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

NIDHI S
231058001

Master of Engineering
ME (Big Data Analytics)

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

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.

Index

List of Figures

Introduction

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.

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:

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.

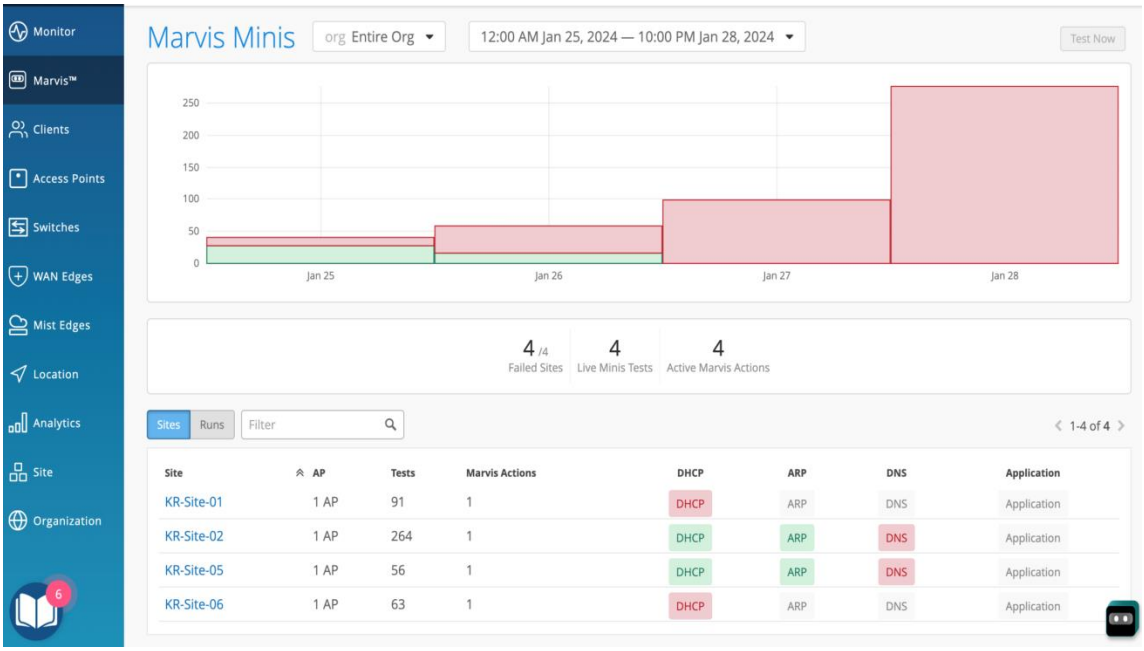

Fig 1. Org Level Marvis Minis Dashboard

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.

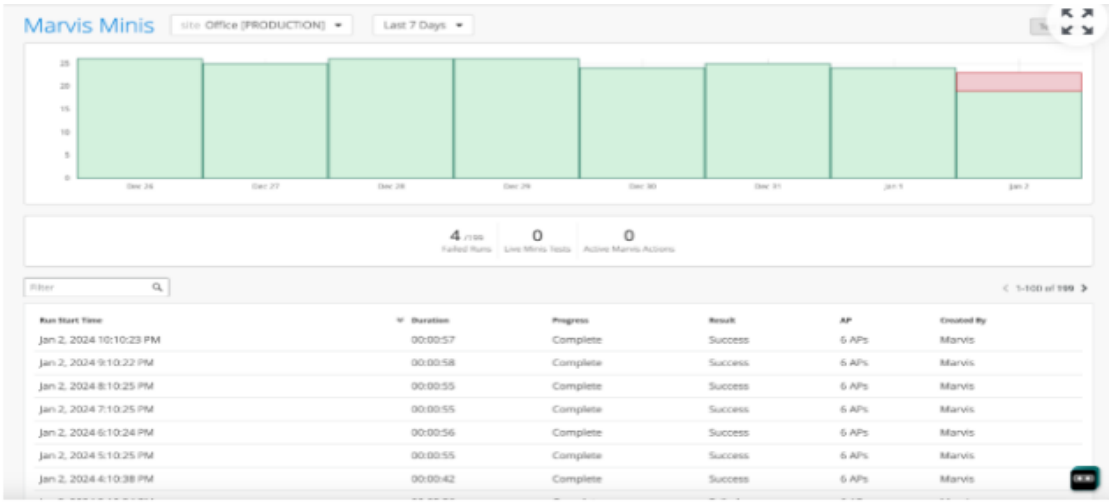

Fig 2. Site Level Dashboard

The Created By column indicates who initiated the validation:

- Marvis: Indicates that Marvis initiated the validation automatically
- 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.

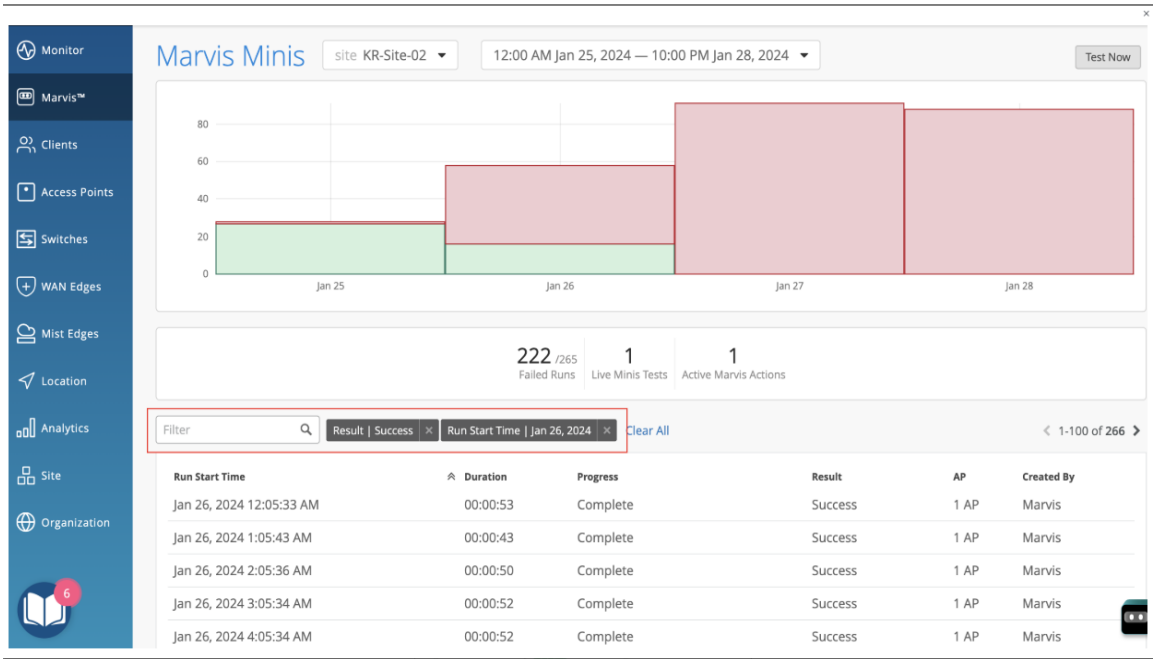

Fig 3. Filtered results for tests run on a specific date Dashboard

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.

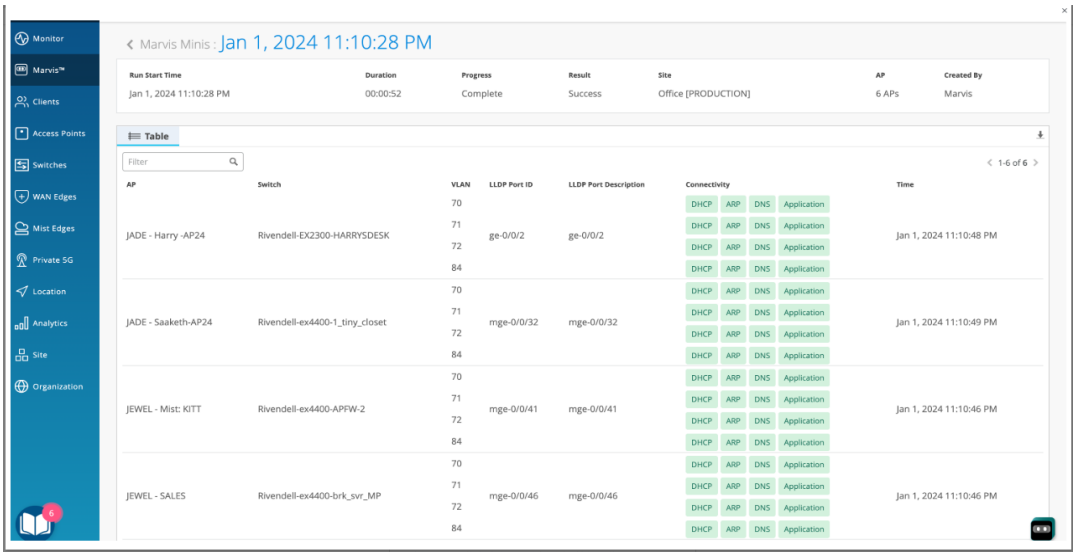

Fig 4. Information on each validation Dashboard

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.

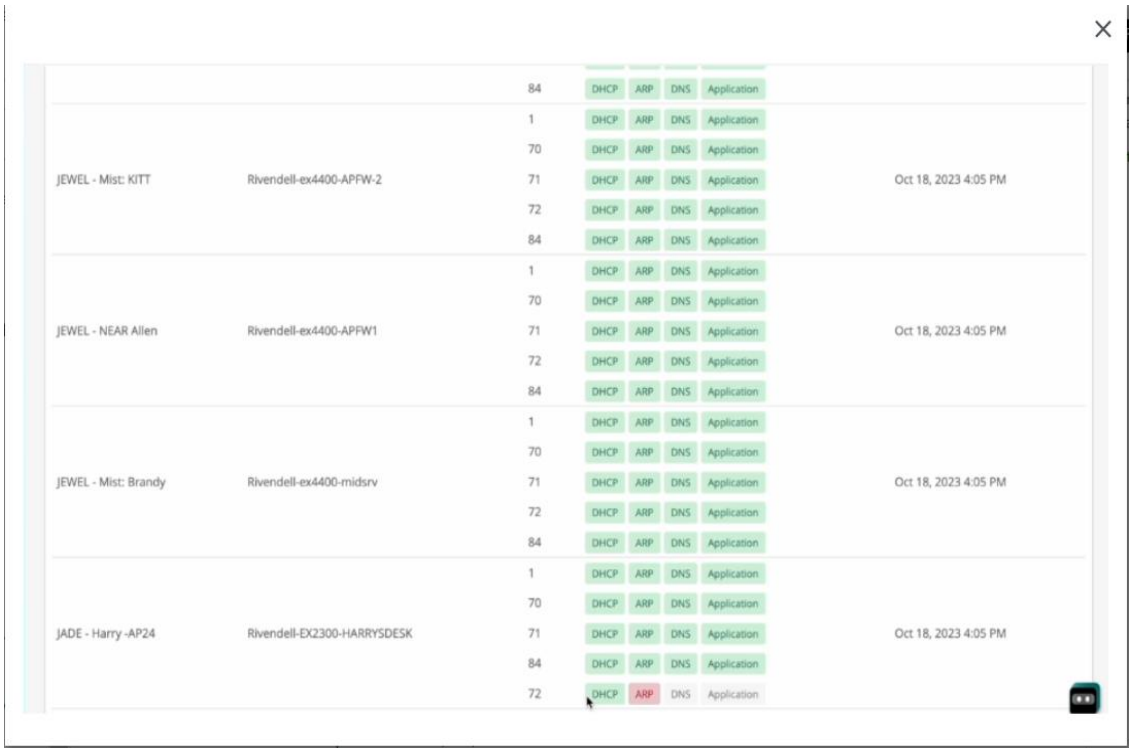

Fig 5. Example of a validation that detected an ARP failure on one of the Aps Dashboard

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.

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.

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.

Methodology

Technology Used

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 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, 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.

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.

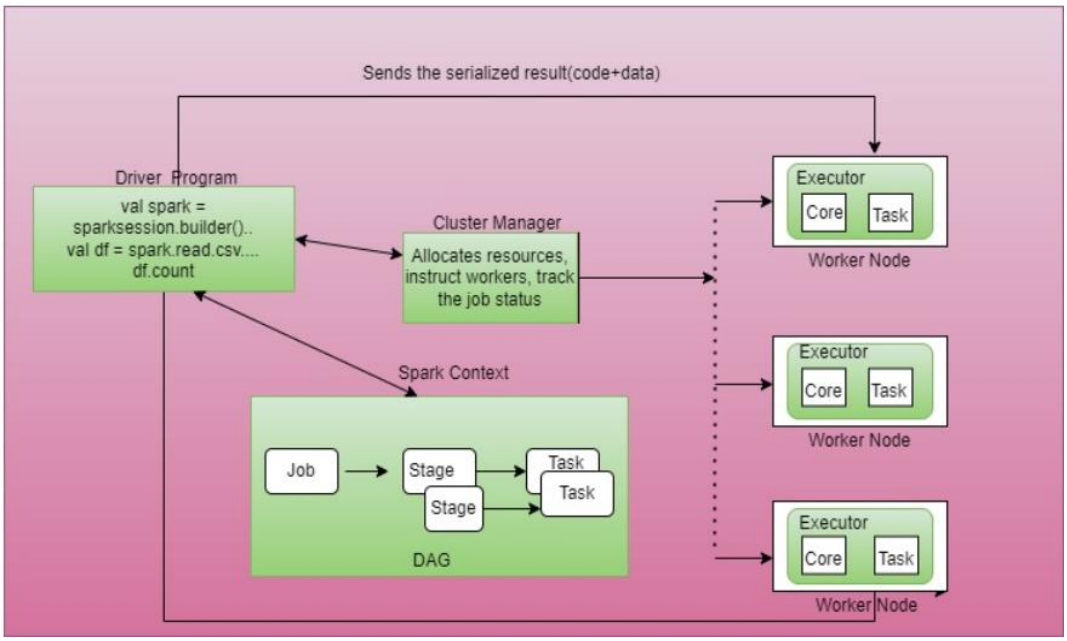

Fig 6. Spark Job Execution

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.

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.

Minions Service Overview

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.

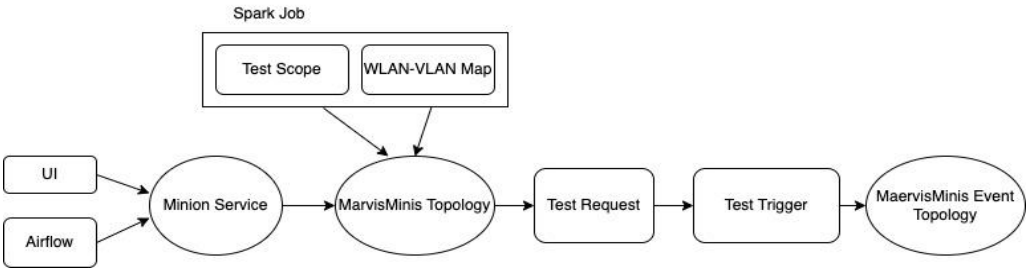

Fig 7. Minis Architecture Diagram

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.

On the other hand, 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.

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.

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.

Work Done

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.

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.

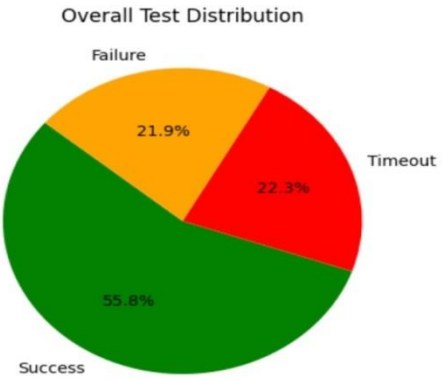

Fig 8. Pie chart representing Test Outcome Distribution

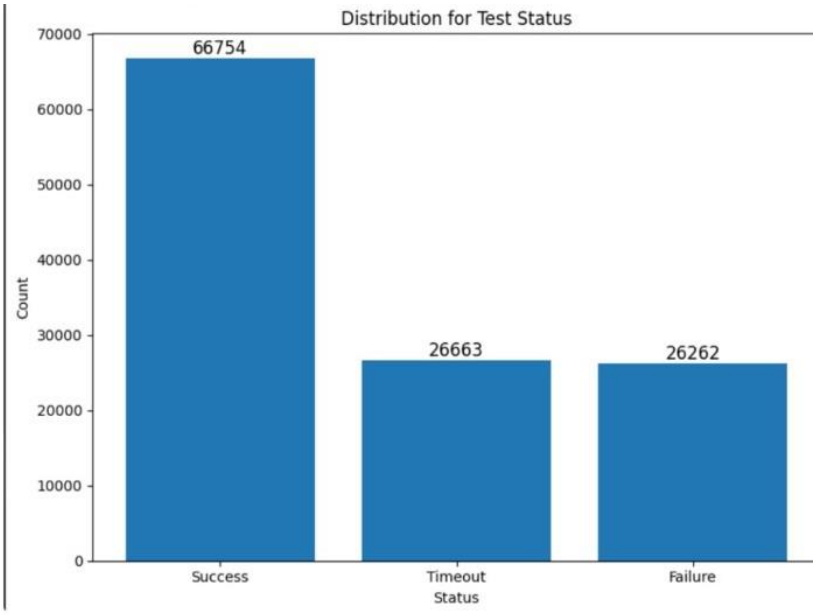

Fig 9. Bar chart represents the Unique test count by Test status

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

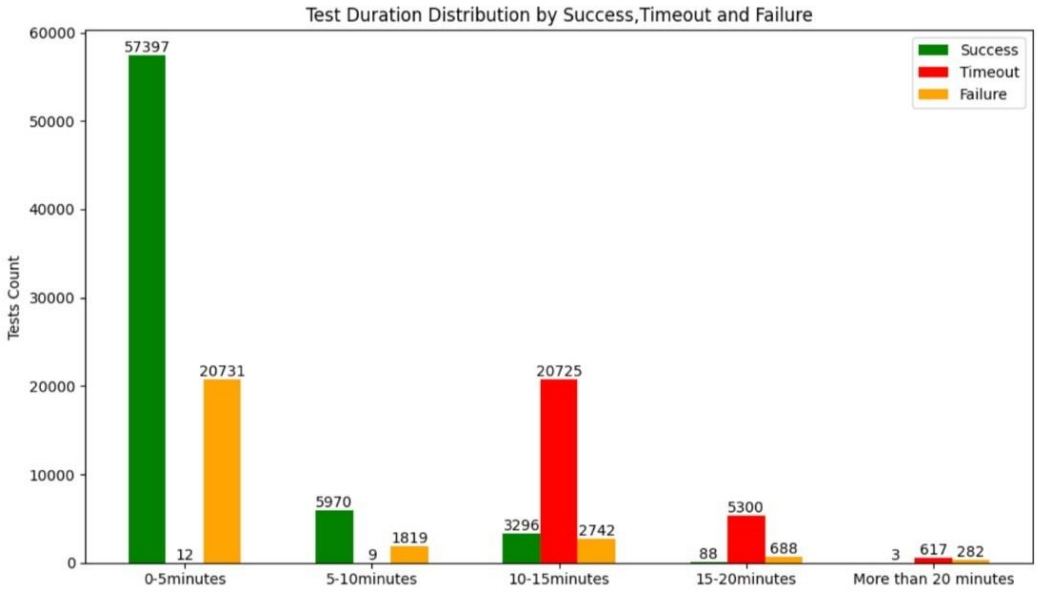

Fig 10. Bar chart representing the test duration distribution for the successful, failed and timed-out tests

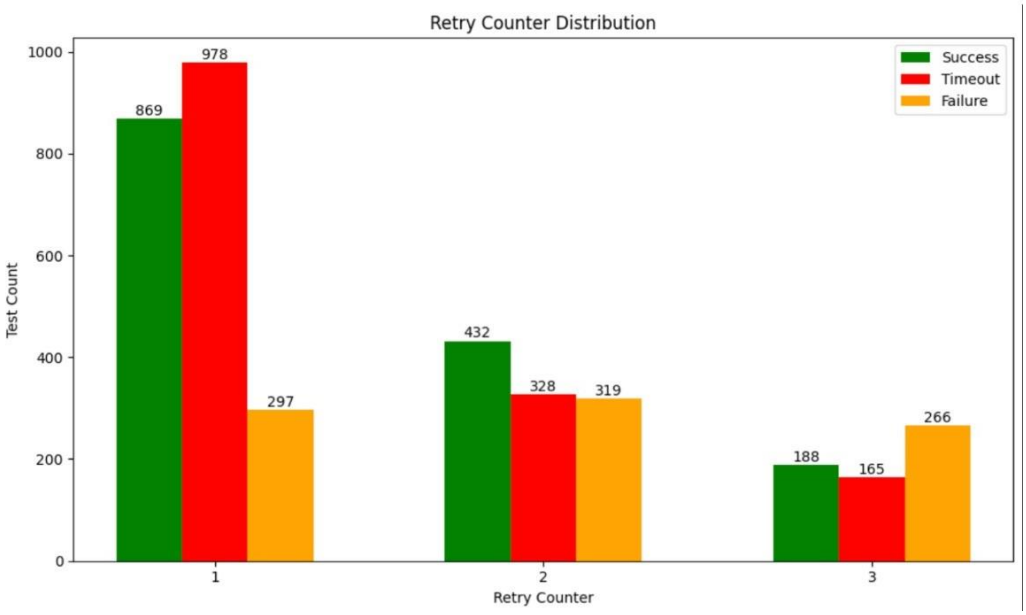

Fig 11. Bar representing the Retry counter distribution

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.

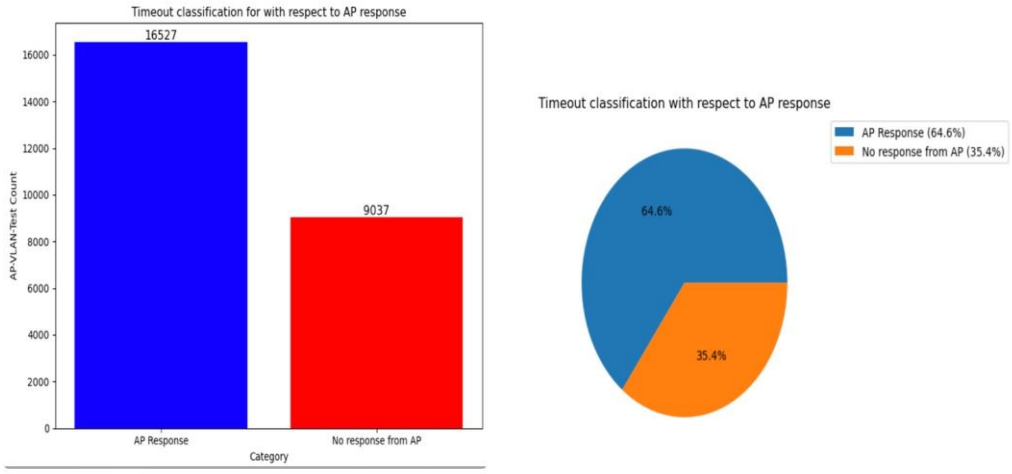

Fig 12. Bar and Pie chart representing the test count with and without AP response

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.

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.

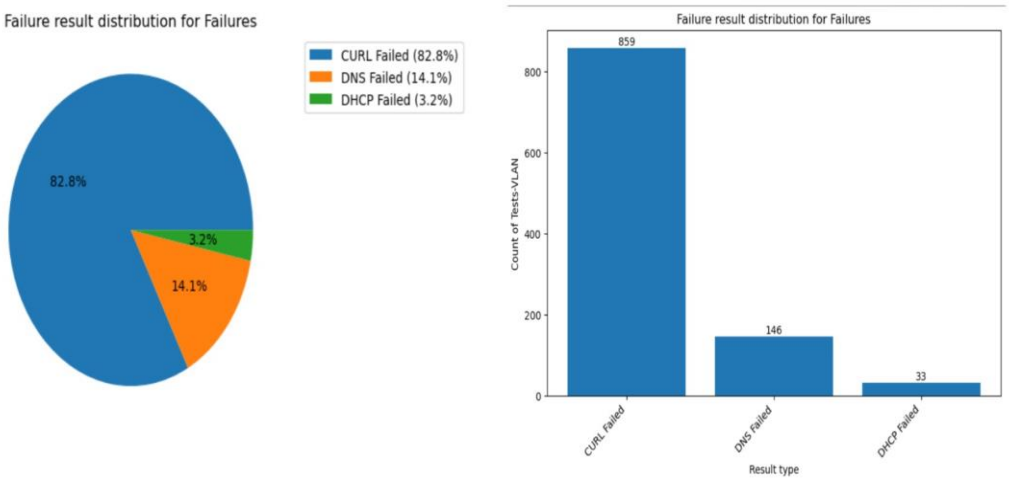

Fig 13. Pie and Bar chart representing the Failure result distribution

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.

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.

My contributions significantly improved the accuracy, efficiency and reliability of synthetic tests by addressing key challenges in the testing process. I focused on optimizing the test scope generation process, ensuring that it delivered maximum coverage without overburdening system resources. By carefully analysing the testing requirements and implementing smart strategies for test design, I was able to reduce redundant operations and unnecessary resource consumption, which led to more efficient and effective test execution. This approach not only enhanced the overall performance of the testing framework but also ensured that the tests were both comprehensive and cost-effective.

Additionally, my efforts in troubleshooting and analysing network-related issues were critical in maintaining the system's reliability. Network problems can often be elusive and disruptive, but through detailed diagnostics, I was able to quickly pinpoint the

underlying causes of instability or failures. By systematically addressing these issues, I ensured that the testing process could continue smoothly, even in the face of dynamic and unpredictable network conditions. This attention to detail and proactive problem-solving significantly reduced downtime and increased the reliability of the testing outcomes.

Ultimately, my work contributed to building a more robust and effective testing process that supported the stability and performance of the system. By enhancing both the technical execution of the tests and the underlying infrastructure, I helped ensure that the system was capable of delivering reliable, high-quality results in a variety of complex scenarios. These improvements not only streamlined the testing workflow but also contributed to the long-term scalability and sustainability of the system, making it better equipped to handle future challenges.

Future Work

In the future, my work will focus on a comprehensive and detailed approach to latency analysis and optimization, which will involve multiple layers of investigation and a range of methodologies to identify and resolve issues within the network.

Analysing trends in latency values to identify normal behaviour patterns over time: In the future, I will focus on tracking and analysing latency values over extended periods to establish normal behaviour patterns for the system. This trend analysis will allow me to understand how latency fluctuates under typical conditions and provide a baseline for identifying anomalies. By regularly monitoring these trends, I will be able to recognize any significant deviations that require deeper investigation, ensuring that performance issues are detected early before they escalate.

Identifying and flagging outliers in latency values for deeper investigation: Once I have established the normal latency patterns, I will develop methods for flagging latency values that significantly deviate from these norms. These outliers will be flagged for further investigation to understand their root causes, which could range from network congestion to hardware malfunctions or configuration errors. By proactively identifying outliers, I can ensure that potential issues are addressed before they disrupt the network's overall performance.

Analysing DHCP, DNS, ARP and CURL latency values for detailed insights: A crucial part of my future work will involve analysing latency across key network protocols such as DHCP, DNS, ARP and CURL. These protocols are fundamental to network communication and their latency values provide critical insights into system performance. For instance, analysing DHCP latency could reveal issues with IP address allocation, while DNS or CURL latency could indicate server-side problems or network congestion. This detailed analysis will help isolate protocol-specific issues and optimize network efficiency.

Performing the latency analysis on a per-URL basis to uncover potential URL-specific issues: In addition to protocol-level analysis, I will conduct latency analysis on a per-URL basis to identify any issues that may be specific to certain websites or endpoints. Certain URLs may experience consistent latency issues, indicating server overloads, routing inefficiencies or external dependencies. By isolating these URL-specific problems, I can implement targeted solutions to improve performance for those specific resources, ensuring that the overall system remains responsive.

Investigating latency patterns across different VLANs to detect any anomalies tied to VLAN-specific configurations: Latency behaviour may vary significantly across different VLANs due to factors such as network configuration, traffic loads or hardware setup. I will analyse latency patterns across VLANs to detect any anomalies tied to VLAN-specific configurations or performance issues.

Extending the analysis to a per-site level to identify site-specific trends and outliers: Networks with multiple sites may experience different latency behaviours based on local infrastructure, geographic location and configuration. I will extend the analysis to a per-site level to detect site-specific trends and outliers. This approach will help identify issues that are unique to specific sites, such as regional network congestion or local hardware malfunctions. By understanding site-specific latency behaviours, I can implement solutions tailored to each location’s needs, improving performance across the entire network.

Using statistical methods like mean, variance and standard deviation to identify outliers: To enhance the accuracy of identifying latency outliers, I will apply statistical methods like mean, variance and standard deviation. These methods will provide a quantitative basis for identifying which latency values are abnormal and require further investigation. By calculating the expected range of latency values based on historical data, I can set thresholds for what constitutes a significant deviation, ensuring that only truly concerning outliers are flagged for further action.

Correlating latency outliers with other factors, such as network load, AP model or firmware version to determine potential causes: After identifying latency outliers, I will correlate them with other network factors such as network load, specific AP models or firmware versions. This analysis will help determine whether external factors like high traffic, certain hardware configurations or out-dated firmware are contributing to latency spikes. For example, if high latency is consistently observed with a particular AP model or firmware version, it may indicate the need for an update or configuration change to improve performance.

Using visualization techniques to map latency trends and highlight outliers for better interpretation and communication of findings: To make my analysis more accessible and actionable, I will use visualization techniques to map latency trends and highlight outliers. Graphs, heatmaps and charts will provide a clear visual representation of latency patterns, making it easier to interpret complex data. These visualizations will also facilitate communication with higher management, allowing them to quickly understand the impact of latency issues and the areas that require attention.

Preparing detailed reports of the analysis, including identified outliers, trends and their potential impact on system performance: After completing the latency analysis, I will prepare detailed reports that summarize the findings, including identified trends, outliers and their potential impact on system performance. These reports will document the root causes of latency issues and provide actionable recommendations for corrective actions. The reports will serve as a comprehensive reference for future troubleshooting and help and ensure that the network remains optimized, offering a clear roadmap for improving system performance and stability.

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