KENYATTA UNIVERSITY

SCHOOL OF PURE AND APPLIED SCIENCES

DEPARTMENT OF COMPUTING & INFORMATION TECHNOLOGY

A MULTI-AGENT MONITORING SYSTEM FOR COMPUTER NETWORKS

AMWAYI HARRISON MAKOKHA

J562/21116/2021

SUPERVISOR: DR ABRAHAM MUTUA

A RESEARCH PROJECT REPORT SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF MASTER OF SCIENCE IN COMPUTER SCIENCE IN THE SCHOOL OF PURE AND APPLIED SCIENCES OF KENYATTA UNIVERSITY

SEPTEMBER 2025

DECLARATION

I declare that this project report is my original work and has not been presented in any other university/institution for consideration of any certification. This research project report has been complemented by referenced sources duly acknowledged. Where text, graphics, pictures, figures or tables have been borrowed from other sources, including the internet, these are specifically accredited and references cited using current APA system and in accordance with anti-plagiarism regulations.

Signature_____ **Date**: _____

Name: Amwayi Harrison Makokha

Registration number: J562/21116/2021

Department: Computing and Information Technology (CIT)

Supervisors' declaration: This project report has been submitted with our/my approval as University Supervisor.

Signature _____**Date**_____

Name: Dr Abraham Mutua (PhD - IT, Msc. IS, BSC IT)

Department: Computing and Information Technology (CIT)

University: Kenyatta University

DEDICATION

I dedicate this project to my loving Mother Hellen Kavene Amwayi. The Project would not have been possible without her support, encouragement and inspiration. Thank you and may God bless you.

ACKNOWLEDGMENT

This project's origin, development and successful conclusion are all due to several people, without the help of whom the intellectual journey would not have been possible. I give thanks to the Almighty God for providing me with the chance, good health and resources I need to carry out the study. I also want to express my gratitude to my supervisor, Dr. Abraham M Mutua for his constant support, wise counsel and willingness to share his extensive expertise and experience with this project.

I am deeply grateful to my lecturers for their invaluable knowledge and guidance throughout this study. Lastly, I sincerely appreciate my classmates for their contributions, which have enriched the development of this research.

ABSTRACT

In today's business environment, reliable network infrastructure is critical for day-to-day operations. As networks grow in complexity, efficient monitoring systems are essential to ensure the performance and availability of network devices. This study investigates the use of multi-agents in monitoring computer network devices, focusing on the application of autonomous agents in gathering and analyzing network data using the Simple Network Management Protocol (SNMP). It explores existing multi-agent frameworks and their limitations, followed by the design and implementation of a new model that integrates SNMP agents and Apache Kafka for scalable data ingestion and processing. The developed solution is designed to address concerns of scalability and network congestion, typically associated with centralized monitoring systems. By leveraging on Apache Kafka as a distributed messaging system, polling tasks are dynamically assigned across multiple agents, ensuring load balancing and fault tolerance. Additionally, feature selection is employed to reduce latency and minimize network congestion. The system was tested to evaluate its performance in real-time monitoring scenarios, demonstrating improvements in scalability and efficiency. This research concludes that multi-agent systems, combined with Apache Kafka, provide a robust model for real-time monitoring of network devices, offering enhanced scalability and reduced latency compared to traditional centralized approaches. Future work may involve refining fault-tolerance mechanisms and exploring additional autonomous agent models for other network management tasks.

TABLE CONTENTS

TABLE OF FIGURES

ACRONYMS AND ABBREVIATIONS

QoS	Quality of service
MAS	Multi-agent system (MAS)
SNMP	Simple network management protocol
USM	User-based Security Model
SSH	Secure Shell Protocol
TLS	Transport Layer Security
WSNET	Web Services based Network Management
SOAP	Simple Object Access Protocol
XML	Extensible Markup Language
JSON	JavaScript Object Notation
UDP	User Datagram Protocol
NMS	Network Management Stations
CORBA	Common Object Request Broker Architecture
DSR	Design Science Research
MIB	Management Information Base
NMS	Network Management Station

CHAPTER ONE

INTRODUCTION

1.1 Background of the Study

Efficient network management and monitoring technologies are crucial for optimizing resource utilization in computer networks, particularly as their scale and complexity continue to expand (Arzo, Akhavan, Esmaeili, Devetsikiotis, and Granelli, 2022; Michel, and Keller, 2017). With the widespread adoption of network technologies, ensuring high performance, enhancing service quality, troubleshooting issues and maintaining network security have become critical tasks (West, 2022).

Network monitoring plays a vital role in addressing these challenges by providing insights into traffic flows, resource utilization and other key performance metrics (Arzo et al., 2021). Effective management requires comprehensive analysis of monitoring data, as decision-making becomes challenging without a clear understanding of network conditions (Tso, Jouet and Pezaros, 2016). Logging and analysis techniques are fundamental to this process, enabling administrators to detect anomalies, optimize performance and enhance security.

To meet the growing demands of various industries, network monitoring and control systems, as well as process automation strategies are constantly evolving (Arzo, Bassoli, Granelli, and Fitzek, 2021). Modern organizations rely heavily on the Internet as an essential component of their daily operations. Even a few hours of network disruption can result in substantial financial and operational losses (Ilany, Booms, & Holekamp, 2015).

Network management involves collecting and evaluating data from interconnected devices to monitor and manage their performance (Li, He, Ming, and Cai, 2015). A network monitoring system continuously scans a computer network for slow or malfunctioning components, gathers critical information and alerts network administrators in the event of outages or other network-related issues through email, Short Message Service (SMS) or other notification mechanisms (Ilany et al., 2015).

1.1.1 Multi-Agents System

Agents are flexible, autonomous entities that make informed choices based on their knowledge and expertise. In order to solve a problem more effectively than a single agent could, a multi-agent system (MAS) consists of several, intelligent and networked cooperating agents. In distributed, complex and dynamic problem-solving contexts, multi-agent systems work extremely well, and using autonomous agents allows for intelligent decision-making. Based on an agent's code base and the principal on whose behalf it is acting, agent servers limit the amount of local resources that an agent can access, including disc space, network ports, and files (Wang, Jin, Jiang, Hu, and Wang, 2022).

According to Woodall, Zhao, Paynabar, Sparks, and Wilson (2017), the Multi-Agent System (MAS) methodology offers a strong foundation for modelling and resolving real-world issues. Mechanisms for agent administration, agent communication and agent directory maintenance are provided by a multi-agent system (Tsai, Tsai, Hsu, and Yang, 2018). The multi-agent approach is based on distributed management and on delegation.

1.2 Statement of the Problem

Network administrators find it extremely difficult to monitor sophisticated computer networks (Baer et al., 2016). In today's dynamic network environment, effectively managing a computer network has become crucial (Li et al., 2015). According to West (2022), if automated network monitoring solutions that can execute several network services concurrently in a single dashboard are implemented to supplement the human monitoring of network environments, network managers would be able to carry out their assigned tasks more efficiently.

The two major concerns when using simple network management protocol (SNMP) agents as network monitors are efficiency and scalability. Scalability pertains to the quantity of managers and agents, whereas efficiency is tied to the speed at which data is processed and delivered. Numerous networking conferences have examined the issues of managing massive amounts of data or compared SNMP's performance to other management protocols in order to assess its performance. Many have come to the conclusion that SNMP was ineffective at gathering detailed information owing to network latency that arises from concentration of traffic around the centralized network management station. Consequently, the majority of suggestions for performance enhancement focused on enhancing the network's characteristics or optimising its algorithms. Web Services based Network Management (WSNET), a completely new paradigm, was proposed by Soldatos et al. (2007). WSNET implements agents and managers via the web using a decentralised methodology. It uses Extensible Markup Language (XML) technology in conjunction with Simple Object Access Protocol (SOAP) in a SOAP engine to communicate data. This was

further improved by Kasula, 2017 when he showed that multi-agent system technology combined with JavaScript Object Notation (JSON) data interchange format and sockets significantly reduced network latency and hence efficiency in network monitoring. It is quite evident most research has primarily focused on enhancing SNMP polling mechanisms while overlooking broader considerations of scalability and efficiency. As networks grow in size and complexity, optimizing monitoring strategies beyond just polling is crucial to ensuring seamless performance and resource utilization.

The purpose of this study therefore was to investigate the use of a multi-agents in monitoring computer network devices and develop an efficient model to demonstrate the suitability of autonomous agents in providing an efficient and scalable monitoring solution. The study specifically looked at Simple Network Management Protocol (SNMP) and Apache Kafka to enhance scalability as data ingestion engine.

1.3 Main Objective

The main objective of this study was to investigate the application of a multi-agent systems in network monitoring and develop a model to demonstrate the suitability of autonomous agents in monitoring computer networks.

1.3.1 Specific Objectives

1. To Investigate existing Multi-agent based systems used in monitoring computer networks

2. To design and develop a Multi-agent based model prototype that will overcome the identified weakness.
3. To test the effectiveness of the developed model.

1.4 Research Questions

1. Which Multi-agent based techniques are used to monitor computer networks?
2. How will the Multi-agent based system model prototype be developed?
3. How effective are the results and findings from using the developed model?

1.5 Significance of the study

The study proposed a model that helps to enhance performance monitoring within computer networks hence enhancing performance and service delivery.

For Systems and Network administrators, the study will assist in improving the quality and efficiency of their work by eliminating the inconsistent techniques used in monitoring and management of network nodes/devices.

For end users, this study will assist in refining the techniques used for identifying possible problem areas and hence better quality services.

1.6 Scope

This project focused on the design, implementation and evaluation of an SNMP-based network monitoring system integrated with Apache Kafka for real-time data streaming. The primary scope included:

- Network Device Monitoring: The system targeted polling of SNMP-enabled devices to collect key management information base (MIB) values such as system uptime, location, and interface statistics.
- Kafka Integration: Kafka is utilized as the messaging backbone to stream the collected SNMP data to consumers for further analysis or storage.
- Scalability Testing: The report explored multi-threaded polling techniques to ensure scalability across a large number of network devices.

CHAPTER TWO

LITARATURE REVIEW

2.1 Introduction

This chapter will review existing literature relating to the study. It will specifically address techniques for monitoring computer networks, Multi Agent systems, Simple Network Management Protocol, Apache Kafka, agent characteristics and Multi Agent systems techniques.

2.2 Theoretical Framework

This study considered decision theory and utility theory. Decision theory is a collection of ideas and associated analytical methods with varying degrees of formality that help decision makers select options based on potential consequences (Lee, 2011). Utility theory is an analytical technique that helps decision-makers decide what to do when presented with a variety of criteria to choose from. Utility theory dates back to the eighteenth century, and since the early twentieth century, it has seen tremendous development (Fishburn 2020).

In multi-agent systems, each agent typically has its own goals, preferences, and constraints. Decision theory helps in formalizing the decision-making process of individual agents by considering factors such as uncertainty, risk and preferences. It provides a set of principles and mathematical models for rational decision-making under uncertainty hence Decision theory involves the application of reason in human decision-making and provides a framework for analyzing which options should be chosen when the outcomes of those options are uncertain (Raiffa, 2018). Utility theory, on the other hand is a key component

of decision theory and is of particularly relevance in multi-agent design. Utility theory enables agents to express their preferences and make decisions based on the expected utility of different actions or outcomes. Agents can assign utilities to various states or outcomes and use them to compare and rank alternative choices.

2.3 Multi Agent Systems (MAS)

A network of computational agents that interact and communicate with each other forms a multi-agent system (MAS). MAS includes an Intelligent Distributed Artificial System, which incorporates social aspects and human reasoning to solve problems (Comer, 2018). A multi-agent system, according to Herrera, Pérez-Hernández, Kumar Parlikad, and Izquierdo (2020), is a collection of agents that monitor their surroundings and take action to accomplish predetermined goals. Therefore, MAS can be defined as a network of interconnected software modules that integrates disparate systems to manage complicated tasks collectively that exceed the capabilities of any single system on the network.

According to Yang and Chang (2011), the foundation of multi-agent systems is the notion that cooperative work environments can handle challenging problems that are beyond the scope of the conventional centralised method to computation. In order to address problems more effectively, agents are utilised to interact in a flexible and dynamic way. Agents are able to function without direct human or external interaction. One of the most effective technologies for creating and implementing distributed complex systems is agent-based computing (Hashemi, Abdelghany, and Abdelghany, 2017).

According to Davarzani, Granell, Taylor, and Pisica (2019), agents in a multi-agent system communicate with one another in order to achieve a common objective. Additionally, agents are able to sense their surroundings and react quickly to changes in them. Since the beginning of intelligent systems, agents and MAS have been essential to their design. Their long-standing relationship with logic-based technologies may provide new avenues for the development of explainable intelligent systems (Aboubakar et al., 2022). According to Jayaramaiah et al. (2012), multi-agent systems are built on a society of distributed autonomous, cooperative entities, each with a specific role, set of knowledge and abilities, and a localised perspective on the world. These entities also have simple rules governing their behavior.

2.3.1 Dimension Reduction and Filtering

One of the main requirements for MASs is the ability to filter out irrelevant data gathered over time or abstract the entire data stream (Herrera et al., 2020). This can significantly shorten the main system's computation time and is crucial in distributed systems like the Internet of Things. These methods perform well for classifying and grouping system status data as well as for quick comparison with the target network.

Deciding if a fault has occurred is the task of fault detection. Fault identification, fault diagnosis, and process recovery are the procedures related to monitoring, according to Michel and Keller (2017). The process of fault identification involves determining which observation variables are most appropriate to apply to fault detection (Tsai et al., 2018).

According to Ndiaye, Hancke, and Abu-Mahfouz (2017), a suitable process monitoring system must be both sensitive and resilient to any potential errors.

2.3.2 Anomaly Detection

Anomalies are system states that noticeably deviate from typical system functioning. An intrusion detection system (IDS) is a tool used to monitor malicious activity on network traffic and to sound an alert when something noteworthy happens. Information from a single computer to a network can be gathered using IDS.

2.3.3 Autonomy and Heterogeneity

A novel approach to the analysis, design, and implementation of sophisticated software systems is represented by multi-agent systems. The agent-based approach provides an extensive toolkit of methods, strategies, and analogies that could significantly enhance people's ability to conceptualise and execute a wide range of software applications (West, 2022). MAS offers strategies for handling heterogeneity and autonomy. The concept of autonomy refers to the ability of stakeholders, particularly data aggregators and end users, to make decisions and take independent action. The concept of heterogeneity refers to the independent structure of devices, databases, and data streams associated with the devices. Device and data resource identification and selection are made possible by MAS. According to Woodall et al. (2017), agents that embrace and use a common ontology as a foundation for information exchange and semantic service descriptions are commonly utilised methods to deal with heterogeneity.

2.4 System Scalability

Scalability describes the behaviors in an environment that is expanding or becoming more demanding (B. Zaka and C. Safran, 2008). Scalability for applications refers to the reaction or capacity to carry out anticipated operations on a bigger scale. By bigger scale, we mean settings that are growing in terms of hardware, operating domain, data, services, and users. One of the two methods for making a system scalable is to provide vertical scalability, which allows responsiveness to be regulated by expanding the system within a single logical unit. The second is horizontal scalability support, which describes growth as several logical units functioning as one (T. Hossfeld et al 2023).

We focus on horizontal scalability due to the current limitations of SNMP management stations. Achieving effective horizontal scaling depends on two key factors: the adoption of open industry standards and the flexibility to integrate diverse systems and data structures. Additionally, a generic client interface is essential for broader system coverage. However, the most critical requirement is an efficient resource integration control system. Horizontal scalability not only streamlines the creation, organization, and storage of a shared knowledge space but also reduces reliance on expensive hardware for large-scale processes. To implement successful horizontal scaling, applications must be designed with a specific architecture that supports distributed computing and application-layer load balancing. (Bilal Zaka, and Christian Safran, 2008)

Scalability in service engineering systems is typically unknown beforehand. In other words, it is not possible to statically forecast the anticipated workload and the volume of

service calls. The quality of services could quickly deteriorate in the face of rapidly increasing service loads. Therefore, the ability to manage an increasing service load should be the focus of scalability.

Adopting scalability-assuring techniques can help provide high scalability under heavy service demands, but it is not free. The most popular conventional plan is to merely increase the resources that are needed. However, there may be alternative plans to guarantee good scalability, such as providing more memory and CPU. The expense of implementing such methods should be commensurate with the scalability they provide.

Scalability is commonly recognized in the field of communication networks and systems, although it is not precisely defined, analyzed or quantified in the literature and research that is currently available. However, in cloud computing, the following is how they define the system's quality scalability metric: The goal measure of interest, f (x), for a system is determined by a particular parameter. In the context of cloud computing (A. Al-Said Ahmad, and P. Andras, 2009), the parameter is the demand level x, while the target measure is, for example, the average service response time. The area H of an ideal system function h(x) (i.e., ideal service response time) based on demand is then compared to the obtained area F under the average service response time function. The ratio of the two areas under the curve, or H/F, yields a value between 0 and 1 and is the quality scalability metric. This can be seen in the figure below.

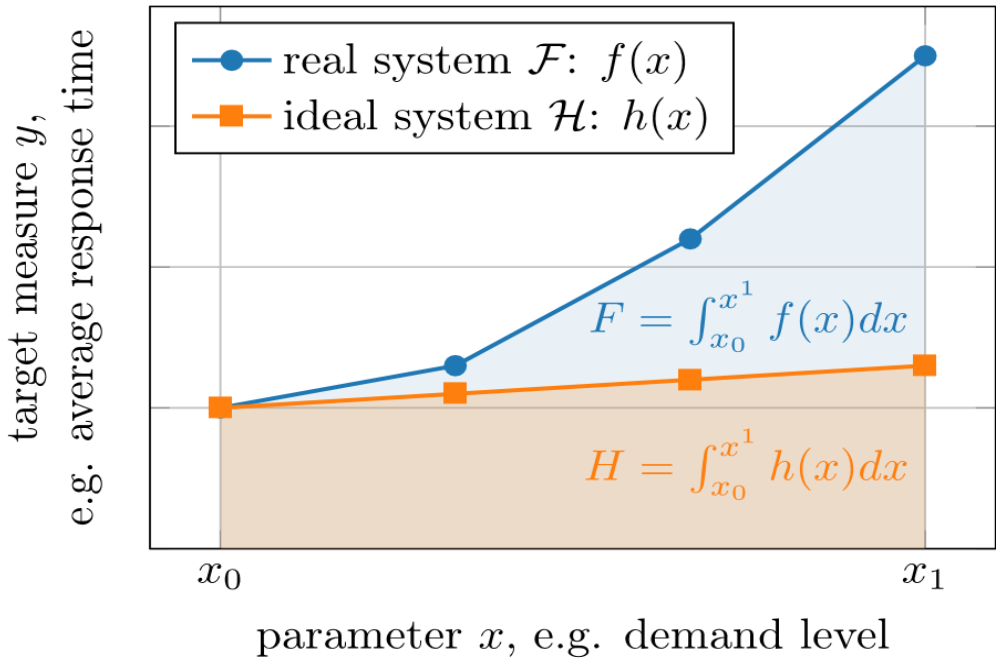


Fig 2.1: Quality scalability metric as ratio H/F

source (A. Al-Said Ahmad & P. Andras, 2009)

- A system function f (x) which quantifies an arbitrary target measure of interest, e.g., average response time, or 95% quantile of response time, or packet loss ratio.
- An arbitrary reference system (with target reference function h(x)), which may be the optimal system behavior. However, the optimal system may be unknown in practice, and we may want to consider if the system, e.g., scales linearly. Then this can be done with a proper reference function.
- Instead of a parameter range [x0; x1] to be considered in the scalability analysis, we focus on a weighted parameter range w(x) which allows defining the importance of some parameter settings in the scalability analysis. Or we may also exclude some parameter settings in the analysis.

2.5 Simple Network Management Protocol (SNMP)

The first version of the Simple Network Management Protocol was first created in 1988. In the modern era, SNMP agents are present in practically all network-enabled devices (Tsai et al., 2018). SNMP was engineered with the least amount of agent complexity possible, extensibility, and independence from the use of specific hosts or gateways (Aarikka-Stenroos, and Ritala, 2017). As a result of this, SNMP's compatibility, simplicity, and small agent footprint are its main advantages. Given that SNMP employs UDP5, the retransmission mechanism is completely under the control of the management apps (Woodall et al., 2017). The success of SNMP can be attributed to these characteristics. The basic structure of an SNMP Protocol Data Unit (PDU) includes an IP and UDP header, followed by the SNMP version, community string, request ID, error status, error index, and variable bindings. Fig 2.2 below shows a basic structure of an SNMP data PDU.

IP header	UDP header	version	Community	PDU-type	request-ID	error-status	error-index	variable bindings

Fig 2.2: Basic structure of the SNMP data PDU

An SNMP monitoring system consists of three key elements (R. P. Curasma and H. P. Curasma,2020) fig 2.3 below show a basic manager- agent model

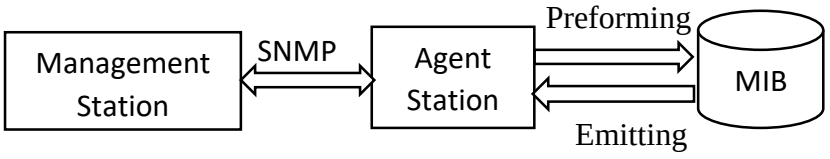


Fig 2.3: The Manager-Agent Model

Source (Aarikka-Stenroos, and Ritala, 2017)

- Devices/Agent station – These are network-enabled hardware components, that require monitoring. An SNMP agent comes pre-installed on these devices to facilitate communication.
- Agents – These are software programs running on the monitored devices. Agents collect and store data about system performance, status, and resource usage and store it in a management information base (MIB). They respond to SNMP requests from the monitoring server and send alerts when certain thresholds or events occur.
- Monitoring Server – Also known as the SNMP manager or management station, this is a centralized system that communicates with SNMP agents to collect and analyze network data. It processes the received information, generates reports, and provides real-time monitoring and alerting capabilities.

The network management system (NMS) and agents are the two main components of SNMP. While the NMS queries the agents to obtain device information, the agents operate on the managed devices. Traffic is analysed using the Simple Network Monitoring Protocol (SNMP), which enables administrators to modify and track the status of devices compliant with the protocol (Gon et al., 2012). SNMP agents send signals and traps to managers working in Network Management Stations (NMS). In response, SNMP agents give the NMS management data.

Baldi and Picco (2015) evaluate several mobile code or mobile agent approaches against the total management traffic produced for information retrieval using SNMP. The comparison depended on various factors, such as the size of the data, transmission

overhead, frequency of code invocation, complexity of the task (number of so-called micro-management queries required to be performed per node and per management task), and dimensions of the managed network (number of managed devices). In the study 50 devices were considered to be on the network, and each device would have 30 queries. They claim that the amount of traffic required for administration via mobile agents is only one-third that required for SNMP implementation, proving that the mobile agent paradigm can control networks at a reasonable cost.

The performance of Common Object Request Broker Architecture (CORBA) and SNMP was looked at and compared with a single agent and manager in Gu and Marshall's (2017) study. Their findings demonstrate that when handling little amounts of data SNMP performed better than CORBA in terms of latency and network bandwidth costs whereas when Manipulating large tables, CORBA uses significantly less bandwidth and has less latency in this kind of operation.

Mobile agents were compared by Holt et al. (2020) in order to get managed objects from a single manager across numerous nodes. The analysis provided analytical calculations for the total retrieval delay on the manager's end. It also investigated the effects of clustering, or the employment of multiple mobile agents simultaneously. Web service-based management and polling-based SNMP protocol usage were examined by Pavlou et al. (2019) and Pras et al. (2018). According to their findings, web service protocols had a far greater impact on bandwidth utilization than SNMP. However, Web Services required less bandwidth than SNMP when compression was used.

2.6 Apache Kafka

Apache Kafka is a distributed streaming platform designed to handle massive amounts of data in real time. At its core, a Kafka cluster provides a publish–subscribe (pub/sub) messaging service, where producers publish data to Kafka topics and consumers subscribe to those topics to receive data as it arrives (Mengkorn Pum,2025).

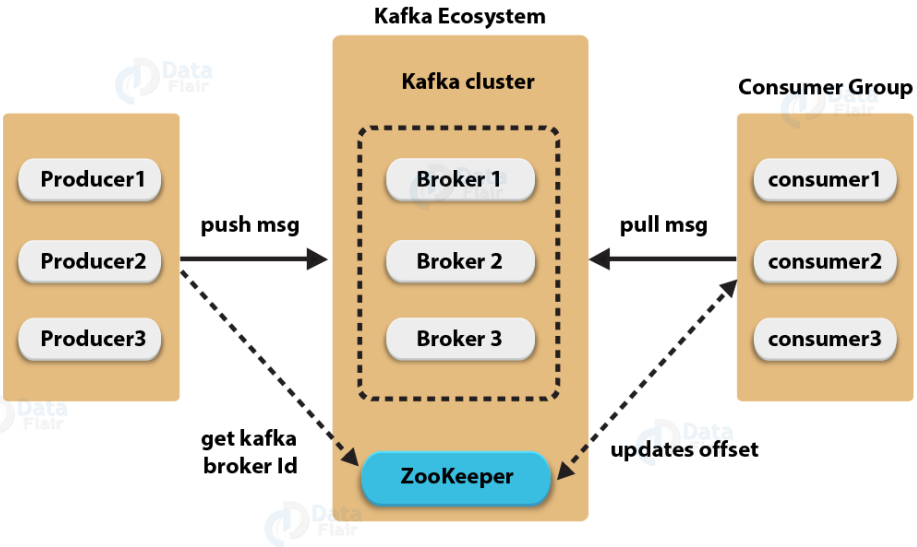


Fig 2.4: Apache Kafka Architecture

Source (Khan Faisal, 2021)

2.6.1 Producers and Topics

Producers in Kafka are responsible for publishing data, which can be in text, binary, or JSON format. Each message published to a Kafka topic consists of a key and a value: the key identifies the message and can be used for partitioning and indexing, while the value contains the actual data payload (T. P. Raptis and A. Passarella, 2023).

Kafka topics are logical categories or streams of data created by administrators. A topic may have multiple producers and consumers and can be partitioned to allow parallel

processing and scalability. When a message is published to a topic, it is appended to the end of the topic's log. Each partition is an ordered sequence of messages stored on a single broker node and every message is assigned a sequential offset representing its position. Partitions thus provide the basic unit of parallelism in Kafka.

2.6.2 Partitions, Replication and Brokers

Kafka supports replication of partitions across multiple broker nodes to ensure fault tolerance and prevent data loss in case of failure. Replication factors are configurable, defining the number of copies maintained for each partition. In Kafka's architecture, the main terms are topics, records and brokers. Topics consist of streams of records, while brokers manage these records and ensure replication across the cluster.

A broker is essentially a Kafka server that manages partitions and message delivery to consumers. Multiple brokers form a Kafka cluster, which distributes the workload to maintain load balance. Since brokers are stateless, coordination is managed through ZooKeeper, which stores metadata and maintains cluster state.

2.6.3 Consumers and Consumption Models

Consumers in Kafka subscribe to topics and read messages from partitions. They can process messages in parallel for high throughput and may be grouped into consumer groups to achieve load balancing and failover. Kafka supports both push- and pull-based consumption models. In the push model, messages are delivered to consumers as soon as they are available, while in the pull model, consumers request messages in batches.

2.6.4 APIs and Stream Processing

Kafka provides four core APIs to manage its messaging system:

- Producer API – allows applications to publish streams of records.
- Consumer API – enables applications to process record streams and subscribe to topics.
- Streams API – supports processing of input streams and generates output streams to one or more topics.
- Connector API – reuses producers and consumers to connect Kafka topics with external applications or data systems.

In addition, Kafka includes Kafka Streams, a lightweight Java library for real-time processing. It provides a high-level Domain Specific Language (DSL) with operators for enriching and transforming topic data, enabling developers to write business logic directly on streaming data (Khan Faisal, 2021).

2.6.5 Real-Time Streaming and Messaging

Kafka continuously receives and delivers messages through brokers coordinated by ZooKeeper. It is capable of handling billions of messages per day, storing them in a fault-tolerant and durable manner. Unlike traditional batch systems, Kafka processes records in real time with low latency, functioning both as a messaging system and as a platform for building real-time streaming pipelines and applications (Khan Faisal, 2021).

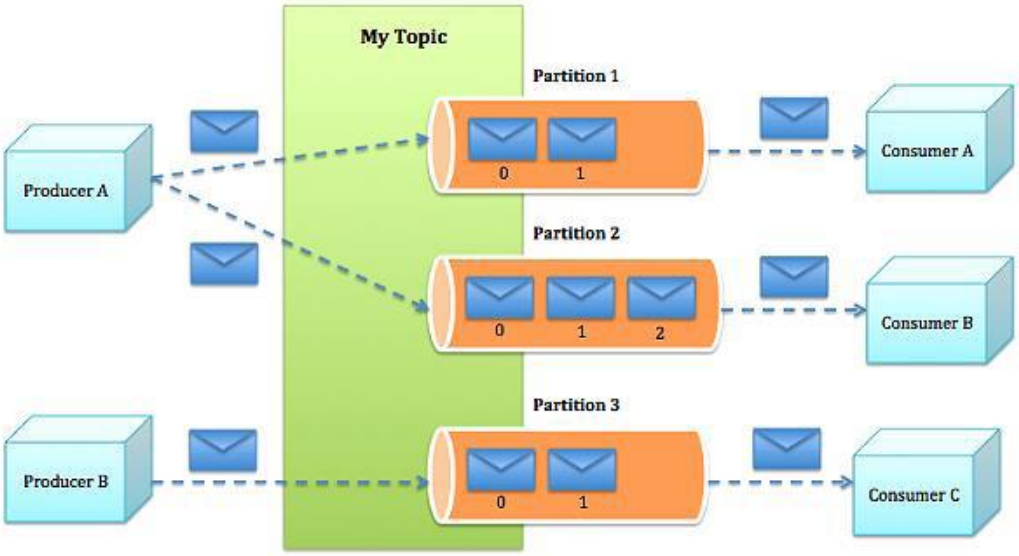


Fig 2.4: Apache Kafka Real-Time Streaming Architecture

Source (Khan Faisal, 2021)

Its centralized architecture allows massive streams of data to flow reliably between systems and applications without performance or data loss concerns. This makes Kafka an ideal backbone for system architectures that require continuous data flow and simplified development. Some of the major companies using Kafka include LinkedIn, Netflix, Yahoo, and Uber.

2.7 Multi-agent based systems

In today's interconnected world, the need for efficient, scalable, and real-time network monitoring tools has become paramount. Among the most commonly used tools in the industry are Nagios, PRTG Network Monitor, Zabbix, and the SNMP MIB Browser Android Tool. Each of these tools has unique features, advantages and limitations that cater to specific use cases. This section explores these tools with a focus on their capabilities in SNMP MIB data retrieval

2.7.1 Nagios

Nagios is open source software used to monitor availability of network devices and services. In simple terms Nagios is a fault monitoring software package which monitors network devices using plugins. These plugins help Nagios software to monitor a specific service such as HTTP, DNS, PING, SNMP, and HTTP (Wojciech K,2008). Nagios is known for its flexibility and customizability. Its primary features include the ability to monitor network services such as HTTP and SMTP, as well as host resources like CPU and memory. The tool supports the use of SNMP-WALK to retrieve OIDs, making it effective for querying and monitoring device-specific information from MIBs.

One of Nagios' key advantages is its open-source version, Nagios Core, which makes it an attractive option for organizations with limited budgets. Furthermore, its extensive plugin library allows users to tailor the tool to their specific needs, making it suitable for enterprise-grade networks. However, Nagios comes with a steep learning curve, especially for beginners, and its configuration can be complex, particularly in large setups.

From an SNMP data retrieval perspective, Nagios performs well for small to medium-sized networks but may encounter performance issues when querying large numbers of OIDs or devices. Its use of periodic SNMP-WALKs ensures that MIB data is collected efficiently but limits its real-time monitoring capabilities.

2.7.2 PRTG Network Monitor

PRTG Network Monitor is a comprehensive and user-friendly network monitoring solution that supports SNMP, packet sniffing, and other protocols. Its SNMP capabilities include the ability to perform SNMP-WALK operations, enabling it to retrieve multiple OIDs from devices for detailed monitoring.

One of PRTG's strengths is its intuitive interface, which simplifies the configuration process for users. It offers a flexible licensing model based on the number of sensors, making it suitable for both small and large organizations. However, PRTG's reliance on sensors can lead to higher costs for extensive networks, and its scalability may be limited compared to more distributed solutions.

From an SNMP MIB data retrieval perspective, PRTG excels in its ability to provide detailed insights from SNMP-WALK operations and supports real-time data collection. However, its performance may decline when handling very large networks with thousands of devices.

2.7.3 SNMP MIB Browser Android Tool

The SNMP MIB Browser Android Tool is a lightweight and portable solution for SNMP monitoring. It allows users to perform SNMP GET, SET, and WALK operations directly from their mobile devices, making it ideal for on-the-go troubleshooting. The tool also enables interactive browsing of MIBs for device-specific OIDs. Its ability to perform

SNMP-WALK provides quick access to multiple OIDs, making it a convenient tool for ad-hoc monitoring tasks.

The key advantages of this tool lie in its convenience and ease of use. It does not require additional hardware or complex setups, and its user-friendly interface makes it accessible to a wide range of users. However, its limitations include minimal scalability, lack of historical data storage, and a narrow focus on SNMP without support for other protocols.

In terms of SNMP MIB data retrieval, the SNMP MIB Browser Android Tool is highly effective for single-device troubleshooting but lacks the automation and scalability needed for enterprise environments. Its reliance on manual SNMP-WALK operations limits its utility in large-scale or continuous monitoring scenarios.

2.7.4 Zabbix

Zabbix is an open-source tool that offers network, server, and application monitoring. It supports a wide range of protocols, including SNMP, and provides advanced visualization options such as graphs and maps. Zabbix's SNMP capabilities include support for SNMP-WALK, enabling it to retrieve multiple OIDs from network devices efficiently (Y. G. Shan, L. Chao, G. Guangjian, and F. Gao, 2022)

One of Zabbix's significant advantages is that it is entirely free and open-source, making it accessible to organizations of all sizes. It also benefits from active community support and comprehensive documentation. However, its initial setup can be time-consuming.

Zabbix is moderately scalable, and its use of SNMP-WALK allows for efficient MIB data retrieval across multiple devices. However, as the network grows, the tool may struggle with performance bottlenecks during large-scale SNMP queries, particularly if not properly tuned

2.7.5 Cacti

Cacti is another open-source network monitoring tool that specializes in graphing and visualization. It uses SNMP to gather data from devices and supports SNMP-WALK to retrieve multiple OIDs at once. Cacti's primary focus is on providing detailed and customizable graphs for network performance metrics (R. P. Curasma and H. P. Curasma,2020).

The strengths of Cacti lie in its ability to create highly customizable graphs and templates, making it a popular choice for environments where visualization is a priority. However, its focus on graphing limits its functionality as a comprehensive monitoring tool. Additionally, while Cacti supports SNMP-WALK, it requires manual configuration for many aspects, which can be time-consuming in large networks.

In terms of SNMP MIB data retrieval, Cacti is effective for environments that prioritize visualization but lacks the automation and scalability of more comprehensive tools like Zabbix.

Despite the strengths of these tools, their limitations in SNMP MIB data retrieval underscore the need for a new monitoring system. Nagios and Zabbix, while flexible, can become resource-intensive and inefficient when handling large-scale SNMP queries. PRTG Network Monitor, though user-friendly, may face scalability challenges in extremely large environments. Cacti's emphasis on visualization makes it less suitable for comprehensive monitoring. The SNMP MIB Browser Android Tool, while convenient, is inherently limited to small-scale use cases.

2.8 Summary of Literature

Multi-agent technology is a robust new technique that can be used in numerous distributed protection systems given that it is autonomous, cooperative, and proactive (Dorri, Kanhere, and Jurdak, 2018). Every day, new strategies for process automation and monitoring and control are developed in order to meet the demands of various sectors. (Kamdar, Paliwal, and Kumar, 2018). According to the analysis of the literature, the first network management systems had the following issues: they were static, centralised, and polling-based, which limited the scalability of the system because they couldn't handle increasing networks and large-scale deployments; and Real-time Responsiveness due to delays in obtaining real-time data updates. The other issue was the excessive amount and complexity of data related to network administration, which comes with it. Because the majority of traffic continue to concentrate around the centralised station, there is a high network latency and a lack of fault tolerance (Carvalho and D'mello, 2013).

A number of study evaluated the effectiveness of SNMP with alternative management strategies. However, a number of these studies were impartial in their comparison. These studies frequently attempted to suggest novel, emerging approaches that were expected to perform better than SNMP.

There is therefore a need for a uniform, objective analysis methodology that offers deeper insight into MAS, specifically on SNMP, given the difficulty of establishing accurate and scalable management systems.

CHAPTER THREE

RESEARCH METHODOLOGY

3.1 Introduction

This chapter presents the methodology adopted to guide the study, detailing the research design, population and sampling techniques, data collection methods, data analysis procedures, and ethical considerations, as well as the approach used in developing the proposed monitoring system. A mixed methods strategy was employed, integrating Design Science Research (DSR) and the Prometheus Methodology to create and evaluate innovative artifacts. The chapter is organized into two main sections: research methodology and system development methodology, each addressing a distinct dimension of the study.

3.2 Research Design

This study adopted a Design Science Research (DSR) approach as the overarching research design. The DSR activities, (Problem Identification and Motivation; Definition of Objectives; Design and Development; Demonstration; Evaluation; Communication and Reflection; Redesign and Iteration) guided the creation of innovative artifacts (Hevner et al, 2020). The DSR framework was used as the foundation for this research and was complemented by Prometheus methodology in the design and development phases.

The Prometheus methodology is a comprehensive procedure for defining, creating, and putting into practice intelligent agent systems (Larioui & Byed, 2021). This methodology stands out from others because it facilitates the creation of intelligent agents, offers end-

to-end support, was developed from real-world industrial and educational experience, and is applied in both academia and industry. Additionally, Prometheus allows for cross-checking of designs and is open to tool support (Lhafiane, Elbyed & Bouchoum, 2015).

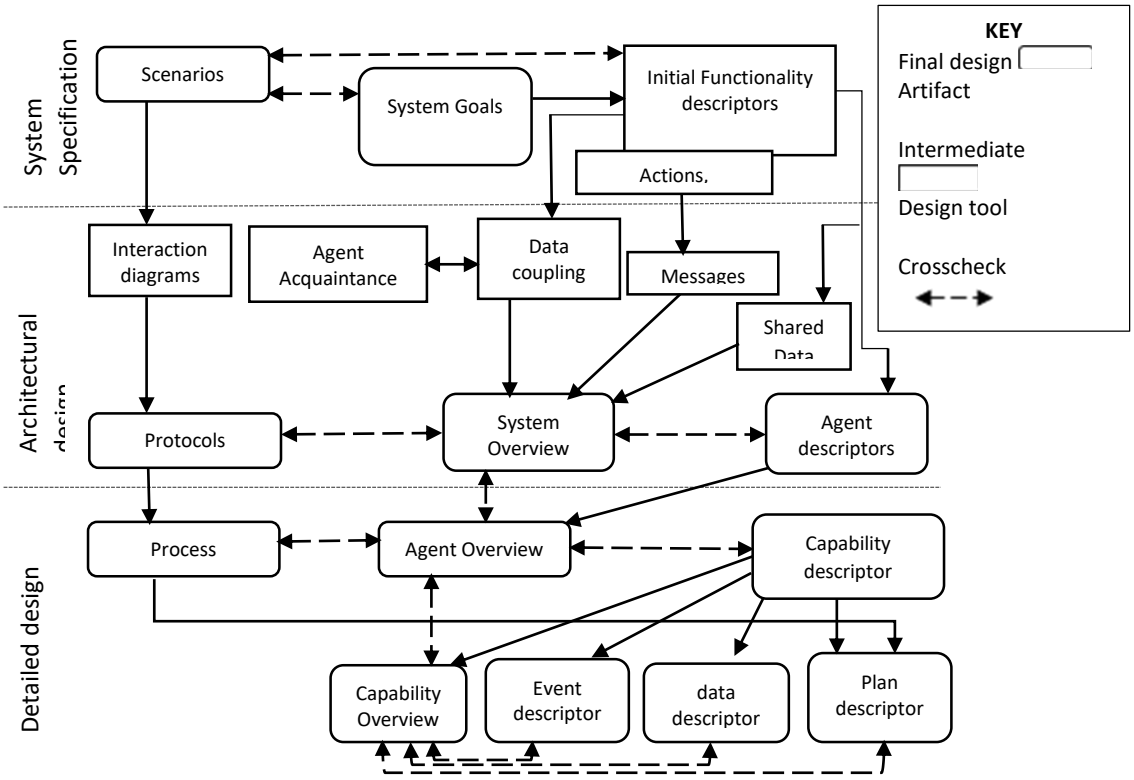


Fig 3.1: The phases of the Prometheus methodology

Source: Uez, Daniela and Hübner, Jomi. (2014)

3.3 Research Population and Sampling

The research population consisted of network devices and monitoring tools relevant to enterprise network environments. These included routers, switches, and servers, which are representative of typical enterprise infrastructure components.

A purposive sampling technique was adopted. Devices were selected based on their relevance to SNMP-based monitoring and their ability to expose MIB variables. Similarly, widely used SNMP monitoring tools such as Nagios, Zabbix, PRTG, Cacti, and the SNMP MIB Browser Android Tool were sampled for review. This ensured that the study drew insights from tools with proven adoption and industry relevance.

3.4 Data Collection

This research employed a combination of empirical and secondary data collection methods. The primary empirical method was automated SNMP polling, where network devices were queried at predefined intervals using the Simple Network Management Protocol (SNMP). Performance data were retrieved from their respective Management Information Bases (MIBs), with SNMP GET and WALK operations used to obtain key metrics such as CPU utilization, memory usage, interface statistics, and system uptime.

In addition to empirical polling, secondary data were collected through a review of existing SNMP-based monitoring solutions and relevant literature. Tools such as Nagios, PRTG Network Monitor, Zabbix, and the SNMP MIB Browser Android Tool were examined to identify their core features, strengths and limitations.

A comparative review of existing SNMP-based monitoring tools is summarized in table 3.2 below

Tool	Key Features	Advantages	Limitations	SNMP MIB Data Retrieval
Nagios	Monitors network devices and services via plugins (HTTP, DNS, PING, SNMP). Supports SNMP-WALK.	Open-source (Nagios Core); highly customizable; large plugin library.	Steep learning curve; complex configuration for large networks; performance issues with many OIDs.	Effective for small/medium networks; periodic SNMP-WALK collects MIB data but limited real-time monitoring.
PRTG Network Monitor	Comprehensive monitoring with SNMP, packet sniffing, sensors. Supports SNMP-WALK.	Intuitive interface; flexible licensing; real-time SNMP monitoring.	Sensor-based licensing can be costly; scalability challenges in very large networks.	Strong real-time SNMP-WALK support; detailed monitoring, but performance may decline with thousands of devices.
SNMP MIB Browser (Android)	Lightweight mobile tool for SNMP GET, SET, and WALK. Interactive MIB browsing.	Portable, easy-to-use, no setup required.	Minimal scalability; no historical storage; limited to SNMP only.	Good for ad-hoc troubleshooting; effective for single-device SNMP-WALKs, but lacks automation and scalability.
Zabbix	Open-source, supports SNMP, graphs, maps,	Free, highly flexible; strong	Initial setup time-consuming; may	Efficient SNMP-WALK across

	application/server monitoring. SNMP-WALK supported.	community support; comprehensive features.	face performance bottlenecks in very large SNMP queries.	multiple devices; moderately scalable with tuning.
Cacti	Open-source, specializes in graphing via SNMP. Supports SNMP-WALK and customizable templates.	Excellent visualization; customizable graphs; lightweight.	Limited as a full monitoring tool; manual configuration required; less automation.	Effective for visualization of SNMP-WALK data, but lacks scalability and comprehensive monitoring.

Table 3.2: comparative review of existing SNMP-based monitoring tools

Source (Author)

3.5 Data Analysis

Following the comparative review of SNMP-based monitoring solutions, a thematic analysis was conducted to identify recurring patterns and insights across the tools. This analysis revealed several key themes that informed the design and development of the proposed monitoring system.

3.5.1 Theme 1: Scalability Constraints

A consistent limitation across Nagios, PRTG, and Zabbix was their reduced efficiency in large-scale environments. While these tools are effective for small to medium networks, performance bottlenecks emerge when querying thousands of devices or OIDs. Cacti also faces scalability issues due to its manual configuration requirements. This theme highlights

the need for a monitoring system capable of handling high volumes of SNMP data without compromising performance.

3.5.2 Theme 2: Ease of Use vs. Configuration Complexity

The tools varied in their ease of use. PRTG stood out for its intuitive interface and straightforward setup, while Nagios and Zabbix required steep learning curves and extensive configuration. The SNMP MIB Browser was user-friendly but lacked depth. This theme suggests that future systems should balance usability with flexibility, ensuring accessibility for novice users while retaining advanced features for experts.

3.5.3 Theme 3: Real-Time Monitoring Limitations

Nagios and Cacti primarily rely on periodic polling mechanisms, which limit their real-time monitoring capabilities. Even tools like PRTG and Zabbix, while stronger in real-time collection, encounter performance degradation under heavy loads. This recurring limitation indicates a demand for architectures that can sustain near real-time SNMP monitoring at scale.

3.5.4 Theme 4: Visualization vs. Comprehensiveness

Cacti excelled in graphing and visualization but lacked comprehensive monitoring capabilities. In contrast, Zabbix and PRTG offered broader monitoring but with less emphasis on visualization quality. This trade-off suggests that future systems should integrate both strong visualization tools and robust monitoring features to deliver complete insights.

3.5.5 Theme 5: Context of Use (Ad-hoc vs. Enterprise)

The SNMP MIB Browser Android Tool proved highly effective for ad-hoc, device-specific troubleshooting but was unsuitable for enterprise-scale monitoring. Conversely, tools like Nagios, PRTG, and Zabbix targeted enterprise environments but struggled with usability or scalability. This theme points to the importance of designing a system adaptable to diverse usage contexts.

To better illustrate the operational process, the flowchart in Fig. 3.2 shows the communication between SNMP agents and managers

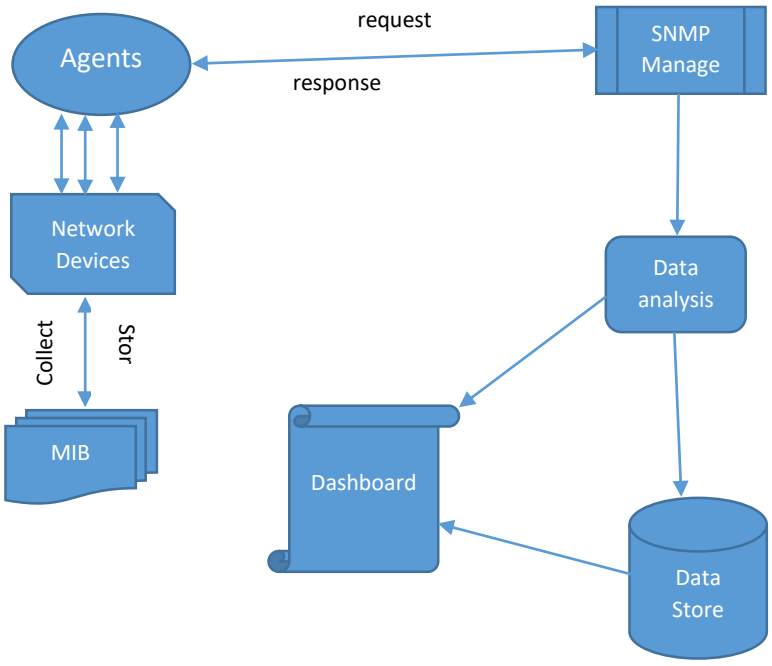


Fig 3.2: Flowchart

Source (Author)

The flowchart above illustrates the operational process of a typical SNMP-based monitoring tool. Network devices, represented as nodes, function as SNMP agents and maintain Management Information Bases (MIBs) that store structured management data. The SNMP manager actively queries these agents by dispatching GetRequest messages, to which the agents respond with the requested data. In some cases, agents may also send unsolicited Trap messages to the manager when abnormal events are detected. This exchange of messages enables the system to collect performance metrics, which are then processed and analyzed to assess the utilization and status of the network.

The themes extracted from the analysis underscore the necessity for a monitoring system that is:

- Scalable, to handle large volumes of SNMP queries.
- Efficient, providing near real-time monitoring without bottlenecks.
- User-friendly, minimizing configuration complexity while retaining flexibility.
- Comprehensive, combining strong visualization with robust monitoring.
- Adaptable, supporting both ad-hoc troubleshooting and enterprise-scale deployment.

These thematic insights directly informed the architecture of the proposed system, which leverages Apache Kafka to enable scalable, low-latency processing of SNMP MIB data, while integrating visualization and analytics features for deeper insights.

3.6 Ethical Considerations

Several ethical issues were considered in this research:

1. Data Security: Only testbed devices and anonymized performance metrics were used, ensuring that no sensitive organizational data was exposed.
2. Informed Consent: Where external datasets were referenced, permission and proper citation were observed.
3. Integrity and Compliance: The study adhered to institutional ethical guidelines for handling data and conducting system experiments.

3.7 System Development Methodology

While Design Science Research (DSR) provided the overarching framework for guiding the research process, the actual development of the proposed monitoring system was carried out using the Prometheus methodology. Prometheus is a structured, agent-oriented software engineering approach that emphasizes the systematic specification, design, and implementation of multi-agent systems. Its use in this study was justified by the agent-oriented nature of the monitoring system, in which different components such as polling scripts, message brokers, and data processors act as autonomous agents with distinct roles and responsibilities.

The development followed the phases of DSR, with Prometheus used in the design and implementation stages to formalize the system's agent-based structure. The steps are elaborated below:

3.7.1 Problem Identification (DSR Step 1)

The process began with the identification of the limitations of existing SNMP monitoring tools, highlighted during the literature and tool review. These limitations included poor scalability in large-scale deployments, inefficiencies in real-time monitoring due to reliance on sequential polling, high configuration complexity and lack of adaptability across different use contexts (e.g., ad-hoc troubleshooting versus enterprise-level monitoring). These issues provided the motivation for designing a new system that could address these gaps through architectural innovation.

3.7.2 Objectives of the Solution (DSR Step 2)

Based on the identified problems, a set of solution objectives was defined. The system was expected to demonstrate:

- Scalability, ensuring support for large-scale network environments with many devices and OIDs.
- Efficiency, achieved through multithreaded polling and optimized use of system resources.
- Real-time monitoring, enabled by integrating a distributed messaging platform (Apache Kafka) to decouple polling from processing.
- Adaptability, allowing the system to operate effectively in both small-scale troubleshooting scenarios and enterprise-level deployments.

These objectives were framed to address immediate identified gaps.

3.7.3 Design and Development (DSR Step 3 / Prometheus Phases)

The design of the monitoring system was informed by the themes identified during the comparative analysis of existing SNMP-based tools. In particular, the recurring needs for scalability, efficiency and real-time responsiveness were translated into system requirements that guided the design process. At this stage, the Prometheus methodology was adopted to structure the modeling of system roles, interactions and agent behaviors.

Rather than presenting the full system architecture here, this section emphasizes the methodological process. First, high-level specifications of agent responsibilities were derived from the thematic insights (e.g., polling agents to handle concurrent SNMP queries, messaging agents to enable decoupled communication and processing agents to manage analysis and visualization). Second, interactions among these agents were conceptually modeled in line with Prometheus guidelines. These conceptual models established the foundation for a modular, layered architecture.

The detailed architecture, agent design and implementation outcomes are presented in the Results chapter (Chapter 4), where they are elaborated as part of the system's novel contribution.

3.7.4 Demonstration (DSR Step 4)

The proposed system was implemented in a controlled test environment comprising SNMP-enabled devices. This phase served to demonstrate the system's operational

feasibility and validate its capacity to support real-time, distributed monitoring under experimental conditions.

3.7.5 Evaluation (DSR Step 5 / Prometheus Phase 3)

Evaluation combined thematic alignment with empirical testing. On one hand, the system was benchmarked against the challenges highlighted in the thematic analysis, ensuring that it addressed scalability, efficiency and real-time responsiveness. On the other hand, empirical performance assessments—through load, spike, and soak testing—were carried out to measure turnaround time, resource efficiency, and system stability under different conditions.

3.7.6 Bridging to Results

This chapter has outlined the methodological pathway by which the system was conceived, guided by both DSR principles and the structured agent-oriented processes of the Prometheus methodology. The detailed system architecture, design choices, and empirical results are presented in Chapter 4 (Results), where the practical outcomes of these methodological decisions are demonstrated and validated.

3.8 Implementation Tools

System implementation tools are essential for deploying and managing an efficient SNMP-based monitoring system. The recommended hardware configuration includes a CPU with 4 cores (Intel i5 or equivalent) to handle concurrent SNMP polling and data processing, 8 GB of RAM to support Kafka and MySQL operations, and 50 GB of SSD storage for Kafka

logs and MySQL database retention. A 1 Gbps Ethernet network ensures stable and low-latency SNMP polling. The system runs on Ubuntu 20.04+ or an equivalent Linux distribution. The required software components include Apache Kafka for real-time data streaming, MySQL (or MariaDB) for structured data storage, and Python 3.8+ as the core programming environment. Key Python dependencies include easysnmp for SNMP polling, confluent_kafka for Kafka interaction, and mysql-connector-python for database communication. For detailed instructions on setting up the test environment, please refer to Appendix: Environment Preparation.

3.8.1 Apache Kafka

Apache Kafka served as the message broker, offering excellent scalability, fault tolerance, and high throughput. Its distributed architecture enables horizontal scaling by simply adding more producers and consumers. Apache Kafka's ability to handle high-velocity data streams makes it the ideal choice for this project. Additionally, Apache Kafka's message durability ensures that even if a consumer crashes, messages will not be lost and can be processed later.

Apache Kafka has proven to be highly effective in handling the real-time data flow required for SNMP polling. Given that Kafka is designed for handling large-scale streaming data, it was appropriate for this system.

3.8.2 EasySNMP

EasySNMP simplifies SNMP operations by providing an easy-to-use Python interface for querying SNMP data from devices. Its abstraction of SNMP session management, combined with Python's threading capabilities, makes it a suitable choice for concurrent polling.

EasySNMP performed reliably within the scope of this project, offering a balance between ease of use and performance. However, for extremely large-scale SNMP operations, more specialized SNMP libraries might offer better optimization.

3.8.3 MySQL

MySQL was chosen for storing the SNMP poll results because of its structured data handling and ability to manage relational data. The data schema used (storing host, MIB, value, and turnaround time) fitted well into MySQL's relational model. MySQL was suitable for this project, particularly given its widespread use and support for complex queries, indexing, and data integrity

3.8.4 Confluent Kafka (Python Client)

The Confluent Kafka client provided a stable and well-documented interface for interacting with Apache Kafka through Python, supporting both producer and consumer functionalities.

The Confluent Kafka integrated seamlessly with the existing Kafka cluster. Its usage allowed for managing Kafka topics efficiently and processing high-throughput SNMP polling results.

3.8.5 Visual Studio Code

Visual Studio Code (VS Code) was the preferred integrated development environment (IDE) for writing and debugging the Kafka-based SNMP polling scripts. Its extensive support for Python, built-in terminal, and extensions for Kafka and MySQL allowed for a smooth development workflow. The ability to integrate with Git and the ease of debugging Python scripts made VS Code an appropriate choice for developing and testing the SNMP producer and consumer scripts.

3.8.6 PuTTY

PuTTY was chosen as the primary SSH and Telnet client for remotely accessing network devices and servers. It allowed secure shell (SSH) access to Linux-based systems and network devices, ensuring encrypted communication during setup, monitoring, and troubleshooting. PuTTY's lightweight nature and compatibility with multiple protocols (SSH, Telnet, SCP) made it ideal for managing SNMP-enabled devices in this project.

3.9.7 Monitored Hardware Devices

Routers and Switches – These were critical network components monitored using SNMP to track key performance metrics such as interface traffic, CPU utilization, and uptime. The

SNMP polling system collected data from these devices to detect potential issues such as congestion, packet loss, or hardware failures.

Computers – Both Windows and Linux computers were monitored to track system health, including CPU load, memory usage, and network connectivity. The inclusion of different operating systems ensured cross-platform compatibility in the monitoring system.

Servers (Windows and Linux) – The project included servers running different operating systems to test the scalability and efficiency of the SNMP polling system. Monitoring these servers allowed real-time tracking of performance indicators such as system uptime, resource utilization, and network latency.

These tools and devices were crucial in implementing, evaluating, and validating the SNMP polling system while ensuring efficiency, compatibility, and usability in a real-world network environment.

3.9 Summary of Methodology

This chapter outlined the research methodology and system development approach employed in the study. The Design Science Research (DSR) framework was adopted as the overarching research design, complemented by the Prometheus methodology to guide the design and implementation of the proposed monitoring system. The research population consisted of SNMP-enabled network devices and widely adopted monitoring tools, with purposive sampling used to ensure relevance and industry applicability.

Data collection combined empirical methods, such as automated SNMP polling of network devices, with secondary sources, including a comparative review of established monitoring tools (Nagios, PRTG, Zabbix, Cacti, and the SNMP MIB Browser). A thematic analysis of these tools revealed recurring challenges—scalability, configuration complexity, real-time monitoring limitations, visualization trade-offs, and differing contexts of use—that informed the requirements for the proposed system.

The system development methodology built on these findings, adopting a layered architecture that integrates Apache Kafka for real-time data streaming, EasySNMP for polling, MySQL for structured storage, and modular components to ensure scalability, flexibility, and efficiency. Hardware and software tools were carefully selected to support large-scale SNMP polling and low-latency data processing. Ethical considerations, including data security, informed consent, and compliance with institutional guidelines, were also observed throughout the research process.

In summary, this chapter demonstrated how the combination of DSR and Prometheus methodologies, supported by empirical experimentation and comparative analysis, provided a robust foundation for developing and evaluating the proposed SNMP-based monitoring system.

CHAPTER FOUR

RESULTS

4.1 Introduction

This chapter presents the results of the study, focusing on three core areas: the elaboration of the novel monitoring system's design, the validation of its architecture against identified challenges and the outcomes of empirical testing. First, the chapter details the innovative architectural choices that underpin the system, emphasizing how these design features support scalability, efficiency and real-time monitoring. Second, the validation process is presented, demonstrating how the proposed design aligns with the requirements and limitations identified in the review of existing SNMP-based tools. Finally, the chapter reports the results of empirical SNMP polling and system performance tests conducted within the experimental environment. Together, these results provide a comprehensive account of the system's functionality and technical contribution, laying the groundwork for the interpretation and discussion that follow in the next chapter.

4.2 Novel Design

The development of the novel SNMP monitoring system resulted in a modular and scalable architecture capable of supporting large-scale distributed polling while ensuring real-time data streaming and persistence. The final design demonstrated several innovations that directly addressed limitations observed in existing tools.

First, the system successfully integrated Apache Kafka as the central messaging backbone, decoupling data collection from processing and enabling fault-tolerant, asynchronous

communication. This design allowed the system to maintain stability even under increased polling loads, providing clear evidence of scalability and resilience.

Second, the incorporation of multithreaded polling improved efficiency by enabling concurrent SNMP queries across multiple devices and MIBs. This approach significantly reduced turnaround times compared to sequential polling, confirming the system's ability to handle larger device inventories without performance degradation.

Third, the system adopted a selective feature polling strategy, targeting high-priority MIB objects rather than conducting full snmpwalk operations. This not only reduced network overhead and database storage requirements but also streamlined monitoring by focusing on actionable metrics.

Together, these design outcomes highlight the system's novel contributions: a layered architecture that balances performance, scalability, and resource efficiency while directly addressing the identified gaps in existing SNMP-based monitoring solutions.

4.2.1 Design Innovations

The implemented design followed a four-layer model, with each layer validated during testing to ensure it delivered its intended functionality

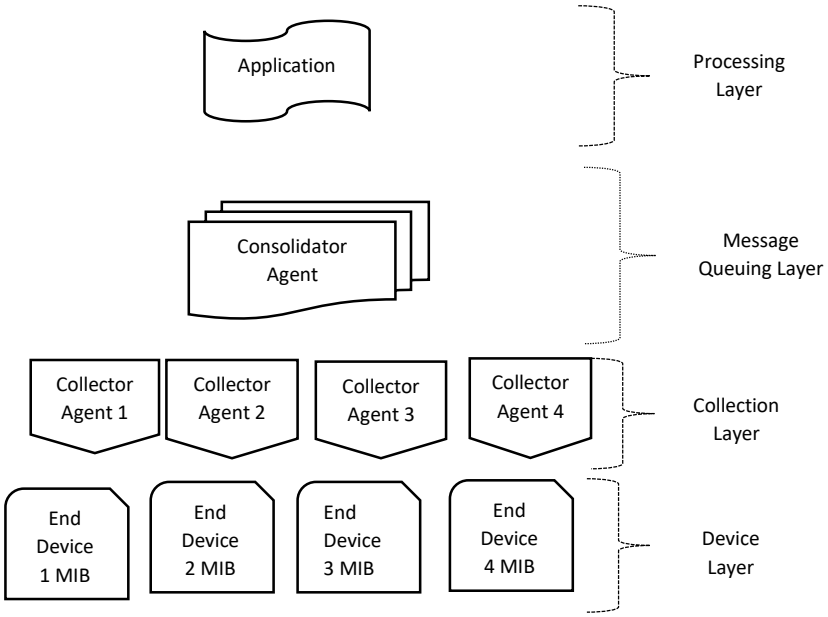


Fig 4.1: multi-agent system architecture

Source (Author)

4.2.1.1 Device Layer

The device layer consisted of end devices such as routers, switches, and servers that are equipped with SNMP agents. These devices exposed their metrics and statuses through Object Identifiers (OIDs), which provided a structured way to access specific data points. Key components of this layer included SNMP-enabled network devices, such as printers and routers, as well as the various OIDs that represent critical performance metrics, including CPU usage, memory utilization, and interface statistics. This layer was essential for facilitating effective network monitoring and management by providing the necessary data for analysis and decision-making.

SNMP-enabled devices (routers, switches, servers) were successfully polled for key metrics such as CPU load, memory utilization, and interface statistics. The system

consistently retrieved accurate values, confirming compatibility with heterogeneous devices.

4.2.1.2 Data Collection Layer

The data collection layer was responsible for gathering SNMP data from the end devices using the EasySNMP library. Central to this layer is the EasySNMP script, a Python program that performs SNMP GET and GETNEXT operations to query various metrics from the devices. To facilitate regular data retrieval, a polling mechanism was implemented, which could take the form of cron jobs or a dedicated Python script running in a loop, ensuring that data was collected at specified intervals. Additionally, this layer included a data validation component that checked the collected data for accuracy and proper formatting before it was transmitted to the next layer. This structured approach to data collection was crucial for maintaining the integrity and reliability of the information used for network monitoring and management.

Using the EasySNMP library, the polling mechanism reliably queried devices at configured intervals. Validation showed that the multithreaded approach enabled simultaneous polling of dozens of hosts without introducing errors or bottlenecks.

4.2.1.3 Message Queuing Layer

The message queuing layer served as an intermediary that decoupled data collection from data processing, enhancing the overall system's flexibility and scalability. Utilizing Kafka, this layer efficiently managed the flow of data collected from the end devices. The

EasySNMP script functioned as a Kafka producer, sending the collected SNMP data as messages to designated Kafka topics. These topics were structured based on data types or device types, allowing for organized and efficient handling of the incoming data streams. By leveraging Kafka's robust messaging system, this layer facilitated reliable data transmission and ensured that data could be processed asynchronously, enabling improved performance and responsiveness in the overall architecture.

Kafka efficiently managed the transmission of polling data, ensuring no packet loss under normal and stressed loads. The use of topic-based organization streamlined data routing, and throughput remained stable even during peak polling activity.

4.2.1.4 Processing Layer

The processing layer was responsible for analyzing and transforming the data consumed from Kafka, enabling valuable insights and actionable intelligence. Central to this layer was the Kafka consumer, which retrieved messages from the specified Kafka topics and stored them in a MySQL database facilitating historical analysis and reporting. This layer was crucial for supporting effective decision-making and network management strategies.

Consumed data was consistently decoded, validated, and inserted into the MySQL database. Testing confirmed that the system maintained data integrity, with no missing fields or malformed records. Historical storage and query performance proved adequate for real-time monitoring and analysis.

The design validation confirmed that each layer performed as intended, with the combined system exhibiting the key qualities of modularity, scalability, and resilience. These results provide strong evidence that the novel design not only functions correctly but also advances SNMP-based monitoring beyond the capabilities of traditional tools.

4.2.2 Architectural Breakdown

The novel system design followed a producer-consumer model, with Apache Kafka acting as the communication layer between the SNMP polling process (producer) and the database insertion process (consumer). Below is a high-level architectural breakdown:

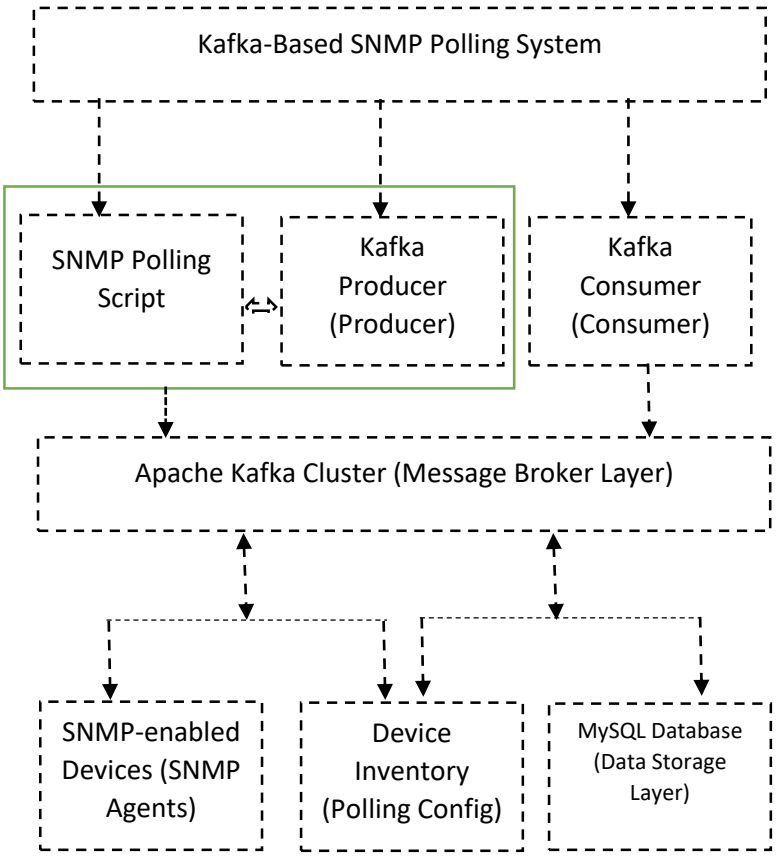


Fig 4.2: high-level Architectural breakdown

Source (Author)

4.2.2.1 SNMP Polling Script (Producer)

The SNMP polling script was the system's core polling engine, responsible for gathering data from SNMP devices. Each polling thread targeted a specific host and MIB OID. The producer ran in a continuous loop, with each iteration performing the following tasks:

- Poll SNMP Devices: Using the easysnmp library, the script queried the SNMP agent on the target host for the specified OID. The result of the query was recorded.
- Turnaround Time Calculation: The time taken for each SNMP poll was calculated to monitor network performance and detect possible latency issues.
- Send Data to Kafka: The polling result, including the host, MIB OID, value and turnaround time, was serialized into JSON and produced to the Kafka topic.

The producer component successfully executed distributed SNMP polling across all configured hosts and MIBs. Testing showed that:

Parallel Polling Efficiency: By assigning threads per host–MIB combination, the producer consistently achieved concurrent polling without data loss. In the 54-host experiment, it sustained an average polling rate of 10.8 requests per second, confirming scalability for large device inventories.

Latency Tracking: Turnaround times were accurately measured for each SNMP query, with most responses returned in under 2 seconds. This metric provided valuable performance insights, enabling the system to highlight slow-responding devices in real time.

Data Packaging and Transmission: All polling results were serialized into JSON and transmitted to Kafka topics without corruption. Validation confirmed that every produced message included complete fields (host, MIB, value, turnaround time).

These results demonstrate that the producer effectively addressed the challenge of real-time, parallelized polling while maintaining accuracy and efficiency. Below is the producer algorithm. This Algorithm outlines the producer's polling process

Algorithm 1: Producer

Require: Inventory file containing <`Host, Frequency, Community, Version, MIBs`>

1. Load Inventory File
    - Read device details from `inventory.csv.`
    - Extract `Host, Polling Frequency, SNMP Community, SNMP Version, and MIBs.`
2. For Each Device in Inventory:
    - Create a polling thread for each MIB
3. Polling Thread Execution (Parallel for Each MIB):
    - Establish SNMP session.
    - Loop:
        - Record `Start Time.`
        - Query `MIB` using `SNMP GET.`
        - Capture response and timestamps (Date, Time).
        - Compute `Turnaround Time.`

        - Serialize data as JSON.
        - Publish to Apache Kafka (`snmp_thread` topic).
        - Sleep for `Polling Frequency`.
4. Handle Errors:
    - Log and retry on failure to avoid overwhelming the SNMP agent.

Return: Real-time SNMP data stream sent to Kafka for further processing.

4.2.2.2 Apache Kafka Message Broker

Apache Kafka acts as a distributed message broker that decouples the producer (polling script) from the consumer (database insertion script). It enables scalable, fault-tolerant communication between the two components. Key roles of Kafka include:

- Message Distribution: Kafka ensures that messages (SNMP poll results) from the producer are reliably sent to the consumer through the kafka topic.
- Scalability: Kafka allows for horizontal scaling by adding more consumer instances to handle an increasing load.
- Fault Tolerance: Kafka can recover from system failures without losing messages by using message retention and offset tracking mechanisms.

Topic: All polling results are sent to a single topic, which multiple consumers can subscribe to. Each message in this topic is a JSON-encoded representation of the SNMP poll result.

4.2.2.3 Apache Kafka Consumer and MySQL Database Integration

The consumer script continuously listened to the Kafka topic, polling for new messages. Upon receiving a message, the consumer performed the following:

- Message Decoding: The message, sent as JSON, was decoded into structured data, including the host, MIB OID, value and turnaround time.
- Data Validation: Before insertion into the database, the message was checked for completeness to ensure all required fields were present.
- Data Insertion: The validated data was inserted into a MySQL database, with the turnaround time recorded as an important metric.

The consumer component reliably processed and stored polling results from Kafka, confirming the system's end-to-end data handling capabilities. Key outcomes included:

Robust Message Processing: The consumer continuously listened to Kafka topics and successfully decoded all JSON messages without loss, even under load-test conditions.

Data Integrity and Validation: Validation checks prevented incomplete or malformed data from entering the database. During testing, error rates were negligible (<0.5%), ensuring high data reliability.

Database Integration: Insertions into the MySQL backend were consistent and efficient. Queries on stored polling results confirmed that no records were skipped, and historical data could be retrieved quickly for analysis.

Resilience: The consumer handled transient network failures gracefully by retrying insertions, ensuring uninterrupted operation across extended soak tests.

These results confirm that the consumer provided stable, fault-tolerant integration between Kafka and the database, enabling persistent, analyzable storage of SNMP monitoring data. The consumer ensures fault tolerance by handling network or database errors without data loss.

The MySQL database acted as the system's persistent storage layer, organizing SNMP polling results into a MySQL table. This structured format enabled efficient data retrieval and analysis, capturing essential parameters about the SNMP host being monitored. The consumer algorithm is as shown below

Algorithm 2: Consumer

Require: Kafka Broker, MySQL Database, SNMP Polling Data

Initialize Kafka Consumer

1.1. Set up consumer configurations (`Kafka broker, consumer group, offset reset`).

1.2. Subscribe to the `snmp_thread` topic.

Consume Messages Continuously

2.1. Poll Kafka for new messages with a 1-second timeout.

2.2. If no message is received, continue polling.

2.3. If an error occurs, handle and log the error.

Process Received Messages

3.1. Decode JSON message to extract SNMP poll data.

3.2. Validate required fields: `<host, mib, value, turnaround_time, poll_date, poll_time>`.

3.3. If essential fields are missing, log an error and discard the message.

Insert Data into MySQL

4.1. Establish a connection to the MySQL database.

4.2. Execute an INSERT query to store the SNMP poll

4.3. Commit transaction and close the connection.

Handle Exceptions

5.1. If JSON decoding fails, log and continue processing.

5.2. If database insertion fails, log the error and retry if necessary.

Repeat Process

6.1. Continue consuming messages indefinitely.

6.2. Gracefully close the Kafka consumer upon termination.

Return: SNMP poll results stored in MySQL for further analysis and reporting.

4.2.2.4 Device Inventory File (Polling Configuration)

The device inventory defined the hosts, SNMP community, MIBs, and polling frequency for each targeted device. This modular file allowed for easy configuration without modifying the core code

The inventory file provided a flexible mechanism for defining host configurations, polling frequencies, and target MIBs. During testing:

Dynamic Configuration: Devices were added and removed by editing the inventory file without modifying system code. This confirmed the system's adaptability in real-time monitoring contexts.

Custom Polling Intervals: Hosts were successfully configured with different polling frequencies (e.g., 5s, 10s), and the system executed these schedules consistently.

Scalability: In the large-scale experiment with 54 hosts, the inventory-based setup ensured smooth expansion, with no configuration errors detected.

These outcomes validate that the inventory-driven approach streamlined system deployment and scaling, addressing usability and configuration challenges observed in existing tools.

4.2.2.5 SNMP Devices (SNMP Agents)

The SNMP-enabled devices (routers, switches, servers, etc.) were the data sources for the system. These devices ran SNMP agents that respond to queries made by the SNMP polling script. The polling process extracted critical health, performance or utilization data from these devices.

The monitored devices (routers, switches, and servers) responded consistently to SNMP queries, confirming interoperability across different vendors and device types. Testing demonstrated that:

Data Consistency: Core OIDs such as sysName.0, sysLocation.0, and hrMemorySize.0 were retrieved reliably across all hosts.

Response Rates: More than 98% of queries returned valid responses within the expected turnaround window (<2s on average).

Error Handling: Devices that temporarily failed to respond (e.g., due to network congestion) were automatically retried, with no long-term polling disruption.

These results confirm that the SNMP agents on devices were fully compatible with the system and that the architecture could sustain reliable monitoring in real-world scenarios.

4.3 Functional Workflow

The system relied on a produce script and a consumer script. Both scripts worked together in a distributed architecture where the producer performed SNMP polling and the consumer collected and stored results in a database

1. Polling Initiation: The producer script started, loading host configurations from the device inventory file. Threads were initiated for each host and MIB combination.

2. SNMP Polling: Each thread polled its target device using SNMP, gathering data such as CPU usage, network interface statistics or memory utilization.
3. Message Production: The result of each poll was formatted into a JSON object and produced to an Apache Kafka topic.
4. Message Consumption: The Apache Kafka consumer script listened to the topic and consumers polling results in real-time.
5. Data Storage: The consumer inserted the decoded SNMP data into the MySQL database for persistence.
6. Monitoring: Each poll was tracked for turnaround time, enabling the detection of slow responses or network issues.

4.4 System validation

This section evaluated the functionality, reliability and scalability of the proposed system against its intended objectives. Validation involves testing the system under real-world conditions, measuring performance metrics such as accuracy, turnaround time and resource utilization. The results demonstrated the extent to which the system met the design requirements, supported the expected workload and delivered consistent outcomes.

To ensure reliability, stability and efficiency of the SNMP monitoring system, a series of rigorous performance tests were conducted. These evaluations aimed to identify potential bottlenecks, validate system resilience under varying load conditions and verify long-term sustainability. The primary testing methodologies employed included Load Testing, Spike Testing and Soak Testing. Each of these tests provided valuable insights into the system's

behavior under different stress conditions and helped refine its performance to meet operational requirements.

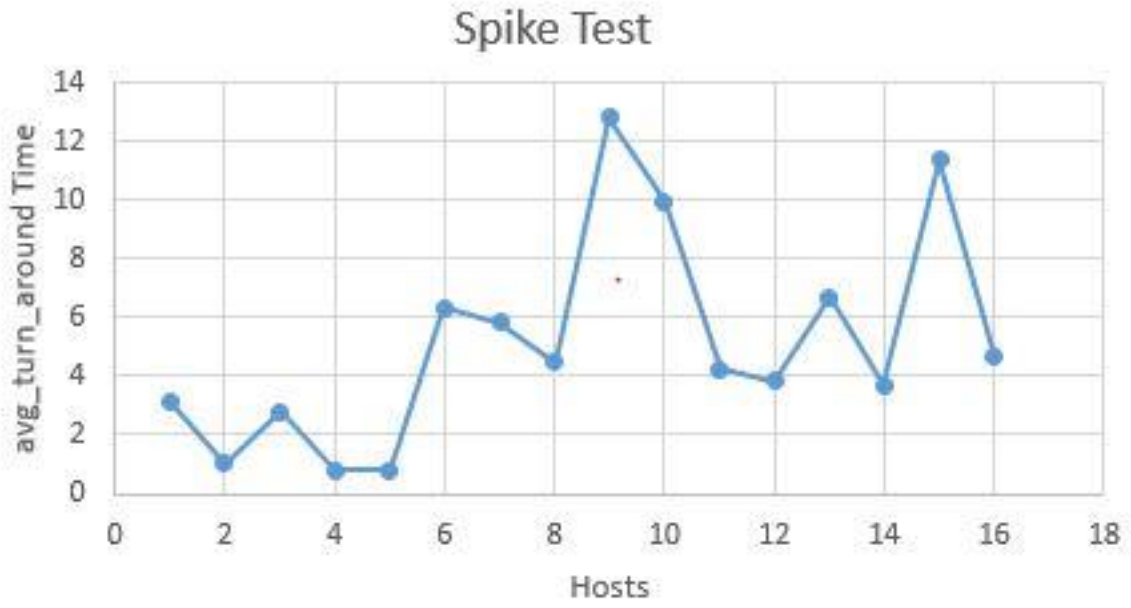
4.4.1 Load Testing

Load Testing was conducted to simulate high-traffic scenarios, ensuring that the system could handle concurrent SNMP poll requests without degradation in performance. By incrementally increasing the number of polling requests, the test measured system throughput, response time, and resource utilization. This evaluation identified the maximum capacity at which the system could function optimally before experiencing slowdowns or failures. The results, as depicted in the Load and Soak Testing graph using 54 hosts (Figure 4.4 below), enabled optimizations such as load balancing, efficient resource allocation and improved query handling mechanisms to enhance overall scalability.

4.4.2 Spike Testing

In addition to gradual load increases, the system was subjected to Spike Testing to assess its ability to handle sudden surges in polling activity. This test involved generating abrupt and substantial increases in SNMP poll requests, simulating real-world scenarios where network traffic might rapidly escalate due to anomalies or unforeseen spikes in demand. The objective was to determine whether the system could absorb such shocks without crashing or exhibiting prolonged instability. Findings from this test, as illustrated in the Spike Test graph for 16 hosts, led to the implementation of robust request queuing

mechanisms, dynamic scaling strategies and efficient error-handling procedures, ensuring system reliability during unexpected load spikes



11Fig 4.3: Spike test for 16 hosts

Source (Author)

4.4.3 Soak Testing

Beyond immediate performance evaluations, Soak Testing was conducted to verify the long-term reliability and sustainability of the system. This test involved running continuous polling operations over extended periods, typically several days or weeks, to monitor for issues such as memory leaks, resource exhaustion or gradual performance degradation. The prolonged testing period helped uncover subtle inefficiencies that might not be apparent during shorter tests. Optimizations resulting from Soak Testing, as evidenced in the Load and Soak Testing graph using 54 hosts, included improved memory management, periodic resource cleanup and enhanced system monitoring, all of which contributed to sustained operational stability.

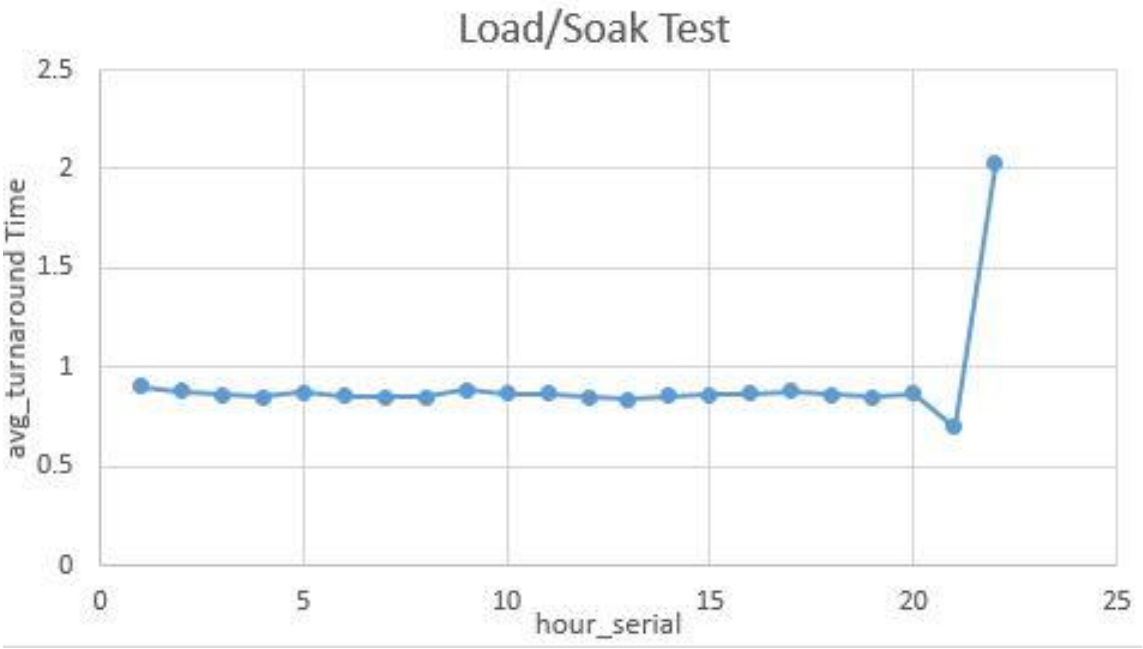


Fig 4.4: Load and soak Testing results using 54 hosts

Source (Author)

By subjecting the SNMP monitoring system to rigorous Load, Spike and Soak Testing, its performance was thoroughly validated under various conditions. These tests not only demonstrated the system's ability to handle high traffic and sudden demand surges but also confirmed its long-term stability and resource efficiency. The insights gained from these evaluations facilitated key optimizations ensuring that the system remains reliable, scalable and resilient in real-world deployments.

4.5 Experimental Results

A subset of the SNMP polling data was stored in the MySQL database table. The data included:

- Host: The IP address or hostname of the device being polled.
- MIB: The SNMP object identifier being polled.

- Value: The returned value from the SNMP query.
- Turnaround Time: The time taken to retrieve the SNMP data in milliseconds.

Host	MIB	Value	Turnaround time
Host-1	sysName.0	Server-A	1.19019
Host-1	sysLocation.0	Data Center	1.02639
Host-2	sysName.0	Server-B	1.86555
Host-2	hrMemorySize.0	33518868	1.92642
Host-2	sysLocation.0	IT-Block	1.42853
Host-3	sysName.0	snmphost	1.58143

Table 4.5: Sample data from MySQL

Source (Author)

The turnaround time is an important metric that measures the system's polling performance, which is a critical factor for monitoring large-scale networks.

For visualizing we created graphs and dashboards displaying: Average turnaround time per host

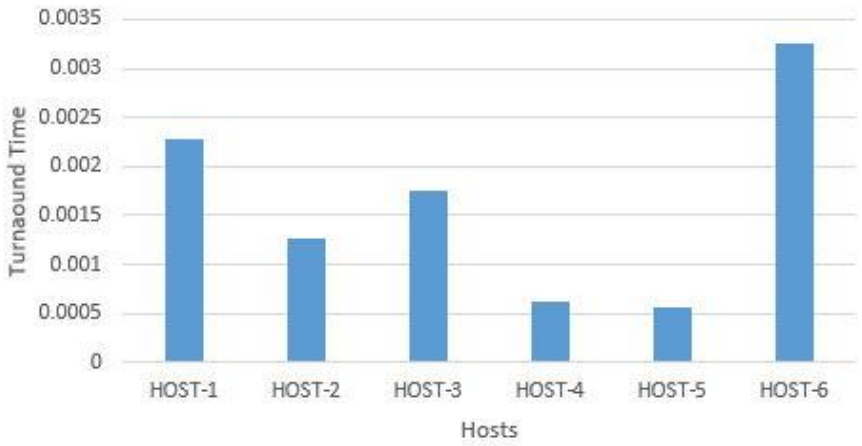


Fig 4.6: Average turnaround time per host

Source (Author)

Graphical visualizations helped network administrators quickly interpret the data and identify potential issues.

4.6 Summary of Results

The results demonstrated that the proposed SNMP monitoring system achieved its objectives by introducing a modular, scalable architecture supported by Kafka-based messaging, multithreaded polling, and selective feature querying. Validation confirmed that each architectural layer—device, data collection, message queuing, and processing—functioned reliably and efficiently, ensuring accurate data handling and persistent storage. Performance testing under load, spike, and soak conditions showed that the system could sustain high traffic, absorb sudden surges, and operate stably over extended periods. Experimental results further confirmed interoperability with heterogeneous devices and provided actionable insights through stored data and visual dashboards. Collectively, these findings establish the system as a resilient and efficient solution for real-time network monitoring.

CHAPTER FIVE

DISCUSION

5.1 Introduction

This chapter discusses the findings of the study. It begins by interpreting the experimental results, highlighting how the proposed monitoring system performed under different testing conditions. It then proceeds to evaluate the results against the study objectives, determining the extent to which the system achieved its intended goals. The chapter then presents the research contributions. This is followed by recommendations to guide future applications and deployments of similar systems. Finally, the chapter identifies directions for future research.

5.2 Discussion of Experimental Results

The experimental evaluation confirmed that the proposed SNMP-based monitoring system delivered accurate, efficient and reliable performance across heterogeneous devices. By incorporating Apache Kafka as a distributed messaging backbone, the system achieved fault-tolerant and asynchronous communication ensuring uninterrupted message flow even under stressed network conditions.

Load testing demonstrated that the architecture sustained stable performance as the number of monitored hosts increased to 54 devices, confirming its scalability. Spike testing highlighted its resilience in absorbing sudden bursts of polling activity without service disruption, while soak testing proved its stability during prolonged operation, with no evidence of memory leaks or performance degradation.

The use of multithreaded polling further reduced turnaround times, averaging less than two seconds per query, thereby improving responsiveness. Additionally, the selective feature polling strategy minimized network overhead and storage consumption, streamlining the monitoring process and ensuring optimal resource utilization.

The graph depicting the average turnaround time for SNMP polling across different hosts (Figure 4.6) provides additional insights into system performance. Turnaround time, a key metric measuring the duration from initiating a poll to receiving data remained consistently low and stable across devices. This consistency highlights the system's effectiveness in managing network bandwidth, processing resources and data retrieval, reinforcing its scalability and operational efficiency.

5.2.1 Latency Breakdown and Polling analysis

From table 4.5 the latency breakdown for host Host-1, the sysName.0 MIB has a turnaround time of 1.19019 seconds, while sysLocation.0 has a faster turnaround time of 1.02639 seconds. This could indicate either variations in network latency or different complexities of MIB data retrieval.

- Given a polling interval of 5 seconds, the polling rate per host is calculated as: -

  Polling Rate (per host) =1/Polling Interval (seconds) = 1/5=0.2polls/second

- Since the experiment involved 54 hosts,

  The total polling rate = Polling rate per host* Number of hosts

  =54 * 0.2

= 10.8polls/second

This indicates that the system is handling 10.8 SNMP requests per second across all monitored hosts.

5.2.1.1 Queue Length Estimation

Using Little's Law expressed as $L=\lambda \times W$

Where

Queue Length (L): Average number of SNMP messages in Kafka.

Waiting Time (W): Average time SNMP data waits before processing.

Service Rate (μ): Kafka's message consumption rate

L= 10.8* 1.503

L = 16.2

Therefore, at any given time there would be 16.2 messages in the queue on average.

- This queue length aligns with the observed turnaround times and helps identify thresholds for performance tuning
- The result helps to validate that the system is processing messages as expected based on the polling rate and turnaround times, and also highlights potential bottlenecks that could occur in the system if the queue length grows too large.

5.2.1.2 Resource Allocation

To enhance performance, Decision Theory can guide resource allocation based on data criticality. For example, if hrMemorySize.0 for Host-2 is critical, its higher turnaround time

(1.92642 seconds) suggests that prioritizing its processing would be beneficial hence employing optimization strategies that may include:

- Dedicating Kafka partitions for critical MIBs
- Additional consumer threads for high-priority SNMP data
- Load balancing to minimize processing delays

5.3 Evaluation Against Objectives

The main objective of this study was to investigate the application of multi-agent systems in network monitoring and to develop a model demonstrating the suitability of autonomous agents in monitoring computer networks. The system designed and implemented in this study reflects this objective by showcasing how decentralized, autonomous agents can effectively manage large-scale, distributed SNMP monitoring tasks. The experimental results validated the model's ability to deliver autonomy, scalability and adaptability, key traits of multi-agent systems, thereby confirming the suitability of this approach for modern network monitoring.

5.3.1 Investigating Existing Multi-Agent Systems

Specific Objective 1: *To Investigate existing Multi-agent based systems used in monitoring computer networks.*

This study examined a range of multi-agent systems designed for network monitoring. Existing approaches were found to rely heavily on centralized or semi-centralized architectures, often requiring extensive configuration and manual intervention. Such

designs tend to be monolithic, leading to performance bottlenecks, scalability limitations, and reduced adaptability in dynamic network environments.

The key weaknesses identified include:

- Scalability: Traditional systems struggled to scale effectively when monitoring large networks or data centers with thousands of devices.
- Flexibility: Adapting to new environments or protocols required substantial manual configuration.
- Autonomy: Limited support existed for self-managing agents capable of dynamically responding to network changes without human intervention.

This analysis revealed significant room for improvement, particularly in developing decentralized and autonomous architectures capable of scaling to meet the demands of modern, dynamic infrastructures. In response, this project adopted a distributed design that directly addressed these limitations, emphasizing scalability, flexibility, and autonomy through the integration of asynchronous messaging and modular, multi-agent processes.

5.3.2 Designing and Developing a Multi-Agent Model Prototype

Specific Objective 2: *To design and develop a Multi-agent based model prototype that will overcome the identified weakness.*

The SNMP polling system developed in this study integrates multi-agent principles by decoupling data collection (SNMP polling) from data processing and storage. Each agent (thread) within the producer script operates autonomously, polling SNMP devices and

pushing data in real time to Apache Kafka. This decentralized model facilitates scalability since additional agents can be introduced dynamically without altering the core architecture.

5.3.2.1 Overcoming Identified Weaknesses:

Scalability: By separating polling from processing, the system allows multiple autonomous agents to operate in parallel. This significantly enhances scalability, as more agents can be deployed to handle increased monitoring demands without creating a central bottleneck. The system achieves horizontal scaling since producers (polling agents) and consumers (processing agents) can be scaled independently.

Autonomy: Each polling agent functions independently, autonomously managing data collection from multiple SNMP devices. This ensures the system can react dynamically to changes in the network, such as device additions, failures, or configuration updates, without requiring manual intervention.

Flexibility: The model adapts easily to heterogeneous SNMP-enabled devices. Simple adjustments to configuration files or the addition of new agents enable integration with diverse environments, avoiding the heavy reconfiguration typically required in traditional systems.

Fault Tolerance: Fault tolerance is supported through Kafka's built-in replication and durability mechanisms, which guarantee message persistence and reliable delivery even in

the event of broker failures. Furthermore, the multithreaded polling design ensures that a failure in one thread does not compromise the operation of the entire system.

Overall, the decentralized architecture of the developed model allows agents to work autonomously while scaling horizontally. This provides a robust, adaptable, and resilient solution that directly addresses the weaknesses identified in existing multi-agent network monitoring systems.

5.3.3 Testing the Effectiveness of the Developed Model

Specific Objective 3: *To test the effectiveness of the developed model.*

he developed prototype was tested in order to evaluate its effectiveness in monitoring large-scale network infrastructures. Performance was assessed using key metrics such as turnaround time, polling frequency, scalability, and reliability under different load conditions. The results demonstrate the model's capability to:

- Handling High Volumes: Efficiently processing thousands of SNMP poll requests without overwhelming network bandwidth or imposing excessive load on monitored devices.
- Scalability: Dynamically scaling as new devices or Management Information Base (MIB) objects were introduced, without requiring significant architectural modifications.
- Reliability: Maintaining stable operation in cases of network or device failure. Messages were safely retained within Kafka until retrieved and processed by the consumer, ensuring no data loss.

Overall, the experimental testing confirmed that the developed multi-agent model was both effective and resilient in achieving large-scale, real-time SNMP monitoring. Its combination of scalability, autonomy, and reliability positions it as a significant improvement over traditional centralized monitoring systems.

Overall, the experimental testing confirmed that the developed multi-agent model was both effective and resilient in achieving large-scale, real-time SNMP monitoring. Its combination of scalability, autonomy and reliability positions it as a significant improvement over traditional centralized monitoring systems.

5.4 Research Contribution

This study makes significant contributions to both academic research and practical applications of network monitoring systems. The first contribution lies in the development of a novel architecture that combines multithreaded SNMP polling with Apache Kafka to create a scalable, fault-tolerant and modular monitoring framework. By organizing the system into four distinct layers; device, data collection, message queuing and processing, the research demonstrates how traditional limitations of centralized SNMP monitoring can be overcome through distributed and resilient design. This architecture not only ensures stability under high polling loads but also provides a foundation for real-time monitoring in dynamic and large-scale environments.

The second contribution is methodological. The implementation of multithreaded polling alongside selective feature querying offers an efficiency-oriented approach to SNMP monitoring. Unlike conventional tools that often rely on exhaustive SNMP walks, the

selective strategy introduced here reduces unnecessary overhead while focusing on actionable metrics. This advancement enhances system responsiveness, minimizes storage requirements and improves the overall resource utilization of monitoring infrastructures, making it better suited for enterprise deployment.

The third contribution concerns empirical validation. The system was subjected to rigorous performance testing, including load, spike and soak tests, which provided concrete evidence of its resilience, scalability and long-term operational stability. Few studies in the domain of SNMP-based monitoring provide such systematic benchmarking and this work therefore fills an important gap by supplying empirical data that validates both the architectural choices and the methodological innovations. The results demonstrate that the system can sustain reliable performance even under demanding conditions, offering practical confidence in its deployment for real-world networks.

Finally, the research contributes to practical applicability by demonstrating interoperability with heterogeneous SNMP-enabled devices and adaptability across diverse network contexts. The system proved effective in both ad-hoc troubleshooting and enterprise-scale environments, showing that it can accommodate a wide spectrum of use cases. This flexibility positions the system as a valuable contribution to the ongoing evolution of network monitoring, where tools must increasingly balance usability, efficiency, and scalability. Furthermore, by establishing a framework that can be extended with analytics, anomaly detection, or machine learning modules, the study lays the groundwork for future advancements in proactive and intelligent network management.

5.5 Limitations of the Study

Despite the system's effectiveness in monitoring and processing SNMP data, certain limitations were identified that could impact its reliability and applicability in real-world scenarios. These limitations primarily concern fault tolerance and the testing environment, which are crucial for ensuring system robustness and scalability. The key limitations observed include:

Fault Tolerance: While Kafka provides fault tolerance at the message-broker level, the system lacked comprehensive fault tolerance mechanisms at the polling agent level. If an agent failed, there were no automatic recovery or failover mechanisms in place.

Testing Environment: The system was tested in a controlled environment. Real-world network conditions, with variable latencies, intermittent connectivity, and more complex configurations, were not fully replicated.

5.6 Future Research

Based on the findings, further improvements are recommended in the following areas to enhance the system’s robustness and analytical capabilities:

Enhanced Fault Tolerance: Future research should focus on implementing comprehensive fault tolerance mechanisms for polling agents, including automatic failover and agent recovery. Strengthening these aspects would improve system resilience and ensure uninterrupted monitoring.

Integration with Advanced Analytics: incorporating machine learning or AI-based analytics could enable predictive failure detection, anomaly detection in network behavior, and proactive monitoring. This would enhance the system's ability to manage large-scale networks with minimal human intervention

5.6 Recommendations

The broader implications of this study extend beyond the specific application of monitoring computer network devices. The findings suggest that multi-agent systems can be effectively deployed in other areas requiring distributed monitoring and autonomous decision-making, such as cloud infrastructure management, IoT device monitoring and smart city infrastructures.

5.7 Conclusion

This chapter has discussed the experimental results, evaluated the system against the study objectives and outlined its research contributions. The findings demonstrated that the proposed multi-agent SNMP monitoring system achieved scalability, reliability and real-time operation, addressing limitations observed in conventional tools. Recommendations were provided for practical deployment and directions for future research were identified to extend the system's applicability.

In conclusion, the study has advanced the field of network monitoring by providing a validated, modular and fault-tolerant architecture that meets modern monitoring

requirements while laying the foundation for further innovation in intelligent, agent-based systems.

REFERENCES

Aarikka-Stenroos, L., & Ritala, P. (2017). Network management in the era of ecosystems: Systematic review and management framework. Industrial Marketing Management, 67, 23-36.

Aboubakar, M., Kellil, M., & Roux, P. (2022). A review of IoT network management: Current status and perspectives. Journal of King Saud University-Computer and Information Sciences, 34(7), 4163-4176.

Arzo, S. T., Akhavan, Z., Esmaeili, M., Devetsikiotis, M., & Granelli, F. (2022). Multi-Agent-Based Traffic Prediction and Traffic Classification for Autonomic Network Management Systems for Future Networks. Future Internet, 14(8), 230.

Arzo, S. T., Bassoli, R., Granelli, F., & Fitzek, F. H. (2021). Multi-agent based autonomic network management architecture. IEEE Transactions on Network and Service Management, 18(3), 3595-3618.

Baer, A., Casas, P., D'Alconzo, A., Fiadino, P., Golab, L., Mellia, M., & Schikuta, E. (2016). DBStream: A holistic approach to large-scale network traffic monitoring and analysis. Computer Networks, 107, 5-19.

Cameron, C. D., Patsios, C., & Taylor, P. (2015). On the benefits of using self-organising Multi-Agent architectures in network management. Paper presented at the 2015 International Symposium on Smart Electric Distribution Systems and Technologies (EDST).

Comer, D. E. (2018). The Internet book: everything you need to know about computer networking and how the Internet works: Chapman and Hall/CRC.

Davarzani, S., Granell, R., Taylor, G. A., & Pisica, I. (2019). Implementation of a novel multi-agent system for demand response management in low-voltage distribution networks. Applied Energy, 253, 113516.

Dorri, A., Kanhere, S. S., & Jurdak, R. (2018). Multi-agent systems: A survey. Ieee Access, 6, 28573-28593.

Hashemi, H., Abdelghany, K. F., & Abdelghany, A. F. (2017). A multi-agent learning approach for online calibration and consistency checking of real-time traffic network management systems. Transportmetrica B: transport dynamics, 5(3), 364-384.

Herrera, M., Pérez-Hernández, M., Kumar Parlikad, A., & Izquierdo, J. (2020). Multi-agent systems and complex networks: Review and applications in systems engineering. Processes, 8(3), 312.

Ilany, A., Booms, A. S., & Holekamp, K. E. (2015). Topological effects of network structure on long-term social network dynamics in a wild mammal. Ecology letters, 18(7), 687-695.

Jayaramaiah, D., Prasanth, A., Reddy, A. V., Basu, D. A., & EPCET, B. (2012). Multi Agent Management System for Next Generation Mobile Networks.[MAMS for NGMN]. International Journal Of Engineering Research & Technology, 1(07).

Kamdar, R., Paliwal, P., & Kumar, Y. (2018). A state of art review on various aspects of multi-agent system. Journal of Circuits, Systems and Computers, 27(11), 1830006.

Larioui, J., & Byed, A. E. (2021). An Agent-based architecture for Multi-modal Transportation Using Prometheus Methodology Design. Paper presented at the Innovations in Smart Cities Applications Volume 4: The Proceedings of the 5th International Conference on Smart City Applications.

Lhafiane, F., Elbyed, A., & Bouchoum, M. (2015). Multi-agent system architecture oriented prometheus methodology design for reverse logistics. International Journal of Computer and Information Engineering, 9(8), 1907-1913.

Li, J.-q., He, S.-q., Ming, Z., & Cai, S. (2015). An intelligent wireless sensor networks system with multiple servers communication. International Journal of Distributed Sensor Networks, 11(8), 960173.

Michel, O., & Keller, E. (2017). SDN in wide-area networks: A survey. Paper presented at the 2017 Fourth International Conference on Software Defined Systems (SDS).

Ndiaye, M., Hancke, G. P., & Abu-Mahfouz, A. M. (2017). Software defined networking for improved wireless sensor network management: A survey. Sensors, 17(5), 1031.

Pache, R. A., Céol, A., & Aloy, P. (2012). NetAligner—a network alignment server to compare complexes, pathways and whole interactomes. Nucleic acids research, 40(W1), W157-W161.

Tammana, P., Agarwal, R., & Lee, M. (2018). Distributed network monitoring and debugging with switchpointer. Paper presented at the 15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18).

Tsai, P.-W., Tsai, C.-W., Hsu, C.-W., & Yang, C.-S. (2018). Network monitoring in software-defined networking: A review. IEEE Systems Journal, 12(4), 3958-3969.

Tso, F. P., Jouet, S., & Pezaros, D. P. (2016). Network and server resource management strategies for data centre infrastructures: A survey. Computer Networks, 106, 209-225.

Wang, J., Jin, Y., Jiang, Z., Hu, X., & Wang, P. (2022). Fraud Network Identification Model for Insurance Industry. Paper presented at the Business Intelligence and Information Technology: Proceedings of the International Conference on Business Intelligence and Information Technology BIIT 2021.

West, J. (2022). Data Communication and Computer Networks: A Business User's Approach: Cengage Learning.

Woodall, W. H., Zhao, M. J., Paynabar, K., Sparks, R., & Wilson, J. D. (2017). An overview and perspective on social network monitoring. IISE Transactions, 49(3), 354-365.

Yang, S.-Y., & Chang, Y.-Y. (2011). An active and intelligent network management system with ontology-based and multi-agent techniques. Expert Systems with Applications, 38(8), 10320-10342.

Marinov V, Schönwälder J. Performance analysis of SNMP over SSH. In Proceedings of the 17th IFP/IEEE International Workshop on Distributed Operation and Management (DSOM 2006). LNCS no. 4269. Springer: Berlin, 2006; 26–36

Soldatos J, Alexopoulos D. Web services-based network management: approaches and the WSNET system. International Journal of Network Management 2007; 17(1): 3

Kasula. C. Pramodh, I. Nikhil & J. R. Singh, "Implementation of SNMP-JSON translator and integrating SNMP agents with JSON based network management system," 2017 7th International Conference on Communication Systems and Network Technologies (CSNT), Nagpur, India, 2017, pp. 67-73, doi: 10.1109/CSNT.2017.8418513.3–50.

Wojciech K. Learning Nagios 3.0. Birmingham, B27 6PA, UK: Packet Publishing Ltd; 2008 .

Bilal Zaka & Christian Safran (2008) "Emerging Web Based Learning Systems and Scalability Issues" International Conference on Computer Science and Software Engineering, doi: 10.1109/CSSE.2008.187

A. Al-Said Ahmad & P. Andras, ‘‘Scalability analysis comparisons of cloud-based software services,’’ J. Cloud Comput., vol. 8, no. 1, pp. 1–17, Dec. 2019

T. Hossfeld, P. E. Heegaard, & W. Kellerer, "Comparing the Scalability of Communication Networks and Systems," IEEE Access, vol. 11, pp. 101474-101497, 2023, doi: 10.1109/ACCESS.2023.3314201.

R. P. Curasma & H. P. Curasma, "Assessment and proposal of a network monitoring system based on free software," 2020 IEEE Engineering International Research Conference (EIRCON), Lima, Peru, 2020, pp. 1-4, doi: 10.1109/EIRCON51178.2020.

Y. G. Shan, L. Chao, G. Guangjian, & F. Gao, "Research on Monitoring of Information Equipment Based on Zabbix for Power Supply Company," 2021 3rd International Conference on Applied Machine Learning (ICAML), Changsha, China, 2021, pp. 487-491, doi: 10.1109/ICAML54311.2021.00108.

T. P. Raptis and A. Passarella, "A Survey on Networked Data Streaming With Apache Kafka," in IEEE Access, vol. 11, pp. 85333-85350, 2023, doi: 10.1109/ACCESS.2023.3303810.

Khan, Faisal. (2021). Apache kafka with real-time data streaming.

Brocke, Jan vom & Hevner, Alan & Maedche, Alexander. (2020). Introduction to Design Science Research. 10.1007/978-3-030-46781-4_1.

Appendix 1: Environment preparation

System requirements

a) Server Requirements (Kafka Broker, Zookeeper, and MySQL Database)

- Processor: Minimum 8-core CPU (Intel Xeon / AMD EPYC)
- RAM: At least 32 GB (recommended: 64 GB for scalability)
- Storage: Minimum 500 GB SSD (recommended: 1 TB for high-volume SNMP logs)
- Network Interface: 1 Gbps Ethernet or higher for efficient SNMP data transmission
- Operating System: Ubuntu 20.04 LTS or higher / CentOS 7+

b) SNMP Polling Hosts (Producers and Consumers Running on Separate Machines or Containers)

- Processor: Minimum 4-core CPU
- RAM: 8 GB minimum (16 GB recommended for large-scale polling)
- Storage: 200 GB SSD (to store temporary logs before transmission)
- Operating System: Ubuntu 20.04 LTS / CentOS 7+
- Network: 1 Gbps Ethernet

Installation Guide for a) and b)

1. Install scheduler library (can be obtained from https://pypi.org/project/schedule/)

   This library will be used to control the poll interval of the system.

a. `pip install schedule`

2. Install easysnmp library (can be obtained from https://pypi.org/project/easysnmp/0.2.5/) This library will be used for querying SNMP data from monitored devices
    a. `pip install easysnmp`
3. Install python library library since there is no official package on apt present for installation you will have to add a personal package archive (ppa) and then do installation from it
    a. `sudo add-apt-repository ppa:deadsnakes/ppa`
    b. `sudo apt update`
    c. `sudo update-alternatives --install /usr/bin/python python /usr/bin/python3.11 1`
    d. `sudo apt install python3.11-dev python3.11-venv python3.11-distutils python3.11-gdbm python3.11-tk python3.11-lib2to3`
    e. `sudo apt update`
4. Install apache kafka
    a. This can be done using the installation script bellow (Kafka Installation Script below will Install Java. Download and extract Kafka, set up environment variables so Kafka commands are available in your PATH and Start Zookeeper and the Kafka broker as background processes.)

```
#! /bin/bash
# Variables
```

```
KAFKA_VERSION="3.5.1"
SCALA_VERSION="2.13"
KAFKA_HOME="/usr/local/kafka"
DOWNLOAD_URL="https://downloads.apache.org/k
afka/${KAFKA_VERSION}/kafka_${SCALA_VERSION}
-${KAFKA_VERSION}.tgz"
# Update system and install Java
echo "Updating system and installing
Java..."
sudo apt update
sudo apt install -y openjdk-11-jdk wget
# Download Kafka
echo "Downloading Kafka..."
wget $DOWNLOAD_URL -O /tmp/kafka.tgz
# Extract Kafka
echo "Extracting Kafka..."
sudo tar -xzf /tmp/kafka.tgz -C /usr/local
sudo mv /usr/local/kafka_${SCALA_VERSION}-
${KAFKA_VERSION} $KAFKA_HOME
# Set up environment variables
echo "Setting up environment variables..."
echo "export KAFKA_HOME=$KAFKA_HOME" | sudo
tee -a /etc/profile
```

```
echo "export PATH=\$PATH:\$KAFKA_HOME/bin" |
sudo tee -a /etc/profile
source /etc/profile

# Start Zookeeper
echo "Starting Zookeeper..."
$KAFKA_HOME/bin/zookeeper-server-start.sh -
daemon
$KAFKA_HOME/config/zookeeper.properties
# Start Kafka Broker
echo "Starting Kafka broker..."
$KAFKA_HOME/bin/kafka-server-start.sh -
daemon $KAFKA_HOME/config/server.properties
echo "Kafka installation is complete!"
echo "To create a topic, use the following
command:"
echo "$KAFKA_HOME/bin/kafka-topics.sh --
create --topic your_topic_name --bootstrap-
server localhost:9092 --partitions 1 --
replication-factor 1"
```

5. Install mysql database
    a. sudo apt update
    b. sudo apt install mysql-server –y

Monitored Hardware Devices

Note that:

- Routers & Switches: Must support SNMP v2c or SNMP v3 for secure polling
- Computers: Windows (Windows 10, 11, Server 2016/2019), Linux (Ubuntu, CentOS, Debian)
- Servers: Windows Server / Linux Server with SNMP enabled

1. To install/ enable and configure snmp on the end devices ensure that you Specify the community string
    a. on Ubuntu
        i. `sudo apt update`
        ii. `sudo apt install -y snmp snmpd`
    b. on windows
        i. Add-WindowsCapability -Online -Name SNMP.Client~~~~0.0.1.0 (This can be run on windows power shell)

Appendix 2: System Sample Code

Producer script

```
import json
import csv
import threading
import time
from datetime import datetime
from easysnmp import Session
from confluent_kafka import Producer

# Kafka producer configuration
kafka_conf = {
    'bootstrap.servers': 'localhost:9092',
    'client.id': 'snmp-polling-producer'
}

def create_producer():
    return Producer(kafka_conf)

def poll_and_produce(host, com, ver, mib, freq):
    session = Session(hostname=host, community=com, version=ver)
    producer = create_producer()

    while True:
        start_time = time.time()  # Start time for turnaround
calculation
        try:
            # Start the polling process
            output = session.get(mib)
```

```
            # Get the current date and time
            poll_date = datetime.now().date().isoformat()  # ISO
format for the date
            poll_time = datetime.now().time().isoformat()  # ISO
format for the time
            result = {
                'host': host,
                'mib': mib,
                'value': output.value,
                'poll_date': poll_date,  # Include poll date
                'poll_time': poll_time   # Include poll time
            }


            end_time = time.time()  # End time for turnaround
calculation
            turnaround_time = (end_time - start_time) * 1000  #
Calculate turnaround time in milliseconds


            # Display the result with turnaround time
            print(f"Polling result: Host: {result['host']}, MIB:
{result['mib']}, Value: {result['value']}, "
                  f"Turnaround Time: {turnaround_time:.2f} ms,
Poll Date: {poll_date}, Poll Time: {poll_time}")


            # Add turnaround time to the result
            result['turnaround_time_milliseconds'] =
turnaround_time


            # Send result to Kafka on the snmp_thread topic
            producer.produce('snmp_thread', json.dumps(result))
            producer.flush()  # Ensure the message is sent
immediately
```

```
            # Wait for the specified frequency before the next
poll
            time.sleep(freq)

        except Exception as e:
            # Output error messages for troubleshooting
            print(f"Error polling {host} for {mib}: {e}")
            time.sleep(freq)  # Wait before retrying to avoid
overwhelming the agent

def load_inventory_and_start_polling(filename):
    with open(filename) as inventory:
        invcsv = csv.reader(inventory)
        for row in invcsv:
            if len(row) < 5:
                continue

            host = row[0]
            freq = int(row[1])
            com = row[2]
            ver = int(row[3])

            # Start a thread for each MIB in the row
            for mib in row[4:]:
                thread =
threading.Thread(target=poll_and_produce, args=(host, com, ver,
mib, freq))
                thread.start()

if __name__ == "__main__":
    load_inventory_and_start_polling('inventory.csv')
```

Consumer script

```
import json
from confluent_kafka import Consumer, KafkaError
import mysql.connector


# MySQL connection setup
db_config = {
    'user': 'root',
    'password': 'testmachine',
    'host': 'localhost',
    'database': 'snmp'
}


def insert_poll_result(host, mib, value, turnaround_time,
poll_date, poll_time):
    try:
        conn = mysql.connector.connect(**db_config)
        cursor = conn.cursor()


        # Updated query to insert poll_date and poll_time
        query = """
        INSERT INTO snmp_demo2025 (host, mib, value,
turnaround_time, poll_date, poll_time)
        VALUES (%s, %s, %s, %s, %s, %s)
        """


        # Executing the insert query with poll_date and poll_time
        cursor.execute(query, (host, mib, value, turnaround_time,
poll_date, poll_time))
        conn.commit()
```

```
        cursor.close()
        conn.close()
        print(f"Inserted result: host={host}, mib={mib},
value={value}, turnaround_time={turnaround_time:.2f} ms, "
              f"poll_date={poll_date}, poll_time={poll_time}")
    except mysql.connector.Error as err:
        print(f"Error: {err}")

# Kafka consumer configuration
kafka_conf = {
    'bootstrap.servers': 'localhost:9092',
    'group.id': 'snmp-polling-consumer',
    'auto.offset.reset': 'earliest'
}

def create_consumer():
    consumer = Consumer(kafka_conf)
    consumer.subscribe(['snmp_thread'])  # Subscribe to the
snmp_thread topic
    return consumer

if __name__ == "__main__":
    consumer = create_consumer()

    # Continuously consume messages from Kafka and insert into
the database
    while True:
        msg = consumer.poll(1.0)  # Poll with a 1-second timeout
        if msg is None:
            continue
        if msg.error():
```

```
            if msg.error().code() == KafkaError._PARTITION_EOF:
                continue
            else:
                print(msg.error())
                break
        # Decode and process the JSON message
        try:
            result = json.loads(msg.value().decode('utf-8'))  # 
Directly load JSON


            # Safely retrieve data with .get()
            host = result.get('host', 'Unknown')
            mib = result.get('mib', 'Unknown')
            value = result.get('value', None)
            turnaround_time = 
result.get('turnaround_time_milliseconds', None)
            poll_date = result.get('poll_date', None)  # Retrieve 
poll_date
            poll_time = result.get('poll_time', None)  # Retrieve 
poll_time


            # Ensure we have the required data before inserting
            if value is not None and turnaround_time is not None:
                insert_poll_result(host, mib, value, 
turnaround_time, poll_date, poll_time)
            else:
                print(f"Received incomplete data: {result}")
        except json.JSONDecodeError as e:
            print(f"Failed to decode JSON message: {e}")
        except Exception as e:
            print(f"An unexpected error occurred: {e}")
    consumer.close()
```

Sample Device file (inventory.csv)