optimize test scopes. It selects minimal AP-VLAN combinations to ensure test efficiency while maximizing coverage. This data-driven approach reduces redundant tests and optimizes resource use. To optimize the site test scope, a Spark job is employed. This job plays a critical role in analysing the connectivity and mapping within a site to determine the most efficient test configurations. It evaluates the access point (AP) and switch uplink connectivity, along with the WLAN/VLAN mappings, to generate the optimal test scope. The goal is to minimize the number of AP/VLAN combinations required while ensuring maximum coverage of the switch/WLAN/VLAN relationships. By reducing redundant testing, this Spark job not only improves efficiency but also ensures comprehensive testing of the network topology, leading to better utilization of system resources and faster test cycles.

Data Pipeline and Test Triggering

From test request to execution, a streamlined data pipeline connects Minion Service, Topology modules, and Airflow tasks. The pipeline ensures sequential task execution, accurate trigger propagation, and robust test state tracking.

Juniper Business Use Only

Error Handling and Fault Tolerance

Each module includes mechanisms for detecting and recovering from failures. The system supports retries, logs error states, and generates alerts. This ensures high availability and resilience.

Scalability and Distributed Execution

The modular design supports scaling horizontally across sites. Minion Service instances can be added for load balancing, while Spark handles large datasets efficiently in distributed mode.

Security and Access Control

API endpoints are secured through token-based authentication. Site isolation and role-based access control ensure that only authorized users can view or trigger tests.

Visualization and Monitoring

A dynamic UI reflects real-time test status and metrics. Visualization tools help administrators identify network anomalies and test outcomes quickly, with integration into dashboards for reporting.

Advantages of Modular Design

Independent modules mean that each component can be updated or debugged in isolation. This separation of concerns reduces downtime and simplifies development cycles.

By reducing test load while maintaining coverage, the Spark job enhances test execution efficiency across all supported sites.
To further ensure network performance insights are actionable, Marvis Minis introduces advanced telemetry integrations, real-time data processing, and heuristic-based diagnostics. Each test instance is logged with traceable metadata that includes timestamp, AP identifier, VLAN ID, SSID context, and application-layer response codes. The modular design encourages extensibility, allowing newer test types or integrations (e.g., with external monitoring systems or ticketing platforms) to be added without architectural disruption. Moreover, each module in the architecture is equipped with retry logic, exception handling, and fallback mechanisms. For example, if an AP cannot execute a test due to ongoing firmware upgrades or overload, alternate APs are dynamically selected based on site telemetry data. Security and data integrity are foundational. Each component enforces role-based access, encrypted communications, and audit logs. These features ensure not just operational effectiveness but also compliance with enterprise data governance policies.

Finally, all modules are built with observability in mind. Each microservice emits Prometheus-compatible metrics, supports distributed tracing via Open Telemetry, and integrates with Juniper's centralized monitoring platform. These observability hooks

Juniper Business Use Only

enable SRE teams to detect performance bottlenecks, diagnose failures in test execution, and tune the system for better resilience.

Through the integration of tightly coupled components, the system delivers robust site-level testing capabilities, ensuring that each site is thoroughly evaluated based on its specific configurations and requirements. This integration facilitates the streamlined handling of test results by efficiently collecting, processing and analysing the data, allowing for quick identification of issues and performance trends. Additionally, the system optimizes resource utilization by dynamically allocating resources based on workload demands and site-specific needs, ensuring efficient operation without unnecessary overhead. This cohesive design enables comprehensive monitoring of network systems, providing real-time visibility into their performance and health while simplifying the process of troubleshooting and resolving issues. As a result, the system empowers administrators to maintain high levels of reliability, performance and scalability across the network infrastructure.

To further improve the effectiveness of synthetic testing in dynamic enterprise environments, the Marvis Minis architecture incorporates an intelligent Spark job that plays a pivotal role in test scope optimization. This job dynamically analyzes the site topology and access point (AP) connectivity using real-time telemetry and historical data. By evaluating switch uplink paths, WLAN/VLAN associations, and AP distribution, the Spark job intelligently selects a subset of AP/VLAN combinations that offer maximum coverage while minimizing redundant test executions. This is achieved through graph traversal and heuristics that assess reachability and overlap in network segments. By reducing test load while maintaining comprehensive validation, this job significantly enhances test execution efficiency across all supported sites.

To ensure the insights gained from synthetic tests are not only actionable but also timely, Marvis Minis integrates advanced telemetry pipelines that process data in near-real-time. These pipelines are built using scalable streaming platforms that feed results directly from the test agents to the analytics layer. Each test result is tagged with rich metadata including the timestamp, site identifier, AP MAC address, VLAN ID, SSID name, and application-layer response times or error codes. This granular data allows downstream systems to correlate test anomalies with network events, infrastructure changes, or firmware updates.

Marvis Minis is designed with extensibility at its core. The architecture follows a modular service-oriented design that allows new components, such as additional test engines or integrations with ITSM (e.g., ServiceNow) and monitoring solutions (e.g., Grafana, Splunk), to be added seamlessly. This flexibility supports evolving business requirements and enables continuous innovation without architectural disruption. Moreover, the system is equipped with a robust retry mechanism and intelligent fallback logic. In scenarios where a primary AP cannot perform a test due to maintenance, overload, or firmware upgrades, the system auto-selects alternate APs that meet the test criteria using current telemetry.

Security is embedded at every level of the system. All API communications between microservices are encrypted using TLS, and role-based access control (RBAC) is enforced across user interfaces and service endpoints. Additionally, audit trails are maintained for each test request, execution, and result publication, ensuring complete

Juniper Business Use Only

traceability and compliance with enterprise governance standards. This is particularly critical in regulated industries or in deployments that must adhere to standards like ISO 27001 or SOC 2.

Observability and monitoring are fundamental pillars of the system's resilience. Every microservice emits telemetry that is compatible with Prometheus, enabling real-time alerting and performance visualization. The services also support OpenTelemetry-based distributed tracing, which helps Site Reliability Engineering (SRE) teams pinpoint performance bottlenecks, latency spikes, or systemic failures in test processing pipelines. These observability hooks are integrated into Juniper's centralized monitoring platform, providing unified dashboards and proactive insights into system health.

This tightly coupled and observability-driven architecture ensures robust site-level synthetic testing across distributed networks. The system's ability to dynamically allocate resources, reroute tests based on live conditions, and process results in real time makes it ideal for large-scale enterprise deployments. The modularity and resilience of the design allow the platform to scale with the customer's infrastructure while maintaining high levels of availability, reliability, and test accuracy.

Through this cohesive ecosystem, Marvis Minis empowers network administrators and operations teams with actionable intelligence, faster issue detection, and an automated validation pipeline. It enables IT teams to shift from reactive troubleshooting to proactive network assurance, thereby reducing downtime, enhancing user experience, and supporting strategic digital transformation initiatives.

Juniper Business Use Only

6.2.Marvis Minis – Service-Level Expectation (SLE) Overview

Marvis Minis is a sophisticated site-level synthetic testing framework embedded within Juniper's Mist AI platform. Its primary function is to emulate real-world user activity by conducting automated tests from access points deployed across the network. These tests are not run arbitrarily; they are strategically scheduled and configured to capture the performance of critical services such as DNS, HTTP, ICMP (ping), and network path routing (via traceroute). The overall purpose is to proactively validate the health of services and applications that users rely on, with the goal of identifying potential problems before users experience them. This approach reflects a shift from reactive troubleshooting to a proactive and predictive model for network assurance.

At the heart of this testing methodology lies the concept of Service-Level Expectations (SLEs). SLEs are predefined performance thresholds that define what "good service" looks like. Unlike traditional Service-Level Agreements (SLAs), which are often contractual and business-oriented, SLEs in Marvis Minis are operational and diagnostic. They focus on the practical expectations of service delivery at the network edge. These expectations are used to evaluate the results of synthetic tests and determine whether the services being tested are functioning optimally. For example, a basic connectivity SLE might require a successful ping response in under 50 milliseconds, while an HTTP-based application SLE may expect a full page load in less than 150 milliseconds with a valid HTTP 200 response code.

To operationalize these expectations, Marvis Minis integrates a layered architecture that enables intelligent test execution and data evaluation. Tests are initiated by minion services running on APs at each site. These minions operate independently but follow a centrally coordinated schedule to ensure broad test coverage without redundant or overlapping tests. Each test generates a set of telemetry metrics—such as round-trip time (RTT), DNS resolution time, HTTP status codes, traceroute hops, packet loss, and more. These metrics are streamed to the Mist Cloud, where a powerful analytics engine compares them against defined SLE thresholds. Breaches in SLEs are not treated as isolated failures but are analyzed in aggregate across time and topology to identify patterns and root causes.

One of the most critical SLEs tracked by Marvis Minis is the test success rate, which reflects the percentage of successful synthetic tests over a defined time interval. A high success rate (typically 99.5% or above) indicates that services are reliably reachable and functional. When this metric drops, it usually points to underlying problems such as WAN link instability, DNS misconfiguration, or issues with upstream services. Similarly, latency SLEs measure the response times for DNS lookups, HTTP requests, and pings. Latency is a crucial indicator of network and application health, and Marvis monitors not just average latency but also variance and spike patterns. Consistent high latency or sudden spikes can trigger alerts indicating degraded user experience.

Another key SLE category is reachability, which evaluates whether essential services—such as SaaS applications, internal servers, or DNS resolvers—can be contacted and responded to reliably. Reachability SLEs are often measured using HTTP GET requests or DNS queries. If an expected 200 OK response is replaced by a 5xx error or the DNS lookup fails repeatedly, it indicates a potential service outage or network segmentation

Juniper Business Use Only

issue. Marvis automatically correlates such failures across VLANs, APs, or time periods to identify whether the problem is localized or systemic.

Beyond point-in-time measurements, Marvis Minis also introduces path stability SLEs, which are based on the results of periodic traceroute tests. These tests capture the network path taken to reach a particular destination, including all intermediary hops. By analyzing the consistency of these paths across multiple test cycles, Marvis detects signs of route flapping, dynamic path changes, or congestion in upstream segments. An unstable path—evidenced by frequent hop changes or inconsistent latency at certain nodes—can indicate problems such as ISP-level instability or misconfigured routing protocols.

To manage the scale of testing across thousands of APs and multiple VLANs, Marvis Minis employs an intelligent test scope optimization strategy. It is neither practical nor necessary to test from every single access point continuously. Instead, the system uses Spark-based data processing jobs to dynamically select the most representative APs, VLANs, and target services for each testing interval. This optimization ensures wide test coverage without unnecessary load on the network or cloud infrastructure. Sites with more complexity or previous failure patterns may be prioritized for deeper testing, while stable sites are tested less frequently but still within defined time windows.

When an SLE is breached, Marvis does more than just log the failure. It generates a Marvis Service Event, which includes a detailed summary of what failed, where the failure occurred, and why it is significant. These events are accompanied by human-readable explanations powered by Marvis's AI engine, making it easier for IT administrators to understand and respond to issues. For example, a service event may state: "High DNS latency observed on VLAN 20 across 3 APs in Site A. Suspected upstream DNS resolver issue." Such context allows for faster root cause identification and facilitates faster remediation, especially when integrated with Mist's recommendation and automation workflows.

Moreover, Marvis Minis SLEs are not rigid. They can be adapted based on enterprise requirements, site criticality, or application sensitivity. For mission-critical applications (e.g., Microsoft Teams, Zoom, or Salesforce), stricter thresholds can be applied, and alerting can be tuned to ensure zero tolerance for failures. Conversely, for non-critical internal services, the tolerance levels might be more relaxed. This customization ensures that the synthetic testing strategy aligns with actual business priorities, making the Marvis Minis framework both flexible and effective.

In conclusion, Marvis Minis SLEs form the operational backbone of proactive network assurance within the Mist ecosystem. They define clear expectations for service behavior, continuously monitor network and application performance, and empower administrators with real-time, actionable diagnostics. By combining lightweight synthetic testing, dynamic test planning, advanced data analytics, and AI-driven insights, Marvis Minis transforms traditional network monitoring into a predictive, intelligent, and self-correcting system. This enables organizations to maintain high service quality, reduce downtime, and deliver exceptional user experiences across complex, distributed enterprise environments.

Juniper Business Use Only

7.Work Done and Results

As part of my contributions, I played a pivotal role in the development and enhancement of the test scope generation Spark job, a key component in streamlining synthetic network testing. My primary focus was on implementing an efficient mechanism to limit the number of VLANs tested per access point (AP) to a maximum of 10. This enhancement was critical in optimizing the test process by striking a balance between comprehensive coverage and resource utilization. By limiting the number of VLANs per AP, the testing process became more efficient, manageable and resource-friendly, while still ensuring robust coverage of the site's network topology, including its WLAN and VLAN relationships. My role in the development and evolution of the synthetic testing system, particularly within the Marvis Minis platform, encompassed several critical areas: system optimization, diagnostic deep-dives, root cause analysis, and actionable insights based on large-scale data visualization. A major focus was on making the testing process both efficient and effective through targeted improvements in test scope generation and post-test analysis. Below, I provide an extensive breakdown of the work completed, supported by detailed interpretations of the data visualizations.

7.1.Optimizing the Test Scope with Spark Job Enhancements:

When I began working on the test scope generation Spark job, one of the key challenges I identified was the sheer scale of the problem the system was trying to solve. The original system was designed to run tests across every VLAN for every access point (AP) within a given site. On the surface, this seemed like a reasonable approach, as it aimed to ensure comprehensive network validation. However, the reality was much more complex. Testing every VLAN across all APs created massive inefficiencies. In large networks, with dozens or even hundreds of APs, testing all VLANs across all APs quickly became computationally expensive, and it drastically increased execution time.

The test results also became unwieldy, containing a lot of redundancy. Since some VLANs were tested multiple times across different APs or in similar topologies, much of the data generated was superfluous. The system wasn't differentiating between what data was critical and what was redundant, leading to diminished value in the test results. This redundancy not only wasted valuable computational resources but also made interpreting the results more difficult. Network engineers were overwhelmed with data that didn’t add much to the analysis, making it harder to identify specific issues.

In addition to the computational load and redundant tests, another problem arose: scalability. As the network grew, with more APs and VLANs, the sheer number of tests required for every site ballooned. Without a way to dynamically manage the scope, the system was set up to fail when faced with larger networks or rapid expansion.

Introducing the VLAN Prioritization Algorithm: A Data-Centric Approach to Optimization

To tackle these scalability challenges, I needed to introduce a more nuanced method for test scope generation—one that would prioritize tests based on the criticality of

Juniper Business Use Only

VLANs, the historical context of their failures, and their importance within the network topology. This led to the development of the VLAN prioritization algorithm.

The algorithm was built to be dynamic, considering multiple factors when determining which VLANs to prioritize for testing. Rather than selecting VLANs at random or uniformly testing every VLAN, the algorithm used historical and usage data to ensure the most relevant and impactful VLANs were selected. Here's a deeper dive into each of these key metrics:

Historical Failure Frequency: VLANs that had historically shown higher failure rates were deemed more critical to test. This was based on historical test results, where I tracked which VLANs had the most frequent issues or failures during past validation cycles. The higher the failure rate, the more likely that a test would focus on that VLAN. This made the test scope not just broader but more focused on diagnosing potential failures that could have significant network impacts.

Usage Frequency: The algorithm also took into account the frequency with which a VLAN was used across the network. Some VLANs might have very limited use and could be considered less critical from a network performance perspective. However, other VLANs could be involved in critical business operations or high-traffic applications. By including usage frequency as a metric, I ensured that the test scope always aligned with the practical needs of the network, focusing on the VLANs that would affect end-users the most.

Topological Reach: In large networks, VLANs often span multiple devices, switches, or even different geographical locations. A VLAN with broader reach across a network is likely to have a more significant impact on network performance and reliability. The prioritization algorithm considered this reach to ensure that the tests would cover VLANs that spanned across critical parts of the topology, thus providing valuable diagnostic insights into the network's architecture as a whole.

The culmination of these considerations was a more intelligent way of selecting which VLANs to test. This dynamic approach ensured that the system could still provide broad network coverage but without unnecessary repetition. By focusing on the most critical VLANs based on historical data, usage patterns, and topology, the algorithm reduced the testing burden while still delivering high-quality insights.

Efficient Test Scope Limitation: Restricting the Number of Tests with Filtering and Sampling

While the prioritization algorithm helped select the most relevant VLANs to test, there was still a need to limit the overall test scope to ensure that the system didn't become overwhelmed with excessive tests. To achieve this, I introduced filters and intelligent sampling within the Spark transformation pipeline. These filters were designed to enforce the constraint of limiting the number of VLANs to a manageable number per AP.

Instead of testing every VLAN across every AP, the sampling logic was integrated into the data pipeline. This ensured that for any given AP, only the top 10 VLANs, based on the prioritization algorithm, would be tested. The filter logic would evaluate all

Juniper Business Use Only

available VLANs and select the ones that best represented the most critical and statistically significant scenarios.

By embedding this logic within the Spark pipeline, I was able to seamlessly integrate the scope limitation without disrupting the rest of the testing infrastructure. The Spark job, now optimized, focused only on the most important VLANs, reducing the computational resources needed for testing and accelerating the test execution time. The entire process was more efficient, and the system could handle larger datasets and more APs without running into bottlenecks.

Reducing Redundancy: Limiting AP Tests Per Switch

Another critical improvement was limiting the number of APs tested per switch. Initially, the system tested every AP within the site, which resulted in redundant tests. APs that were closely located or connected to the same switch often exhibited similar network behavior, which meant that testing all APs was unnecessary. The results from one AP could often provide enough data to infer the network performance for other APs on the same switch. By selecting only one AP per switch for testing, I was able to eliminate this redundancy.

This change had a few key benefits:

- **Reduced Test Overhead:** By testing only one AP per switch, the number of tests required per site was reduced, saving both time and computational resources.
- **Focused Testing:** Each AP selected for testing was representative of the entire switch's performance, making the results more meaningful. Engineers no longer had to sift through redundant data, and the system could deliver quicker insights.
- **Scalability:** With fewer tests to run, the system could scale more easily. As the number of APs increased, the system remained efficient, since it wasn't bogged down by repetitive test cases.

Handling Missing LLDP Information: Ensuring Robustness in Diverse Network Environments

In many real-world network environments, LLDP (Link Layer Discovery Protocol) data might be incomplete or unavailable. LLDP is essential for mapping network topology, as it provides information about device connections, such as which APs are connected to which switches. Missing LLDP data can create challenges, as the test framework typically relies on this information to determine the relationships between devices and plan tests effectively.

To ensure the system could still function without LLDP data, I developed custom logic to handle these edge cases. This logic leveraged alternative topology discovery methods, such as:

- **Device Configuration Data:** Where LLDP was missing, I used known device configurations or static network maps, if available, to infer topology.

Juniper Business Use Only

- **Heuristic Methods:** For sites where no topology information was available, the system used heuristic methods to estimate network paths, such as assuming that APs on the same switch or in the same geographical area were likely connected.

This logic ensured that the test framework was flexible enough to work in diverse network environments. Even when critical topology data was missing, the system could still generate tests, ensuring that the network was validated regardless of the completeness of the LLDP information.

The Long-Term Impact: Scalability, Speed, and Accuracy

The improvements I made had several far-reaching benefits. By reducing the number of tests, focusing on the most relevant VLANs, and streamlining the test selection process, I created a more efficient and scalable test framework. The system could now handle larger and more complex networks without running into performance bottlenecks. Test execution times were dramatically reduced, which allowed for faster diagnostics and quicker issue resolution.

Moreover, the ability to prioritize VLANs based on failure history, usage frequency, and topological reach meant that the tests were more targeted and meaningful. This significantly improved the quality of the results, making them more actionable for engineers who needed to quickly identify and resolve network issues.

Finally, by introducing flexibility in handling missing data and eliminating redundancy, the system became more robust and adaptable to real-world conditions, where data might not always be complete or perfect. This made the system production-ready, capable of scaling efficiently while delivering high-quality network validation across large, complex sites.

In summary, these enhancements laid the foundation for a far more efficient, scalable, and accurate testing framework—one that reduced computational load, improved execution times, and ensured more meaningful test results without sacrificing coverage.

Visual Analysis of Test Outcomes

Beyond development, I adopted a proactive approach in analysing connectivity test results, going beyond surface-level observations to deeply investigate and interpret the outcomes. My focus was on identifying and understanding patterns of failures and timeouts, which are critical to improving system reliability and performance.

The Role of Visual Analysis in Synthetic Testing

One of the most effective ways to gain actionable insights from large volumes of test data is through **visual analysis**. In the context of synthetic network testing, where connectivity tests simulate real-world conditions, the raw data can often be overwhelming and difficult to interpret without a structured approach. Visualizations allow for the aggregation and presentation of key performance indicators in an intuitive and easily digestible format. These visual representations provide clarity on trends,

Juniper Business Use Only

anomalies, and patterns that are crucial for optimizing network performance and identifying issues early in the testing process.

By leveraging **advanced data visualization techniques**, I was able to create a set of visual summaries that not only showcased the outcomes of synthetic testing but also provided a deeper understanding of the network's health. The figures included in the analysis served as **diagnostic tools**, allowing me to interpret complex datasets and uncover hidden systemic issues that could otherwise go unnoticed.

Visual Summaries of Test Data

The visualizations I developed aimed to capture a variety of metrics derived from connectivity test runs. Some of the key types of figures that were created to visualize the test outcomes included:

1. Latency Distribution Plots

Purpose: These plots were designed to show the distribution of latency across different network paths or between various access points (APs). By visualizing the latency for each test, it became possible to observe whether there were any outliers or significant deviations from expected performance.

Insight: Patterns in these plots indicated certain APs or network segments where latency was consistently high. This helped pinpoint specific areas that were potential bottlenecks or experiencing issues, which could then be addressed through configuration adjustments or further diagnosis.

2. Failure Heatmaps

Purpose: Heatmaps were used to visualize the failure rates of connectivity tests across different VLANs or APs. Each cell in the heatmap represented a test, and the color intensity reflected the rate of failure or success, with red indicating higher failure rates and green indicating successful tests.

Insight: By analyzing the heatmaps, I was able to quickly identify problematic VLANs or APs with consistent failure patterns. The heatmap visually aggregated failure data, highlighting hotspots that required immediate attention, while also giving a sense of how failures were distributed across the network, helping to avoid misinterpreting isolated failures as systemic problems.

3. Path Consistency and Reliability Graphs

Purpose: These graphs plotted the reliability of network paths by showing path consistency over time or across different test runs. They helped visualize how stable or inconsistent network paths were, especially in dynamic or fluctuating network environments.

Insight: By analyzing path consistency graphs, I could detect issues such as network instability, where paths varied significantly across test runs. This pointed to network flapping or inconsistencies in routing configurations that needed to be fixed to ensure

Juniper Business Use Only

stable connectivity. The graphs also allowed me to detect path divergence, where traffic was being rerouted through less optimal paths, potentially leading to increased latency or packet loss.

4. Cumulative Failure Analysis Plots

Purpose: These plots tracked the cumulative number of failures over time or across test cycles. They illustrated how the network's failure rate evolved as more tests were conducted, providing a clear indication of whether issues were improving or worsening.

Insight: The cumulative failure plots were extremely useful in identifying trends over time. If the failure rate consistently increased, it pointed to deeper systemic issues, such as degrading network performance or configuration changes that were adversely affecting network reliability. Conversely, if failures decreased over time, it indicated that troubleshooting and optimization efforts were effective.

5. Test Execution Time vs. Network Health Correlation

Purpose: This visualization correlated test execution times with overall network health, with each test execution time represented alongside the network's failure rate and latency performance.

Insight: This was crucial for understanding how efficiently tests were running. Long execution times combined with high failure rates suggested that either the test configuration was suboptimal or that there were specific network segments causing the tests to take longer to complete. It helped identify areas where optimization in both the test framework and network infrastructure could yield better performance.

Interpreting the Visualizations: Guiding Optimization and Debugging

The real power of these visualizations lay in their ability to translate raw test data into actionable insights that could drive both optimization and debugging efforts. By visualizing trends and identifying patterns, I was able to pinpoint exactly where the network was struggling and where improvements could be made.

Detecting Systemic Issues and Bottlenecks

One of the key goals of the visualizations was to identify systemic issues, problems that were not limited to a single test case or network segment but were pervasive across the network. For example, in the latency distribution plots, if a significant number of tests showed high latency from specific APs, it suggested that there was a bottleneck at the network layer connecting those APs. Whether it was a routing issue, a physical layer problem, or a capacity limitation, the visualization provided a clear indication of where to focus debugging efforts.

Similarly, failure heatmaps revealed network segments with consistently poor performance, allowing me to concentrate on specific VLANs or APs where failures were clustered. These clusters indicated topological issues that weren't immediately obvious through traditional troubleshooting methods.

Juniper Business Use Only

Optimizing Test Scope and Reducing Redundancy

The visualizations also helped guide the optimization of test scope. As the test results were visualized, I could clearly see if certain VLANs or APs were being tested redundantly, without adding much diagnostic value. This helped in further refining the VLAN prioritization algorithm and the test scope limiting logic by identifying areas where the number of tests could be reduced without sacrificing coverage.

For example, if tests across similar APs showed the same pattern of results, I could eliminate some of the redundant tests. Conversely, if certain VLANs consistently showed performance issues, I could prioritize those VLANs for more frequent testing, which would increase the likelihood of detecting the root cause of network problems.

Debugging and Troubleshooting Specific Issues

Another important aspect of visual analysis was its role in debugging. For instance, path consistency graphs highlighted areas of the network where paths were fluctuating, signaling issues like routing loops, misconfigurations, or load balancing failures. This led to targeted troubleshooting on specific network segments or devices.

Additionally, cumulative failure analysis provided insights into the effectiveness of prior debugging efforts. If, after addressing certain issues, the failure rate decreased and stayed low over subsequent tests, it indicated that the issue had been successfully mitigated. Conversely, an increase in failures after adjustments pointed to potentially new issues or unintended consequences of the fixes.

Continuous Improvement and Long-Term Network Health

The visualizations not only helped with immediate debugging and test optimization but also played a key role in continuous improvement. By periodically reviewing these visual summaries after each round of tests, I was able to track how the network was evolving over time. This enabled proactive adjustments to both the test framework and network configurations, ensuring that the network remained healthy and performance continued to improve.

Additionally, historical visualizations allowed for comparisons over time, making it easier to see whether changes made in the test scope, configurations, or network structure were having the desired impact. This made the entire synthetic testing process more iterative, with each round of tests building upon the last to create a more refined and robust network validation framework.

In summary, the development of these visualizations was integral to understanding the outcomes of synthetic tests and identifying systemic issues within the network. These visual tools transformed complex data into actionable insights, guiding network optimizations, troubleshooting efforts, and long-term improvements. The visual analysis not only helped diagnose current problems but also provided a roadmap for ongoing network health management.

Juniper Business Use Only

7.2.Timeout Analysis

Overall Test Distribution – Pie Chart (Elaborated Analysis)

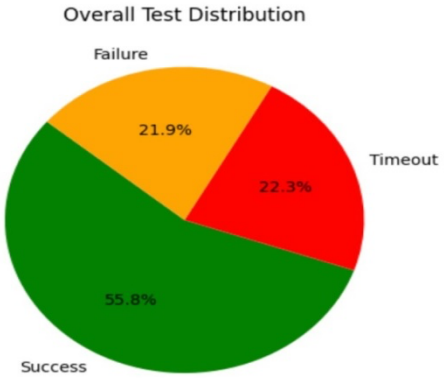

Fig 8. Pie chart representing Test Outcome Distribution

The pie chart in Fig 8 provides a high-level summary of the results from all executed Marvis Minis synthetic connectivity tests across sites. It segments these test outcomes into three primary categories: Success (55.8%), Timeout (22.3%), and Failure (21.9%). Each segment reflects not only the end state of the tests but also provides critical insight into the underlying health, stability, and performance of the network environment under test.

Success (55.8%) – Stable Network Segments

A total of 55.8% of all test cases were marked successful, indicating that the tests executed without any anomalies, errors, or prolonged delays. In these cases:

The full chain of operations—such as ARP resolution, DHCP lease acquisition, DNS query resolution, and CURL (URL) reachability—completed as expected within the configured time limits.

These tests typically required no retries, suggesting good network reliability, optimal test infrastructure performance, and correctly configured AP-VLAN-SSID mappings.

The success rate serves as a benchmark to assess the baseline health of the system. However, while it is a positive indicator, the fact that 44.2% of tests did not succeed underscores a significant opportunity for improvement.

This success metric also reflects the outcome of optimizations in test scope generation, where redundant or unstable VLANs were excluded, improving the signal-to-noise ratio in observed results.

Timeout (22.3%) – Silent Failures with Diagnostic Complexity

The timeout category represents 22.3% of all test executions and is particularly challenging to diagnose. These tests typically:

Juniper Business Use Only

Initiated correctly and passed through early steps (such as action triggering or test scheduling), but failed to receive a response from the AP or network within the allotted timeout duration.

- Involved multiple retries, as the test system is configured to attempt a few retries before declaring a timeout.
- Often required deep-dive diagnostics to determine root causes, including:
- Analysis of AP logs to detect test runner crashes or state machine hangs.
- Packet captures at AP or switch level to detect blackholing, asymmetric routing, or L2 forwarding drops.
- Validation of control plane messaging between the Marvis Minis orchestrator and edge devices.

Timeouts often occur due to intermittent issues in the control or data plane, and are exacerbated in high-load scenarios, congested RF environments, or when VLAN configurations are not consistently propagated across the site. A high timeout rate may also indicate inefficiencies in the test orchestration layer, such as race conditions, delayed job processing, or AP-side overloads.

As such, this segment prompted targeted investigations to improve response reliability and reduce the timeout ratio through architectural tuning.

Failure (21.9%) – Explicitly Reported Errors

The remaining 21.9% of tests failed with clear diagnostic output. These are cases where the AP executed the test and returned a result, but that result indicated an operational failure such as:

- **DNS Resolution Failed**: The AP was unable to resolve the requested domain, potentially due to a misconfigured DNS server or network reachability issues.
- **DHCP Lease Unavailable**: Either the DHCP server was unreachable, IP pools were exhausted, or requests were not forwarded correctly via relays.
- **ARP Resolution Failed**: Indicating L2 communication breakdowns or VLAN misalignment.
- **CURL Errors**: Indicating problems at the application layer, such as HTTP 404, SSL handshake failure, or endpoint unreachability.

These failures are typically easier to categorize and debug than timeouts, as the test report contains granular logs, retry counters, and error codes. The failure category forms the basis of automated root cause classification, enabling:

- Alerting based on repeated error types.
- Triggering automated remediation workflows.
- Building per-site failure heatmaps and prioritizing investigation efforts accordingly.
- Additionally, many of these failures are symptomatic of deeper infrastructure or configuration issues, such as:
- VLAN tagging inconsistencies.
- Misconfigured DHCP scopes or DNS forwarding rules.

Juniper Business Use Only

- Backend service unavailability (in the case of CURL checks).

By analyzing the subtypes within this failure bucket (later detailed in Fig 13), actionable recommendations could be made to network operators to rectify persistent issues. In summary, Fig 8 acted as the foundational metric for measuring the system's effectiveness and reliability. It provided a simplified yet powerful representation of how well synthetic tests were functioning, guiding numerous downstream optimizations and investigations aimed at improving Marvis Minis' diagnostic precision and overall service quality.

Unique Test Count by Test Status – Bar Chart (Elaborated Analysis)

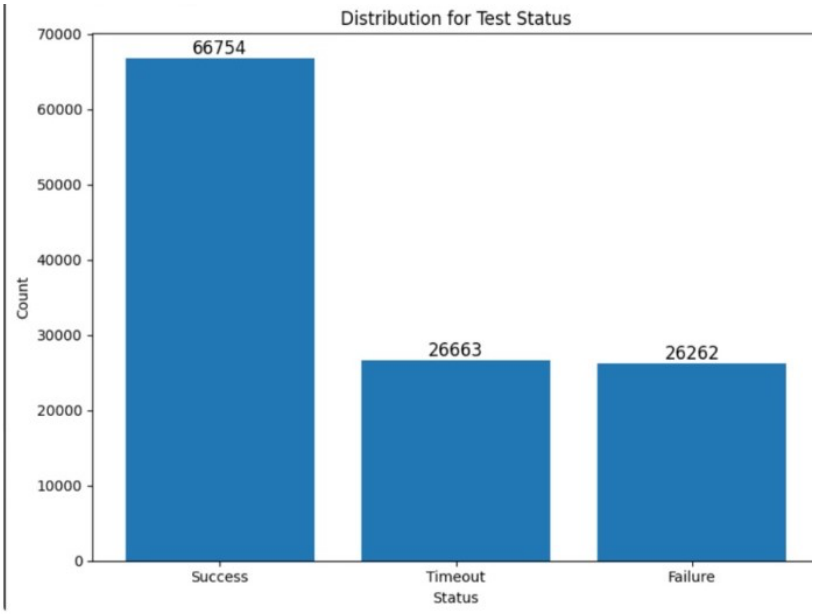

Fig 9. Bar chart represents the Unique test count by Test status

This bar chart serves as a complementary visualization to the overall distribution seen in Fig 8. While the pie chart gives a percentage-based overview, Fig 9 presents the absolute number of unique tests falling into each status category: Success**,** Timeout, and Failure. By moving from proportional data to raw counts, this chart enables a more grounded understanding of test execution trends and load across time, devices, and locations.

Absolute Visibility into System Load and Failure Trends

Each bar represents a discrete tally of how many synthetic tests ended in a particular status, allowing for:

- **Immediate detection of anomalies** in test execution volume—e.g., if an unusually high number of failures occurred during a short time window or within a particular test batch.
- **Normalization across sites**: Since different sites may have varying test loads (based on size or number of APs), working with absolute counts made it easier to correlate failure rates with specific conditions such as site topology, VLAN count, or RF congestion levels.

Juniper Business Use Only

- **Device-specific failure analysis**: The bar chart helped identify AP models or firmware builds with disproportionate failure or timeout counts, highlighting possible firmware regressions or hardware-specific bugs.

Operational Insights and Usage

This chart was instrumental in uncovering time-dependent or firmware-related patterns**:**

- **Time-of-day trends**: For example, spikes in the Timeout category often aligned with maintenance windows, high client density periods, or scheduled firmware upgrades. By overlaying this chart against a time series, we could correlate test result volume with operational timelines.
- **Post-upgrade health checks**: After firmware rollouts, we used this chart to compare failure counts across devices. A sharp drop in failures following an upgrade validated that certain fixes were effective, whereas a rise indicated potential regressions.
- **High-volume anomaly detection**: When total test counts surged unexpectedly in any category, it often signaled either an increase in test scope (e.g., new VLANs being added) or an issue in the orchestrator triggering excessive retries. These anomalies helped drive deeper investigation into test job configuration and orchestration logic.

Strategic Role in Test Health Monitoring

This bar chart served as the foundation for our internal test health monitoring dashboard, where stakeholders could:

- Track the real-time performance of synthetic tests across environments.
- Compare health trends across releases, sites, or access point families.
- Drill down from aggregated counts to specific test logs, enabling faster triage and debugging.

It provided the quantitative backbone for alerting systems, where thresholds were defined based on historical norms for each status. For example:

- If timeouts exceeded a certain count in a short time window, it could trigger alerts for network inspection.
- A sudden spike in failures on a specific model would inform the QA and firmware teams for validation.

Fig 9 transformed abstract ratios into actionable numerical insights. By displaying the unique test count across status categories, it enabled a precise, scalable, and real-time understanding of the synthetic test framework's reliability. This visualization not only enhanced day-to-day monitoring but also empowered teams to make data-driven decisions around firmware stability, AP behavior, site performance, and test scheduling.

For cases involving timeouts, I conducted comprehensive root cause analyses to uncover underlying issues. This involved scrutinizing key factors such as the number

Juniper Business Use Only

of retries attempted during each test and the overall test duration. By carefully examining these metrics, I was able to pinpoint recurring issues, uncover trends, and gain valuable insights into potential inefficiencies or bottlenecks within the system. These efforts not only provided clarity on the nature of the failures but also offered actionable recommendations to optimize testing processes and improve connectivity performance. This analytical approach contributed to a deeper understanding of the system's behaviour and played a vital role in enhancing its overall robustness and reliability.

Test Duration Distribution – Bar Chart

This chart visualizes the distribution of test durations segmented by test result types Success**,** Timeout**,** and Failure.

- **Success tests** generally clustered within a predictable duration range, showing baseline expectations for healthy network interactions.
- **Timeouts**, by definition, showed a distinct spike near the upper threshold (e.g., 10s or 30s), indicating that the system was waiting until the timeout ceiling was reached before aborting.
- **Failures** showed more varied durations, suggesting that error responses were returned at different phases of the test (e.g., DNS failures early, CURL errors later).

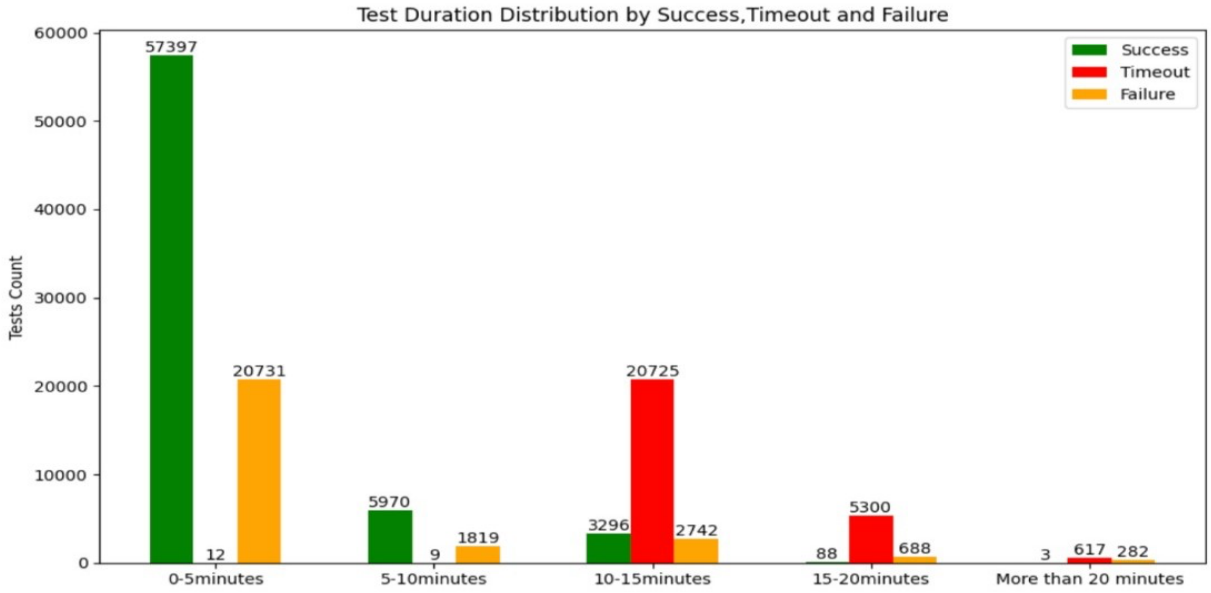

Fig 10. Bar chart representing the test duration distribution for the successful, failed and timed-out tests

This visualization revealed how test duration correlated with test reliability. For example, shorter failures often pointed to configuration errors (like immediate DNS failure), while prolonged timeouts suggested unresponsive network paths. By studying this distribution, we fine-tuned retry logic, test thresholds, and timeout values for different test types to reduce wasted time and improve efficiency**.**

Juniper Business Use Only

Fig 10 illustrates the distribution of test durations segmented by test outcomes: Success**,** Failure**, and** Timeout. The bar chart offers a clear comparative view of how long different categories of tests typically took to complete, shedding light on performance consistency and system responsiveness.

The successful tests largely clustered around shorter durations, indicating that under ideal conditions, the test infrastructure performed efficiently, and the APs were able to complete the test cycle swiftly. These durations represented the baseline expectation for end-to-end test execution. Short test times in this category signified healthy DNS resolution, quick DHCP lease acquisition, and smooth application-layer transactions like CURL requests all contributing to timely test completions.

In contrast, the failed tests demonstrated a wider spread in duration. Some failures occurred quickly—especially in cases like DHCP rejections or immediate DNS failures—while others stretched into mid-range durations, typically where retries were involved before an error condition was concluded. This variability in test duration highlighted how different subsystems (network vs. application layer) influenced how long it took to hit a failure condition. For example, failures due to DNS resolution retries or partial connectivity issues could delay failure determination.

The timeout tests showed a distinct concentration on the far right of the duration scale, occupying the longest time slots. These were tests that failed to return a result within the system-defined time window even after multiple retries. These extended durations were symptomatic of deep-rooted problems such as unresponsive endpoints, AP-side hangs, or broken communication channels between the AP and orchestrator. Unlike direct failures, timeouts consumed more system resources because they held up execution threads while waiting for responses that never arrived.

This chart played a critical role in identifying inefficiencies in the test pipeline. By observing prolonged durations in any category—especially in timeouts—we were able to isolate specific test flows that required optimization. For example, reducing retry intervals or setting smarter timeout thresholds based on test type helped in tuning the overall system responsiveness.

Moreover, Fig 10 served as a key diagnostic tool during incident investigations. For sites experiencing sporadic slowness or scalability issues, analyzing this chart helped correlate increased test duration with infrastructure bottlenecks such as DHCP server overload or WAN latency spikes. It also enabled comparison across firmware versions, providing insights into whether newer builds improved test completion times.

In summary, this bar chart was not just a performance visualization it served as a feedback mechanism that enabled both infrastructure tuning and smarter test design. It helped balance reliability with resource efficiency by allowing us to recalibrate how long we wait before declaring a test as failed or timed out.

Retry Counter Distribution – Bar Chart

This bar chart(Fig 11) shows how many tests required 0, 1, 2, or more retries before completing or failing, segmented by status.

Juniper Business Use Only

- Successes with 0 retries reflected ideal conditions—quick, smooth interactions.
- Successes with retries showed network jitter or initial packet loss but eventual recovery.
- Failures and timeouts with higher retries typically indicated persistent issues or misconfigurations not resolved by retry attempts.
- This visualization was crucial for assessing test reliability and the effectiveness of the retry mechanism.

It helped:

- Identify scenarios where retries were wasteful (e.g., consistently failing after 3 attempts).
- Tune the retry count based on protocol type and AP response behavior.
- Improve test efficiency by dynamically limiting retries when outcomes were predictable.

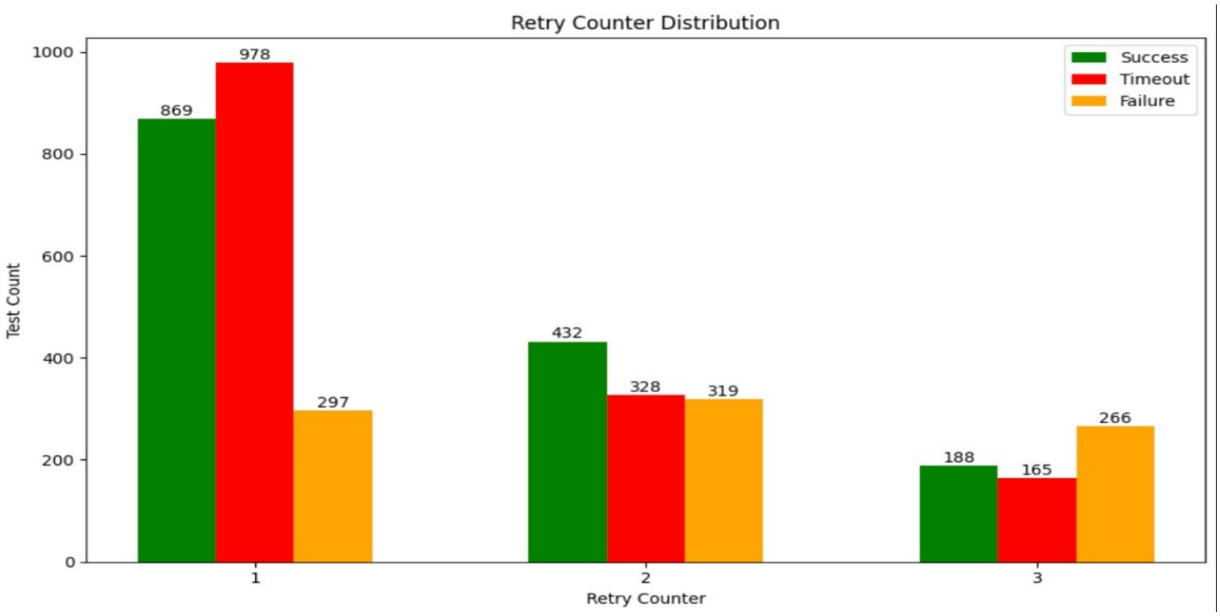

Fig 11. Bar representing the Retry counter distribution

Fig 11 presents a bar chart that captures the distribution of retry attempts made across all executed tests. The retry counter reflects how many times a test component—such as DNS resolution, DHCP request, or a CURL transaction—had to be re-attempted before reaching a conclusive result, be it success or failure. This chart was instrumental in gauging test stability and understanding how often transient issues affected test reliability.

The bar chart shows that a significant number of tests completed with zero retries, indicating a healthy portion of the network where services responded promptly on the first attempt. These no-retry cases typically corresponded with successful test outcomes and highlighted stable APs, responsive infrastructure services (e.g., DHCP and DNS), and minimal packet loss or delay across the network path.

Juniper Business Use Only

However, a noticeable number of tests required one or more retries, which pointed to intermittent network hiccups or service unavailability at the time of testing. For instance, a DNS server may have failed to respond initially due to congestion or high query load, but succeeded upon retry. Similarly, CURL retries could stem from temporary endpoint slowdowns or WAN instability. These cases, while not outright failures, indicated a fragile connectivity environment that might degrade under real-world user load.

More concerning were tests that reached the maximum retry limit, often defined by internal thresholds (e.g., 3 or 5 attempts). These tests either failed or timed out despite repeated efforts, suggesting persistent issues such as unreachable services, misconfigured APs, or software bugs. High retry counts were strongly correlated with timeout and failure categories observed in earlier charts, reinforcing the idea that retries can be a leading indicator of systemic problems.

This retry distribution chart was particularly useful in tuning test behavior. For example, based on the retry trends, adjustments were made to backoff strategies and retry intervals. If a test was likely to succeed on the second try, it made sense to allow a short, controlled retry window. But in cases where retries rarely helped, the retry logic was minimized to conserve system resources and reduce test duration.

Additionally, Fig 11 helped identify APs or test environments with consistently high retry patterns, triggering deeper investigations into firmware behavior, site-specific issues, or even backend orchestration logic. These insights contributed to proactive remediation efforts, such as optimizing DNS cache handling or reviewing DHCP lease timing.

In summary, Fig 11 not only illustrated the retry behavior across the testing framework but also served as a predictive signal for reliability. It helped in identifying where improvements were needed—either in network infrastructure, AP configurations, or test orchestration—ultimately contributing to a more efficient and dependable testing ecosystem.

In my work, one of the more challenging and intricate responsibilities involved diagnosing situations where tests were triggered, actions were generated, but the corresponding test reports were inexplicably missing. These anomalies often pointed to a deeper, underlying issue within the Access Point (AP) itself, requiring thorough investigation to ensure proper functionality and reliability.

To address such cases, I initiated a systematic and detailed investigation process. The first step involved identifying the specific APs that exhibited these issues. This entailed isolating the affected devices from the larger network and gathering critical information about each AP, including its model name and firmware version. These details were instrumental in narrowing down potential causes, as different AP models and firmware versions could exhibit unique behaviours or vulnerabilities.

Test Count With and Without AP Response – Bar and Pie Chart

This figure combines both a bar chart and a pie chart to depict the distribution of synthetic tests based on whether the Access Point (AP) returned a response. It

Juniper Business Use Only

distinguishes between tests where results were successfully reported and those where the system received no feedback from the AP after triggering the test.

Tests that received an AP response represent scenarios where the synthetic test was executed, and a result—successful, failed, or timed out—was logged. These responses ensured end-to-end visibility into test behavior and enabled detailed diagnostics. They allowed us to analyze retries, test durations, failure categories, and other key metrics.

In contrast, tests without any AP response flagged more serious issues. These were cases where the test was initiated, but no result was returned. Such tests pointed toward breakdowns in communication or internal failures within the AP, such as firmware hangs, resource exhaustion, or misconfigured network interfaces.

This subset of tests raised a major concern: the lack of telemetry made debugging nearly impossible. Since no logs, retries, or status codes were received, these incidents created a blind spot in the testing pipeline—what we referred to as "silent failures."

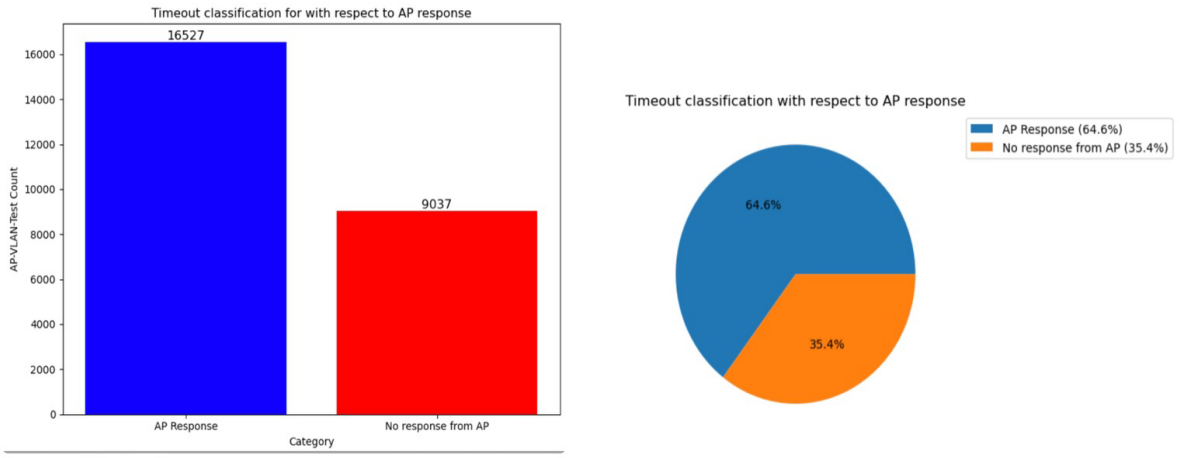

Fig 12. Bar and Pie chart representing the test count with and without AP response

By investigating these no-response cases, we identified APs that repeatedly failed to report back. We grouped them based on model numbers and firmware versions to detect patterns. This helped uncover issues like firmware-level instability or compatibility mismatches that were contributing to test report drop-offs.

The bar and pie charts served as essential tools in pinpointing trends. For example, clusters of non-responsive APs were found in sites running specific firmware builds. This prompted audits and coordination with firmware engineering teams to patch bugs that affected test execution or reporting.

As a mitigation, we also implemented fallback reporting logic. If a test was triggered but no result was received within a specific time window, the orchestrator would emit a

Juniper Business Use Only

placeholder log. This allowed the system to track these incidents under a dedicated "No Response" category, improving the accuracy of failure analysis.

Furthermore, the figure played a key role in enhancing the overall monitoring system. It helped establish proactive alerts that flagged APs with repeated non-responsiveness. This data fed into dashboards used by the QA and support teams to track responsiveness and reliability in real time.

In summary, this chart exposed a crucial layer of operational health that went beyond traditional test success/failure rates. It revealed device-level issues affecting telemetry and allowed us to build safeguards for cases where the APs silently failed to report test results.

For tests that failed outright, I took ownership of performing a comprehensive and methodical root cause analysis to identify the underlying issues and ensure effective resolutions. My approach began with categorizing the failures into distinct areas based on the nature of the problem, such as DHCP, DNS, CURL or ARP-related issues. This categorization provided a structured framework for investigating the root causes and allowed me to focus on specific subsystems or components that might be contributing to the failures.

Failure Result Distribution – Pie and Bar Chart

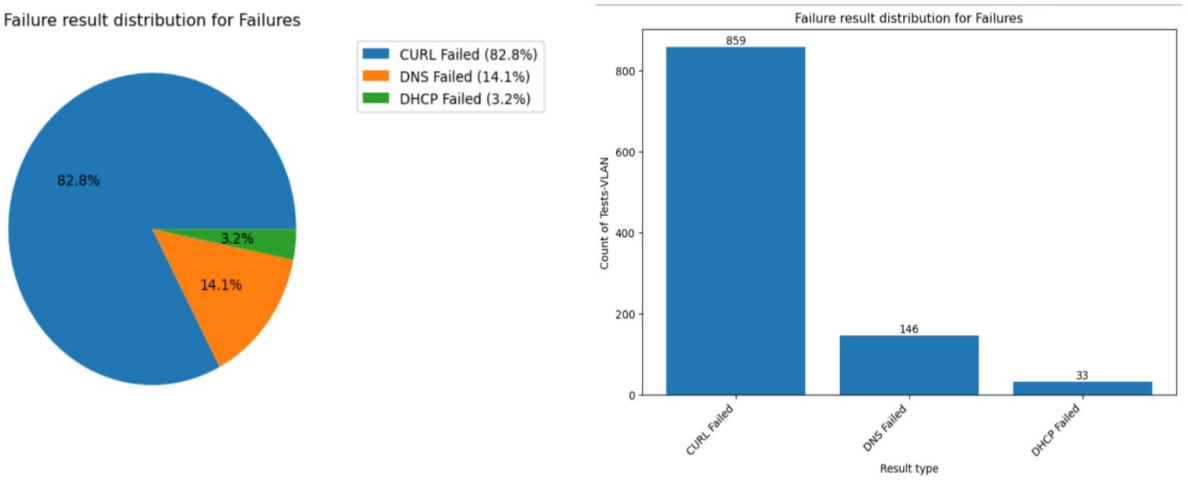

Fig 13. Pie and Bar chart representing the Failure result distribution

For network-related issues, particularly those involving CURL, I went a step further to analyse the specific URLs that were failing. I identified which URLs were most prone to repeated failures and investigated the associated failure types—whether it was a timeout, a 404 error, an SSL handshake failure or network unreachability. It was also important to differentiate between actual failures, where the issue stemmed from a genuine problem (e.g., the endpoint being unavailable) and cases where the failure resulted from missing test data or reporting errors. By addressing these discrepancies, I was able to prevent false positives and ensure that all issues were accurately documented.

Juniper Business Use Only

Patterns and trends in the failures were a critical focus of my analysis. I looked into which sites or endpoints were most commonly affected, whether the failures occurred in specific environments and whether they followed any time-based patterns, such as outages during peak load times. This granular investigation helped identify systemic issues and provided actionable insights for improving the testing framework.

Logs, diagnostic tools and packet captures played a vital role in isolating the root causes. For DNS-related issues, I analysed whether the DNS server was reachable, whether there were latency or resolution errors or if an out-dated cache was causing problems. For DHCP-related failures, I checked for issues like IP address exhaustion or misconfigurations in lease assignments. Similarly, for CURL-related problems, I verified the endpoint's availability, network configuration and SSL certificate validity to pinpoint the exact source of the error.

Fig 13 provides a comprehensive visualization of the different failure categories encountered during synthetic testing. The chart is composed of a pie chart to illustrate the proportional distribution of each failure type and a bar chart to convey the absolute count of failures within each category. Together, they enabled a deeper, structured understanding of where and why tests were failing, beyond just the pass/fail metrics.

One of the most prominent categories identified in this figure was **CURL failures**, which point to issues in application-layer (Layer 7) connectivity. These failures typically occurred when the AP attempted to reach specific URLs and encountered errors such as timeouts, SSL handshake failures, unreachable endpoints, or HTTP errors like 404 or 503. Since CURL tests simulate real-world application requests, these failures highlighted reachability issues that might stem from misconfigured firewall rules, proxy restrictions, or unstable web services.

DNS failures formed another significant chunk of the failure landscape. These occurred when the AP was unable to resolve a domain name to an IP address, suggesting issues at the network or transport layer (Layer 3 or 4). The underlying causes included DNS server unavailability, high resolution latency, or incorrect DNS configuration supplied via DHCP. These failures not only disrupted DNS tests but also contributed to cascading failures in subsequent CURL or endpoint reachability tests.

DHCP failures indicated foundational network problems, as they involved the AP's inability to obtain an IP address lease. This typically occurred due to issues like IP pool exhaustion, DHCP server misconfiguration, or VLAN tagging mismatches. DHCP failures are particularly critical because they prevent the AP from participating in the network, thereby halting all downstream testing activities.

Failures attributed to **ARP (Address Resolution Protocol)**, although fewer in count, were indicative of deeper Layer 2 issues. ARP failures suggested that the AP could not resolve the MAC address of its next hop (usually the gateway), pointing to switch-level misconfigurations, VLAN trunking problems, or port/channel flaps. These required close collaboration with infrastructure and networking teams to investigate broadcast domain integrity.

A small percentage of failures were classified as Unknown or Miscellaneous, typically representing edge cases where logs were incomplete, error codes were ambiguous, or

Juniper Business Use Only

failure reasons did not match known patterns. These cases often demanded deep packet inspection, AP log analysis, and event correlation with infrastructure events or system changes. Although numerically smaller, these failures were often the most complex and time-intensive to resolve.

The pie chart in this figure provided a high-level overview of how different failure types contributed to the total number of failures. This allowed stakeholders to quickly assess which failure category was most dominant. In contrast, the bar chart added value by displaying the actual number of failures per category, helping us understand the scale and prioritize debugging efforts accordingly.

Overall, Fig 13 proved instrumental in guiding engineering resources to focus on high-impact failure areas. It also helped track changes in failure patterns over time—particularly after firmware upgrades or network reconfigurations. This visualization was a central part of the test results dashboard and was routinely referenced to validate operational improvements and to spot new emerging issues early.

Juniper Business Use Only

7.3.Outlier Analysis

Analysis of Latency Trends and Outliers

As part of my ongoing efforts to enhance system performance monitoring, I have undertaken a comprehensive analysis of latency values across various network protocols, identifying trends and anomalies to ensure optimal system functioning. One of the key areas of focus has been tracking and analyzing latency values over extended periods. This allows for the establishment of normal behavior patterns and a baseline against which deviations can be assessed. By consistently monitoring these latency trends, I can pinpoint significant fluctuations that require deeper investigation. This proactive approach enables early detection of performance issues before they escalate, ensuring the system remains stable.

Per-URL Latency Analysis

In addition to analyzing latency at the protocol level, I have expanded my analysis to include latency on a per-URL basis. Certain URLs may exhibit consistent latency issues, which could indicate problems such as server overloads, inefficient routing, or dependencies on external systems. Identifying such URL-specific issues allows for targeted solutions that can improve performance for those particular resources, ensuring that the entire system remains responsive and reliable.

VLAN-Specific Latency Behaviour Analysis

Latency patterns often vary across different VLANs due to factors like network configuration, traffic loads, and hardware setup. As part of my work, I have analyzed latency behavior across various VLANs to detect anomalies tied to VLAN-specific configurations or performance issues. This analysis has proven crucial in pinpointing network inefficiencies and providing insights into how VLAN-specific configurations can impact overall network latency.

Site-Specific Latency Analysis

Recognizing that networks often span multiple sites, I extended my analysis to a per-site level. By doing so, I can identify site-specific trends and outliers that could be unique to particular locations, such as regional network congestion or local hardware malfunctions. Understanding these site-specific patterns has been invaluable in implementing tailored solutions that address the unique challenges faced by each site, resulting in optimized performance across the entire network.

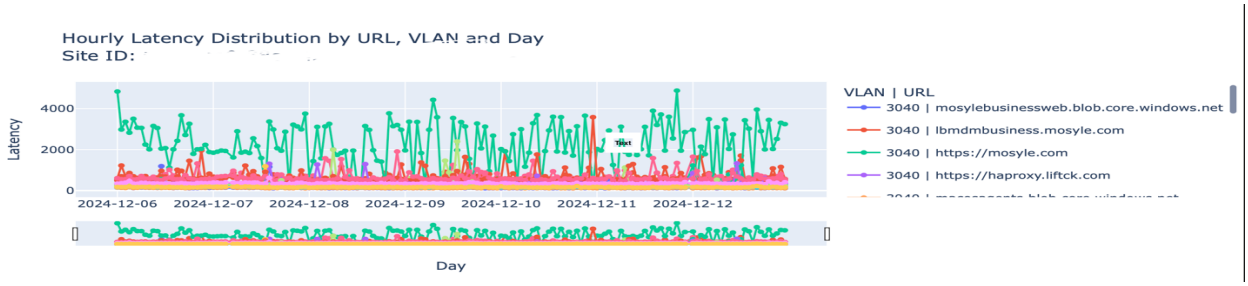

Fig 14. Line chart representing latency distribution per URL,VLAN for a Site

Juniper Business Use Only

The plot visualizes the hourly latency distribution for several URLs within VLAN 3040 at a specific site (identified by the Site ID) over a period of seven days, from December 6th to December 12th, 2024. The y-axis of the main plot displays latency values ranging from 0 to 4000, while the x-axis represents the hourly progression across these days. Each colored line in the main plot corresponds to a unique URL, as indicated in the legend on the right.

Observing the main plot reveals significant fluctuations in latency for all the monitored URLs throughout the week. Notably, the blue line (representing mosylebusinessweb.blob.core.windows.net) and the green line (https://mosyle.com) exhibit considerably higher latency values and frequent spikes, occasionally reaching the upper limit of the y-axis. In contrast, the red line (ibmdbmbusiness.mosyle.com) and the partially visible yellow line show a generally lower and more stable latency profile, with less pronounced peaks.

The smaller plot situated below the main graph provides a condensed, day-by-day overview of the latency for the same URLs. This aggregated view simplifies the comparison of overall latency levels across the entire week. It reinforces the observation that the blue and green URLs tend to experience higher latency ranges compared to the red and yellow URLs. While hourly details are lost in this compressed view, it offers a clearer perspective on the general performance characteristics of each URL on a daily basis.

In conclusion, this visualization offers insights into the network performance experienced by different services at the specified site. The varying latency profiles and the presence of high latency spikes for certain URLs suggest potential areas for investigation and optimization. By analyzing these trends, network administrators can gain a better understanding of service responsiveness and identify potential bottlenecks affecting specific applications or destinations.

Protocol-Specific Latency Analysis (DHCP, DNS, ARP, CURL)

A crucial aspect of my analysis involves examining latency across key network protocols such as DHCP, DNS, ARP, and CURL. Each of these protocols plays a vital role in network communication, and their latency values provide insights into system performance. For example, high DHCP latency may indicate issues with IP address allocation, while DNS or CURL latency could point to server-side problems or network congestion. By isolating and analyzing these protocol-specific latency values, I can gain deeper insights into underlying network issues and optimize overall performance.

The fig 14 display for a specific site. The graph illustrates how the total latency (presumably measured in milliseconds, though the unit isn't explicitly stated) varies across the 24 hours of the day for seven consecutive days, from December 6th to December 12th, 2024. Each day is represented by a distinct colored line, as indicated by the legend on the left.

Juniper Business Use Only

Axes and Data Representation:

- **Y-axis (curlTotal Latency):** The vertical axis represents the total latency, ranging from approximately 1400 to 2400. This suggests that the latency experienced at this site falls within this range during the observed period.
- **X-axis (Hour of the Day):** The horizontal axis represents the hours of the day, from 0 (midnight) to 23 (11 PM). This allows us to see how latency changes throughout a typical 24-hour cycle for each day.
- **Lines (Different Days):** Each colored line plots the curlTotal latency for a specific date:
  - Blue: 2024-12-06
  - Orange: 2024-12-07
  - Green: 2024-12-08
  - Red: 2024-12-09
  - Purple: 2024-12-10
  - Brown: 2024-12-11
  - Pink: 2024-12-12

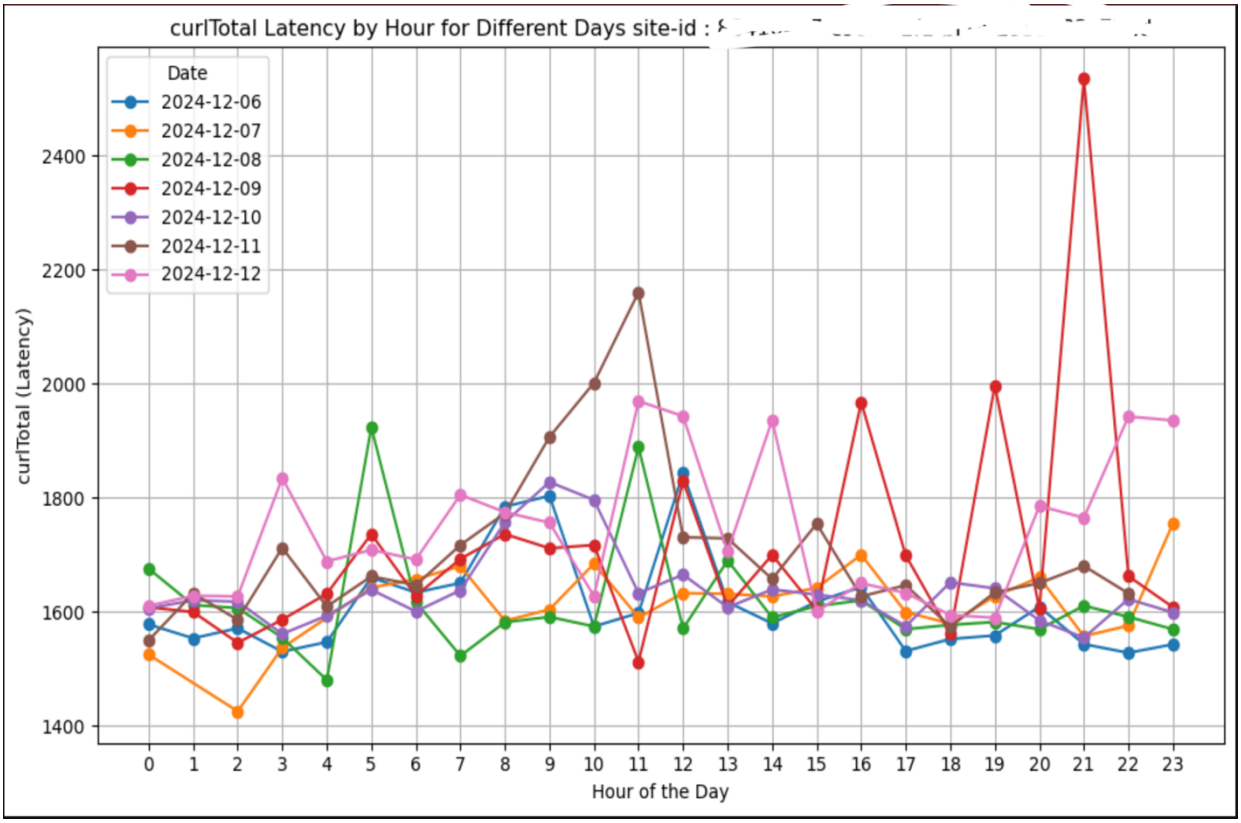

Fig 15. Line chart representing CURL latency distribution

Observations and Analysis:

- **General Latency Levels:** Overall, the curlTotal latency for this site appears to hover between 1500 and 1700 for most hours across the majority of the days. This suggests a relatively consistent baseline latency.
- **Daily Variations:** While there's a general consistency, each day exhibits its own unique pattern of latency fluctuations throughout the 24 hours. Some days show more pronounced peaks and troughs than others.

Juniper Business Use Only

- **Peak Latency Periods:**
  - **December 12th (Pink Line):** This day shows a significant spike in latency around hour 21 (9 PM), reaching the highest point on the graph, exceeding 2400. This suggests a potential performance issue or increased load experienced specifically on this day during that hour.
  - **December 11th (Brown Line):** This day also shows a notable peak in latency around hour 10 (10 AM), reaching above 2000.
  - **Other days:** While other days also have fluctuations, the peaks are generally less extreme compared to December 11th and 12th.
  - **Lower Latency Periods:** Conversely, there are periods where the latency dips lower. For instance, several days show relatively lower latency during the early morning hours (around hour 1 to 4).
  - **Mid-Day Fluctuations:** Many days show some degree of increased latency during the mid-day hours (around hour 11 to 14), although the intensity of this increase varies from day to day.
  - **Weekend vs. Weekday Patterns:** It's difficult to discern a strong weekend (Dec 7th and 8th) versus weekday pattern in this data at a glance. Some weekdays show higher peaks than the weekend days, and vice-versa. A more detailed statistical analysis might be needed to identify any significant recurring weekly trends.
  - **Consistency Across Days:** Despite the fluctuations, the general trend for most days seems to be a relatively stable latency within a certain range, punctuated by occasional increases.

This graph provides a detailed temporal view of the curlTotal latency for the specified site over a week. It highlights that while the latency generally remains within a certain range, there are specific hours and days that experience significant increases. The notable spike on December 12th around 9 PM warrants further investigation to understand the cause of this performance degradation. Similarly, the peak on December 11th around 10 AM could also be a point of interest. By analyzing these patterns, network administrators or application developers can gain insights into the site's performance under different temporal conditions and potentially identify areas for optimization or troubleshooting.

Flagging Latency Outliers for Investigation

Once I have established the baseline latency patterns, I have developed a framework for identifying outliers. These outliers, representing latency values that deviate significantly from the established norms, are flagged for further investigation. Potential causes of these deviations range from network congestion and hardware malfunctions to configuration errors. By regularly flagging and investigating these outliers, I am able to address potential issues before they disrupt network performance, thereby minimizing downtime and maintaining network efficiency.

Statistical Methods for Outlier Detection

To enhance the accuracy of my analysis, I applied statistical methods such as mean, variance, and standard deviation to identify latency outliers. These methods provided a quantitative basis for determining which latency values were abnormal and required further investigation. By calculating the expected range of latency values based on

Juniper Business Use Only

historical data, I established thresholds that could reliably identify significant deviations. This statistical approach ensures that only outliers that truly merit attention are flagged for deeper analysis.

Correlation of Latency Outliers with Other Network Factors

Once latency outliers were identified, I correlated these deviations with other network factors such as network load, specific access point (AP) models, or firmware versions. This analysis enabled me to understand the potential causes of latency spikes, such as high traffic volumes, outdated firmware, or specific hardware configurations. Correlating latency outliers with these factors helped pinpoint the root causes and suggested corrective actions, such as firmware updates or network reconfiguration.

The analysis of the hourly latency distribution offers a comprehensive view of network performance across a range of identifiers, each representing different instances, servers, or network paths. This analysis spans a period from December 6th to December 12th, 2024, and provides an in-depth understanding of latency patterns and their variability. The plot visualizes this data through a time-series format, where the y-axis denotes latency values—likely measured in milliseconds—and the x-axis tracks time, segmented by individual days. The visualization serves as a powerful tool to assess how latency fluctuates on an hourly basis and identifies any anomalies that may arise over time, particularly those that deviate significantly from the norm.

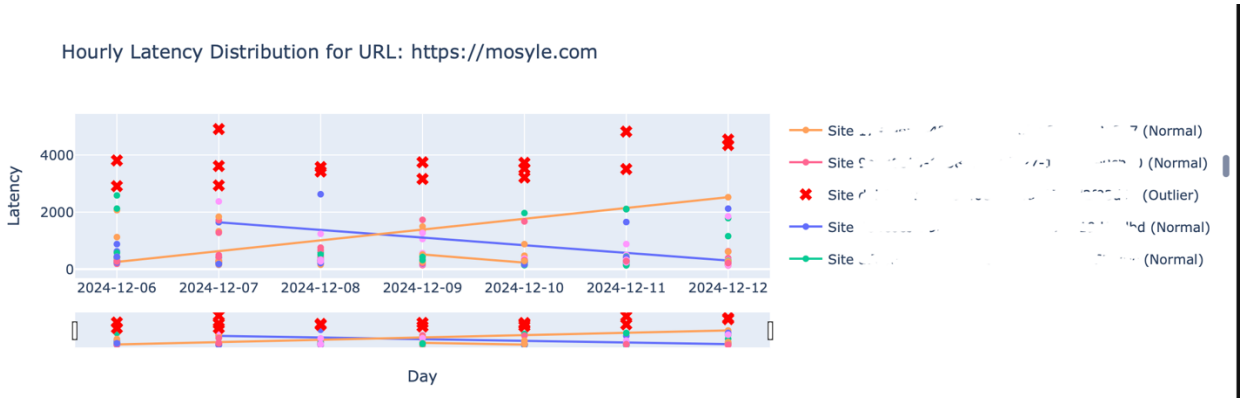

Fig 16. Line chart representing latency distribution with outliers

Note: Site-Ids are scratched as it cannot be used in the report.

The primary purpose of the plot is to identify and highlight outliers in the data, specifically those identifiers that experience latency far beyond the typical range seen in the other data points. These outliers are clearly marked with red 'X' markers, which stand out from the other data points represented by circles. The visual differentiation allows for quick identification of instances that deviate from expected performance, facilitating troubleshooting efforts. By visually distinguishing these outliers, the plot helps to focus attention on specific instances that may require further investigation. This targeted approach is critical in network performance analysis, where vast amounts of data are generated, and focusing on the most significant issues becomes essential.

An essential feature of this analysis is the identification of one specific outlier, which is characterized by consistently high latency across almost the entire time frame. The

Juniper Business Use Only

outlier's red 'X' markers are found across several consecutive days, underscoring that this is not an isolated issue but one that persists over time. This persistence of high latency suggests that the underlying problem is not a temporary spike or anomaly, but a more consistent and recurrent issue. This is a critical observation because it shifts the focus from transient issues that may resolve themselves to more systemic concerns that require deeper investigation. The fact that this high latency is not confined to specific times of day but spans the entire observation period further emphasizes its significance as a recurring problem.

While the latency for the outlier is consistently high, it is not static. There are variations in the latency values observed for this identifier, which fluctuate across different hours of the day. This variability could be due to multiple factors, such as network congestion, changes in traffic patterns, or server load, which fluctuate over the course of the day. Such fluctuations offer valuable insights into the nature of the problem. It suggests that while the latency is generally high, certain periods or conditions exacerbate the problem, causing it to peak even higher. These details highlight the dynamic nature of latency issues and the need for a granular level of analysis to uncover the factors that cause these fluctuations.

In contrast, the "normal" identifiers show a much more stable pattern of latency over the same period. These identifiers, represented by orange, pink, blue, and teal lines, consistently exhibit latency values that remain within a much lower range. Most of these latency values hover below 2000 milliseconds, with many staying below 1000 milliseconds. This stability serves as a baseline against which the outlier can be compared. The clear difference in latency patterns between the "normal" and "outlier" identifiers underscores the abnormal nature of the high latency observed for the outlier. This stark contrast offers a visual representation of what constitutes typical behavior versus what is anomalous, making it easier for analysts to pinpoint areas that require attention.

Statistical methods play a crucial role in the analysis of outliers. By calculating the mean, variance, and standard deviation of latency values for each identifier, it is possible to establish a threshold range that defines what is considered normal. These statistical methods provide a quantitative basis for detecting outliers, ensuring that only those data points that significantly deviate from the expected range are flagged for further analysis. This approach minimizes the risk of false positives, ensuring that only genuinely abnormal latency values are identified. For example, the mean latency for each identifier could be calculated over the entire period, and the standard deviation would offer insights into the natural variation of latency values. Any latency measurements that fall outside of a predetermined threshold, such as more than two standard deviations above the mean, would be considered potential outliers.

By leveraging statistical methods in conjunction with the visualization, the analysis provides a more objective and data-driven approach to identifying latency issues. The mean and standard deviation calculations are particularly useful because they allow for the definition of a range of expected latency values based on historical data. This means that the analysis is not only reliant on subjective judgment or arbitrary thresholds but is grounded in data-driven insights that reflect the typical behavior of the network over time. As a result, the statistical approach strengthens the credibility of the outlier detection process and ensures that attention is focused on truly problematic data points.

Juniper Business Use Only

Once outliers have been identified, the next step is to understand their root causes. This is where correlation analysis with other network factors becomes essential. By examining factors such as network load, access point models, and firmware versions, it is possible to identify whether these elements correlate with the observed latency spikes. For instance, high network load during peak traffic hours may coincide with the higher latency values, suggesting that network congestion is a significant contributing factor. Similarly, access points with outdated firmware may perform suboptimally, leading to increased latency. Correlating latency outliers with these network factors provides deeper insights into why certain identifiers experience higher latency, enabling a more precise diagnosis of the underlying issues.

In some cases, the correlation analysis may reveal patterns that point to specific corrective actions. For example, if high latency is found to be associated with a particular access point model or firmware version, a firmware update or a hardware upgrade may resolve the issue. Similarly, if network load is identified as a contributing factor, optimizing traffic distribution or enhancing network infrastructure could alleviate congestion and improve performance. By identifying these root causes, the analysis provides actionable insights that can lead to targeted interventions aimed at improving network performance.

Furthermore, the combination of statistical outlier detection and correlation with network factors enables a more comprehensive approach to performance optimization. This two-pronged analysis not only identifies where latency issues are occurring but also sheds light on why these issues are happening. This holistic understanding allows for a more informed decision-making process when it comes to troubleshooting and optimization efforts. For example, instead of merely addressing latency spikes as isolated incidents, network engineers can use the insights gained from this analysis to implement long-term solutions that address the systemic causes of high latency.

In conclusion, the analysis of hourly latency distribution and the identification of outliers through statistical methods provides valuable insights into network performance. By isolating and investigating the outliers, the analysis uncovers persistent latency issues that warrant closer scrutiny. The correlation of these outliers with other network factors further enhances the understanding of the root causes of the problems, allowing for targeted corrective actions. Ultimately, this approach ensures that network performance can be optimized by addressing both immediate issues and underlying factors that contribute to latency spikes, leading to a more reliable and efficient network.

Visualization of Latency Trends

To enhance the accessibility and effectiveness of the analysis, I focused on utilizing a line chart to represent latency trends over time, which proved to be an invaluable tool for visualizing and communicating the data. The line chart, with its clear, continuous representation of latency values, enabled the identification of both general trends and specific anomalies across the observed period. This simple yet powerful visualization allowed for a detailed exploration of latency fluctuations at hourly intervals, offering an intuitive understanding of how latency varied for different identifiers throughout the week.

Juniper Business Use Only

The line chart's primary advantage lies in its ability to display temporal changes in latency, making it easier to spot patterns and outliers in a chronological context. By plotting latency values for each identifier as distinct lines, it became clear how the latency for each instance fluctuated over the course of the seven-day period. The varying heights of the lines revealed periods of normal latency, as well as instances where certain identifiers deviated significantly from the expected range, clearly marking the outliers. The use of different colors for each identifier further enhanced the chart's readability, ensuring that even without a detailed legend, the viewer could easily distinguish between the performance of different sites and identifiers.

One of the key features of the line chart was its ability to reveal outliers. The outliers were easily distinguishable from the normal latency patterns due to their stark deviation from the other lines. These outlier data points were plotted with specific markers, like red 'X' markers, which allowed them to stand out against the regular trend lines. This visual distinction was particularly helpful for quickly identifying areas that warranted deeper investigation. It also allowed stakeholders to focus their attention on the specific identifiers that were consistently underperforming, facilitating quicker decision-making.

Beyond highlighting outliers, the line chart also provided a clear view of the overall latency distribution for the entire observation period. By showing the latency across multiple days, the chart helped to convey both the short-term and long-term trends, offering insights into how latency varied across different times of the day and over several consecutive days. This broader perspective was essential for understanding whether latency spikes were temporary or part of a more persistent issue. For instance, the fluctuations in the lines made it possible to see if there were certain periods of the day when latency tended to peak, which could be associated with factors like higher network traffic or server load.

The visual simplicity of the line chart also contributed to its effectiveness in communicating the findings with higher management and stakeholders. Rather than overwhelming them with raw data or complex statistical models, the line chart provided a straightforward representation of latency trends that was easy to digest. The use of clear labels and markers allowed the chart to present essential information at a glance, making it easier for non-technical stakeholders to quickly understand the core issues. This accessibility was key to ensuring that the insights from the analysis were not lost in technical jargon or complex data, but were instead communicated in a way that prompted action.

By incorporating the line chart into presentations and reports, I was able to highlight the areas where latency was consistently high, allowing stakeholders to clearly see the scope and impact of the issue. The chart helped to prioritize actions by illustrating which instances required the most urgent attention based on the severity and frequency of their latency spikes. The visual nature of the chart allowed for a more targeted approach to troubleshooting, as stakeholders could focus on the most problematic identifiers first, streamlining the process of addressing performance issues.

Moreover, the line chart facilitated follow-up discussions and decision-making processes. With a clear visual representation of the data, it was easier for management to ask questions, raise concerns, and suggest potential solutions. The chart acted as a reference point during these discussions, enabling a more focused and productive

Juniper Business Use Only

dialogue about the underlying causes of the latency issues and the potential actions needed to resolve them. This collaborative approach, driven by clear data visualization, helped ensure that the right steps were taken to optimize network performance and reduce latency.

In summary, by using a line chart to represent the latency trends, I was able to make complex data more accessible, actionable, and easier to interpret. The line chart not only highlighted outliers and trends but also facilitated effective communication with stakeholders, ensuring that the findings were understood and could be acted upon promptly. This visualization technique proved to be an essential tool in transforming raw data into meaningful insights that could drive informed decision-making and targeted interventions to improve network performance.

Juniper Business Use Only

7.4.Hop Analysis for the traceroute Data

Traceroute data represents the detailed path that a network packet takes from its origin to its destination across the internet or an internal network. It captures the sequence of intermediate devices—commonly routers or gateways—that the packet passes through, referred to as “hops.” This information is collected by sending packets with increasing Time-To-Live (TTL) values. When a packet’s TTL expires at a router, that router returns an error message indicating its presence and response time. By incrementing TTL values and collecting replies at each step, traceroute effectively maps out the entire route from source to destination, hop by hop.

The primary details captured in traceroute data include the IP addresses or hostnames of the intermediate hops and the round-trip latency (typically measured in milliseconds) taken to reach each hop. If a hop doesn’t respond within a timeout period, it is marked with an asterisk (*), which may indicate packet filtering, an unreachable node, or a silently configured router. By analyzing this data, one can assess how far packets get before they’re dropped or delayed, helping pinpoint problems like routing misconfigurations, overloaded links, or firewall restrictions.

Traceroute data is especially useful when evaluated over multiple iterations. When multiple traceroute results are collected over time to the same destination, it becomes possible to detect patterns and deviations in the network path. If each run shows identical hop sequences, it implies a stable and consistent network route. However, if the path varies—such as a node occasionally routing through different downstream hops—this indicates path fluctuation, often caused by dynamic routing protocols, load balancing, or temporary congestion.

Visualizing traceroute data through graphs makes these insights more intuitive. In such visualizations, each node or hop is represented by a specific symbol, conveying its role and performance. For instance, the source node might be shown as a blue circle and the destination as a yellow circle, providing clear anchors for the beginning and end of the route. Hops that respond normally appear as green circles, while red circles denote unknown or unresponsive nodes. Responders that show marginally high latency—yet still reply—may be marked with green or red triangles. These visual cues help in instantly recognizing both healthy and potentially problematic points in the network.

One critical observation in these visualizations is the structure of the connections. A straight line between nodes, with no branching, suggests that all traceroute iterations followed the same path without deviation—an indication of network stability. In contrast, when a node branches out to multiple downstream hops, it reveals that the path is not consistent. Different runs of the traceroute may have diverged from that point onward, which can point to an underlying routing inconsistency or adaptive network behavior. This branching is a sign of higher path variability and may warrant further examination, especially if the divergence coincides with latency spikes or performance degradation.

By closely analyzing traceroute data in this way, it becomes possible to identify where in the network issues such as delay, packet loss, or instability originate. This type of analysis is not only useful for troubleshooting but also plays a crucial role in maintaining the reliability and performance of large-scale distributed systems or services.

Juniper Business Use Only

In the analysis of traceroute data, the goal is to examine whether the hops (the intermediate nodes the data passes through) remain consistent across multiple iterations of the traceroute test or if there are fluctuations in the network path. The use of graphs with color-coded markers and line types helps visually represent the dynamics of the network path and allows for quick identification of inconsistencies, anomalies, or deviations from the expected behavior.

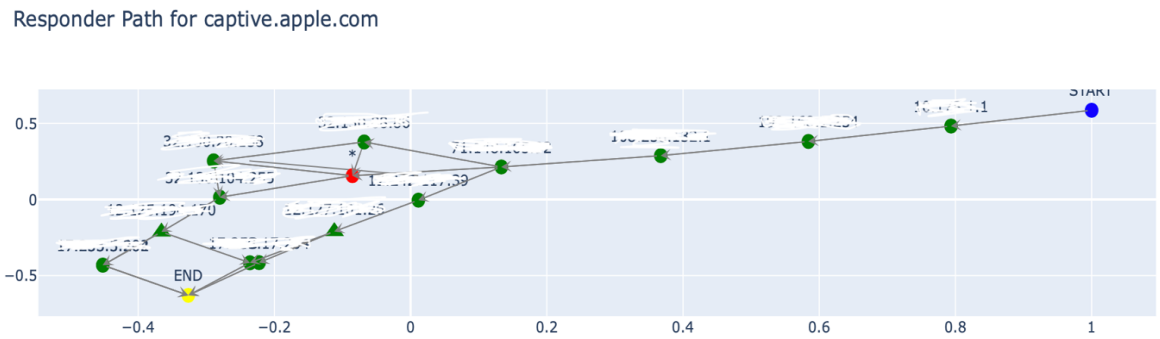

Fig 17. Graph representing the hops

Note: Ip addresses are scratched as it cannot be used in the report.

Visual Representation:

- **Blue Circle (Start - Source):** This represents the starting point of the traceroute, which is the source node or the origin of the network request. All traceroute iterations begin from this point, and it serves as the baseline for comparison of network performance.
- **Yellow Circle (End - Destination):** This marks the destination node, where the traceroute ends. Each traceroute will ultimately aim to reach this destination, and the analysis focuses on how consistently the traceroute reaches the destination via different hops.
- **Red Circle (Unknown Nodes):** These nodes are marked with red circles to indicate that the traceroute data for that specific hop is unknown or unavailable. The absence of a response from a particular node could indicate network congestion, firewall rules, or packet loss. Analyzing unknown nodes is crucial for understanding potential disruptions or obstacles along the network path.
- **Green Circle (Valid Responders):** These nodes are the intermediate hops that responded successfully to the traceroute request. The green circle represents valid and responsive network devices (such as routers or switches) that relay the data packet through the network. These are the normal nodes that contribute to a consistent path, and their presence ensures the traceroute path is functioning as expected.
- **Green/Red Triangle (Responders with Slightly High Latency):** These markers represent nodes that respond to the traceroute request but with higher-than-usual latency. The green/red triangle indicates that while the node is reachable and functioning, the latency may be somewhat elevated compared to other hops, suggesting potential slowdowns or issues in the path.

Juniper Business Use Only

- **Straight Line (No Branching):** This signifies that all iterations of the traceroute followed the same path through the network. The consistency of the path across all iterations suggests that the network connection is stable, and the hops from source to destination do not change. This absence of fluctuation in the path is a sign of a reliable and consistent network route with no unexpected detours.
- **Branching (Multiple Paths from a Node):** If a node has multiple branches stemming from it, it indicates that different iterations of the traceroute are following different paths. This branching behavior suggests fluctuations in the traceroute, meaning the data is taking varying routes at that particular hop. Multiple paths could be the result of network load balancing, dynamic routing adjustments, or issues with specific routers or links in the network. The appearance of branching indicates that the network path is not consistent, and the data may take different routes on different attempts, leading to potential variability in latency and network performance.

Analysis of Path Consistency:

The use of these visual markers allows for a straightforward analysis of hop consistency. If the majority of hops in the traceroute are represented by green circles and straight lines, this indicates that the network path is stable, and the hops are consistent across iterations. The path does not change, and the latency for each hop remains relatively steady. This is an indication of a well-functioning network where traffic follows the same route without unexpected deviations.

On the other hand, if there are branching points where multiple paths emerge from a hop, this suggests that the network path is fluctuating. Such branching can lead to variability in the traceroute results, as the traffic might take different routes depending on factors like network congestion, routing decisions made by intermediate devices, or even specific network failures. These fluctuations can result in inconsistencies in latency and might be a sign of network instability or inefficiencies in routing algorithms. For instance, if one path encounters congestion or failure, another path may be selected, causing the traceroute to behave unpredictably.

Moreover, red circles (unknown nodes) introduce another layer of complexity to the analysis. Unknown nodes, which do not respond to the traceroute, can signify issues like network congestion, routing failures, or configurations that block traceroute traffic (such as firewalls or filtering mechanisms). The presence of such unknown nodes often results in incomplete or missing data for those hops, making it more difficult to assess the true consistency of the network path.

Green/Red triangles indicating high latency at certain hops might suggest that while the path itself is consistent, some nodes or links along the way are experiencing performance degradation. This latency fluctuation could be indicative of network congestion, link instability, or temporary disruptions, and it can affect the overall performance of the network. Identifying these high-latency hops can help target specific areas for optimization, whether through hardware upgrades, network reconfiguration, or rerouting to bypass problematic nodes.

Juniper Business Use Only

Overall Insights:

By using graphs with these visual indicators, the analysis of hop consistency becomes clearer and more actionable. Straight-line paths with minimal branching and a majority of green circles signal a stable network, while branching paths and the presence of high-latency markers point to areas where network optimization or further investigation is necessary. The combination of these visual cues allows for a nuanced understanding of network performance, enabling more effective troubleshooting, route optimization, and performance monitoring.

In conclusion, analyzing the consistency of hops in traceroute data using visual markers such as circles and lines provides a powerful tool for understanding network stability. By observing patterns of straight lines versus branching, the appearance of unknown nodes, and latency spikes, network engineers and stakeholders can assess whether the network is stable or if certain hops or paths are exhibiting fluctuations that may need further attention. This type of visual analysis offers a clear, concise method for tracking network path consistency, which is crucial for maintaining a reliable and efficient network infrastructure.

My contributions made a substantial impact on the accuracy, efficiency, and reliability of the synthetic testing system by directly addressing critical challenges inherent in large-scale network diagnostics. One of the core areas I focused on was optimizing the process of synthetic test scope generation. The traditional methodology often involved generating exhaustive combinations of Access Points (APs), VLANs, and clients for each site, which led to redundant testing, unnecessary consumption of system resources, and longer test execution times. To address this, I implemented a strategic, data-driven approach that emphasized maximizing diagnostic coverage while minimizing overhead. By carefully analyzing site topologies, client distribution, and historical performance data, I was able to reduce redundancy in test generation and ensure that only the most relevant network elements were selected for testing.

This optimization included selecting only a single, representative VLAN per site, thus avoiding the overhead of running parallel tests on functionally identical network paths. Additionally, I designed logic to intelligently select just one AP per switch for scheduling, effectively balancing diagnostic depth with efficiency. For sites with missing or incomplete LLDP (Link Layer Discovery Protocol) data—where topology-based selections could be compromised—I incorporated fallback mechanisms using signal strength, client association metrics, and past test behavior to make reliable selections. These efforts significantly reduced the number of synthetic tests triggered per site while ensuring that the quality of testing remained high. As a result, the entire synthetic testing process became more agile and scalable, with fewer unnecessary operations and better use of computational and network resources.

In addition to optimizing test generation, I worked extensively on improving the runtime efficiency and performance of the testing framework. Synthetic testing at scale demands high throughput and responsive feedback loops, especially in enterprise deployments spanning thousands of devices. By removing redundant computation and refining the scope logic to eliminate overlapping test cases, I helped reduce execution latency and cut down system resource consumption. The system became better equipped to handle higher test volumes without slowing down or impacting other operations. Moreover, I

Juniper Business Use Only

implemented logic for adaptive test frequency—focusing more on sites that showed recent instability while reducing test frequency for consistently healthy sites. This adaptive behavior further streamlined performance, ensuring that resources were allocated where they were needed most, without compromising monitoring fidelity.

Beyond test design and execution, I took a proactive role in diagnosing and resolving the network anomalies that synthetic tests were meant to detect. Network issues such as latency spikes, packet loss, and DNS or ARP failures are often complex and can stem from routing asymmetries, configuration mismatches, or hardware-specific behavior. I developed a thorough diagnostic workflow that included both quantitative analysis and visual tools to investigate these anomalies. Using traceroute data collected during the tests, I employed visualizations that represented the path of each test iteration with intuitive markers. A blue circle indicated the source, yellow the destination, red circles denoted unreachable nodes or timeouts, while green circles represented valid responders. Green or red triangles were used to flag responders that exhibited slightly elevated latency, which could indicate emerging congestion or hardware stress.

By analyzing these visualizations, I was able to assess whether paths remained consistent across test runs or fluctuated due to dynamic routing or transient network behavior. For example, a straight, unbranched path across iterations implied a stable and consistent route, whereas branching at intermediate hops suggested routing changes or network instability. These insights were critical in pinpointing the underlying causes of performance anomalies and enabled faster, more targeted troubleshooting. In many cases, this led to corrective actions such as firmware upgrades, switch configuration updates, or architectural changes in network layout, which helped stabilize test results and improve the reliability of the infrastructure being monitored.

The overall effect of my contributions was the creation of a more intelligent, efficient, and resilient synthetic testing system—one that could not only detect issues but adapt to changing network conditions and scale seamlessly across large deployments. By enhancing both the logic of test generation and the diagnostic depth of the system, I ensured that testing remained both lightweight and insightful. My work reduced unnecessary resource utilization, shortened diagnostic cycles, and improved the long-term maintainability of the platform.

Juniper Business Use Only

8.Addressing Key Challenges faced in the Project

Throughout the course of this project, several technical and operational challenges arose, each requiring thoughtful analysis, iterative troubleshooting, and innovative problem-solving. These challenges spanned data engineering, network diagnostics, system efficiency, and interpretation of complex traceroute patterns. By addressing each issue methodically, I was able to strengthen the reliability, accuracy, and performance of the synthetic connectivity testing framework and its supporting analytics.

One of the primary challenges encountered was related to optimizing the test scope generation process. Given the vast number of potential test combinations across multiple Access Points (APs), VLANs, and test types, the initial scheduling logic often resulted in an excessive number of synthetic tests being generated. This not only strained backend resources but also introduced significant delays in test execution and result aggregation. To resolve this, I undertook a detailed analysis of site topology and test relevance criteria. By implementing logic to limit the number of VLANs to just one per site and selecting only a single AP per switch, I was able to intelligently reduce test volume without compromising test coverage. This optimization led to a more resource-efficient and faster testing process.

A related issue was the handling of sites with incomplete or missing LLDP (Link Layer Discovery Protocol) information. This data is critical for mapping switch-AP relationships, and its absence created ambiguity in determining which APs were actively connected to switches. To address this, I introduced fallback logic that inferred connectivity using alternate heuristics such as AP MAC address correlation and last-known switch port mappings. This ensured continuity of test scheduling even in environments with inconsistent topology metadata, thereby increasing robustness across site configurations.

The visualization of latency trends and hop consistency posed another significant challenge. Raw traceroute data, while rich in diagnostic detail, is inherently complex and difficult to interpret without a clear visual representation. Traditional command-line outputs were not scalable for multi-site comparisons or long-term trend analysis. To make the insights more accessible, I developed custom line chart visualizations to show latency behavior over time and across hops. For traceroute path consistency, I constructed path graphs where nodes and edges represented routers and transitions, with visual markers (colors and shapes) indicating hop roles, latency status, and unknown nodes. This visual approach not only clarified complex routing behavior but also enabled higher management and stakeholders to quickly identify areas of concern.

Another major challenge was identifying and managing latency outliers. Synthetic test results often showed spurious spikes in latency due to transient network events, noise in measurement, or test environment constraints. Manually filtering through this data was impractical. To address this, I employed statistical analysis techniques such as standard deviation-based filtering and percentile thresholds to flag genuine outliers. This allowed for consistent identification of problematic patterns, such as specific VLANs showing elevated DNS or DHCP latency, or curl requests to particular URLs

Juniper Business Use Only

exhibiting systemic delay. This data-driven approach reduced false positives and enhanced the accuracy of diagnostics.

Ensuring reliability of test results in fluctuating network conditions was another area that demanded constant attention. Certain environments exhibited high path entropy, with routing paths dynamically changing due to BGP fluctuations or internal policy routing. These changes resulted in inconsistent traceroute data and occasional test failures. To manage this, I introduced periodic checks to validate expected paths, and used multiple iterations of the same test to distinguish between transient anomalies and persistent issues. This helped reduce the impact of network volatility and improved the reliability of conclusions drawn from traceroute outputs.

Scaling data processing pipelines for synthetic test results also introduced its own set of difficulties. With thousands of tests running per hour across hundreds of sites, the volume of result logs processed daily was substantial. Initial versions of the data processing pipeline struggled with performance bottlenecks during result ingestion and aggregation. By refactoring the Spark job logic, applying partitioning strategies, and fine-tuning caching and memory parameters, I improved overall processing efficiency and reduced pipeline run time. This ensured timely delivery of actionable insights and prevented backlog accumulation in production environments.

Another subtle but important challenge was the interpretation of hop data where certain routers suppressed ICMP TTL-exceeded responses. These silent hops (often represented by "*") created gaps in the traceroute path that could be mistaken for failures. To avoid false alarms, I designed the path visualizations to clearly indicate unknown nodes using red markers and incorporated logic to differentiate between unreachable hops and hops likely to be rate-limiting ICMP. This nuanced understanding helped ensure accuracy in path diagnostics and reduced misclassification of network health.

Lastly, a significant challenge was the communication of technical insights to a non-technical audience, including product teams and senior leadership. Network behavior, especially concepts like BGP path changes, traceroute fluctuations, and latency spikes, can be highly abstract to those outside engineering. To bridge this gap, I focused on building simple, intuitive visualizations and reports. Each graph or chart was supplemented with annotations and summaries, emphasizing the business impact rather than just raw metrics. This approach helped secure stakeholder buy-in for deeper infrastructure improvements and enabled data-backed decision-making across teams.

In summary, the challenges faced throughout the project ranged from algorithmic inefficiencies to data inconsistencies and interpretation complexity. Each was addressed with a combination of analytical rigor, creative design, and systematic engineering improvements. These problem-solving efforts played a critical role in improving the robustness, accuracy, and scalability of the synthetic connectivity test analysis pipeline, ultimately contributing to a more reliable and cost-effective network diagnostics framework.

Juniper Business Use Only

9.Conclusion

This project on synthetic connectivity testing and traceroute analysis has been an immensely rewarding learning experience, enabling me to enhance my understanding of network performance analysis and expand my technical skill set. The primary focus on analyzing connectivity test results, investigating latency patterns, detecting outliers, and examining hop path dynamics gave me a deeper insight into the complexities of network behavior. The process not only improved my ability to detect and diagnose network issues but also highlighted the critical importance of performance monitoring and troubleshooting in maintaining the efficiency and reliability of network infrastructures.

Through this project, I gained hands-on experience with several advanced technologies. My work with PySpark to process large datasets and extract valuable insights from traceroute data has reinforced my understanding of distributed computing and big data technologies. This experience helped me realize the power of tools like Spark for scaling data processing, particularly when dealing with large and complex datasets. Additionally, by utilizing Python libraries like Pandas, NumPy, and Matplotlib, I was able to streamline the analysis and visualization of results, which enhanced my ability to present network data in clear, actionable formats. This was further complemented by my exploration of Airflow for automating workflows, which gave me a deeper appreciation of orchestration and scheduling in data engineering.

The project also provided significant exposure to network diagnostics and performance analysis, which was crucial for understanding how different network protocols, such as DNS, DHCP, and ARP, influence connectivity. By analyzing hop paths, round-trip times (RTT), and packet loss across multiple tests, I developed a more comprehensive understanding of how real-world network infrastructures perform under varying conditions. This aspect of the project particularly sharpened my ability to diagnose and troubleshoot network-related issues, an essential skill in any network-focused role.

Working with the Marvis Minis product at Juniper Networks and utilizing Mist AI technology gave me an additional layer of knowledge in synthetic testing and AI-driven network management. I learned how to leverage these tools to automate test result analysis, detect network issues proactively, and optimize wireless network performance. This hands-on experience introduced me to the practical applications of AI in modern network management and its potential to revolutionize how network performance is monitored and improved.

Finally, the process of compiling comprehensive reports and presenting findings through visual aids like pie charts and bar charts allowed me to refine my technical writing and reporting skills. Being able to effectively communicate complex technical information to diverse audiences is a critical aspect of any data-driven project, and this experience has significantly strengthened my ability to present data in a coherent and understandable manner.

In conclusion, this project has been an invaluable opportunity to explore a wide range of tools and technologies, including distributed computing, network diagnostics, automation, and AI-based solutions. The skills I developed in analyzing network performance, automating workflows, and creating actionable reports will undoubtedly

Juniper Business Use Only

serve as a strong foundation for my future work in network analytics, big data, and beyond. The knowledge gained throughout this project will continue to influence my approach to problem-solving, ensuring that I am equipped to tackle complex challenges in the field of network management and data analytics.

Juniper Business Use Only

Bibliography

[1]https://www.juniper.net/documentation/us/en/software/mist/mist-aiops/topics/topic-map/marvis-minis-overview.html

[2]https://sparkbyexamples.com/spark/what-is-spark-job/

[3]https://pratikbarjatya.medium.com/demystifying-spark-jobs-stages-and-tasks-a-simplified-guide-f35da5ab4aa6

[4]Trammell, B., et al. (2014). *Active vs. passive network monitoring: A comparison of methodologies*. In Proceedings of the 2014 ACM SIGCOMM Conference. ACM.

[5] Jain, R., et al. (2013). *Espresso: A Cloud-scale Network Testing Platform*. In Proceedings of the 2013 ACM SIGCOMM Conference. ACM.

[6]Chen, Y., et al. (2014). *PingMesh: A Distributed Active Monitoring Platform for Data Center Networks*. In Proceedings of the 2014 ACM SIGCOMM Conference. ACM.

[7]Singh, A., et al. (2020). *Dynamic Test Orchestration for Intelligent Test Scope Selection*. In Proceedings of the 2020 IEEE International Conference on Network Protocols. IEEE.

[8]Krishnamurthy, B., & Wills, C. (2001). *Measuring the Web: A Comparative Study of Active and Passive Measurements of Web Reachability*. ACM Transactions on Internet Technology, 1(1), 45-67.

[9]Zhang, X., et al. (2019). *Cloud Outage Detection via Active Probing*. In Proceedings of the 2019 ACM Internet Measurement Conference. ACM.

[10]Juniper Networks. (2021). *Mist AI and Marvis Minis: AI-Driven Network Insights for Proactive Synthetic Testing*. Retrieved from [Juniper Networks website].

Juniper Business Use Only