HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

Project I

COLLECT SNMP DATA FROM SERVER AND NETWORK DEVICES

Chu Trung Anh
Anh.CT225564@sis.hust.edu.vn

Phan Khoi Nguyen
Nguyen.PK225582@sis.hust.edu.vn

Major: Cybersecurity IT-E15

Supervisor: MSc Nguyen Quoc Khanh ______________

Signature

Department: Computer Engineering

School: School of Information and Communications Technology

HANOI, 05/2024

THESIS ASSIGNMENT

1. Student's information:

Name: Chu Trung Anh email: Anh.CT225564@sis.hust.edu.vn

Name: Phan Khoi Nguyen email: Nguyen.PK225582@sis.hust.edu.vn

Class: IT-E15 01 K67

Affiliation: Hanoi University of Science and Technology

Duration: 1/05/2024–09/07/2024

2. Thesis title: Collect SNMP Data From Server And Network Devices

3. Thesis statement: The thesis aims to investigate how SNMP (Simple Network Management Protocol) can effectively gather extensive data from server and network devices. Our approach involves developing an SNMP browser in Java to demonstrate SNMP operations and parse the returned values into a human-readable format. This practical implementation will exemplify the practical applications and benefits of SNMP in network management.

4. Declarations/Disclosures:

We, Chu Trung Anh and Phan Khoi Nguyen, declare that:

1. This thesis titled "Collect SNMP Data From Server And Network Devices" represents our original work conducted under the supervision of MSc Nguyen Quoc Khanh at Hanoi University of Science and Technology.
2. The content presented is our own academic endeavor, based on research and understanding of SNMP and its application in network management.
3. All sources used in this thesis, including publications and articles, have been appropriately cited and acknowledged.
4. Any opinions, findings, conclusions, or recommendations expressed herein are solely those of the authors and do not necessarily reflect the views of Hanoi University of Science and Technology or any other entity.
5. This thesis or any part of it has not been submitted for any other degree or qualification at any university or institution.
6. We affirm that all content in this thesis is original and properly attributed, understanding the severity of plagiarism in academic contexts.
7. We grant Hanoi University of Science and Technology permission to archive and provide access to this thesis for educational and research purposes.

Hanoi, date month year 2024
Authors

Chu Trung Anh Phan Khoi Nguyen

5. Attestation of thesis advisor:

. . . .

. . . .

Hanoi, date month year 2024
Thesis advisor

MSc Nguyen Quoc Khanh

ACKNOWLEDGMENT

The successful completion of this report would not have been possible without the invaluable contributions of several individuals and groups. I extend my heartfelt appreciation to my academic advisor, MSc. Nguyen Quoc Khanh, for his expert guidance and constructive feedback throughout the project and writing process. His support was instrumental in shaping this work.

Additionally, I am grateful for the unwavering support of my family and friends. Their encouragement and understanding were essential in maintaining my motivation throughout this project.

ABSTRACT

This thesis addresses the challenge of collecting SNMP data from server and network devices, a critical aspect in both Cybersecurity and IoT domains. Despite the availability of numerous software solutions, both paid and open-source, our team has developed a custom application for educational purposes and to meet specific requirements.

In this project, we endeavor to implement an SNMP browser application using the Java programming language with the JavaFX framework for the user interface. Our application supports SNMP Get, Get Next, and Walk operations using the snmp4j library [8] to send and receive SNMP values. These values are then parsed into human-readable format based on MIB files provided in JSON format. Initially in ASN1 format, we converted these MIB files to JSON using a script based on the Python tool, pysmi [6]. Our database currently holds 70 MIB files essential for the software's functionality. These JSON files are read and processed using the Jackson library [2].

The primary contribution of this thesis is the development of a custom SNMP browser application tailored to our educational objectives and specific needs. The results demonstrate that our solution is capable of efficiently retrieving and interpreting SNMP data, offering a more customizable and insightful approach compared to existing methods.

Student

(Signature and full name)

TABLE OF CONTENTS

CHAPTER 1. Introduction

1.1 Problem Statement

We highlight three prevalent challenges in network management, particularly significant for medium and large organizations facing overwhelming computer infrastructures. These organizations require an optimal, effective, and centralized solution to manage all hardware devices across their networks.[12]

1. Monitoring Server Resources

With thousands of servers running various operating systems (OS), how can we monitor resources (CPU usage, remaining hard drive space, RAM usage, etc.) across all servers daily and hourly to promptly detect servers at risk of overload? Connecting individually to each server isn't feasible due to their sheer number and diverse OS-specific monitoring methods.

2. Monitoring Traffic on Switch and Router Ports

Managing thousands of network devices from different manufacturers, each with multiple ports, poses the challenge of monitoring continuous traffic flow across all ports 24/7. How can we detect and address potential port overloads in a timely manner without direct access to issue commands due to varying device-specific protocols?

3. Automated Instant Fault Warning System

With numerous network devices in operation, instant detection of events such as port failures (link down), failed login attempts, or device restarts is crucial. How can administrators receive real-time alerts for such events as they occur?

Note: This problem differs from the second problem above in that it focuses on detecting unknown events as they happen, rather than continuously updating known information.

1.2 Background and Problems of Research

Network management protocols are crucial for defining the rules and procedures used to monitor, maintain, and manage computer networks. These protocols help ensure smooth communication across the network, addressing issues and maintaining stable connections. They are also essential for troubleshooting connectivity problems between hosts and clients.

To solve mentioned challenges above, various protocols have been developed to manage network devices effectively and meet the specific needs of both individuals and enterprises. Key protocols include SNMP, NETCONF, RESTCONF, gNMI, SDN, Collectd, Prometheus, etc ... all used for monitoring servers and network devices. Among these, SNMP (Simple Network Management Protocol) is the oldest, having been used for nearly four decades. However, due to its age, many manufacturers nowadays have stopped supporting SNMP in favor of newer, more modern protocols that address its limitations.

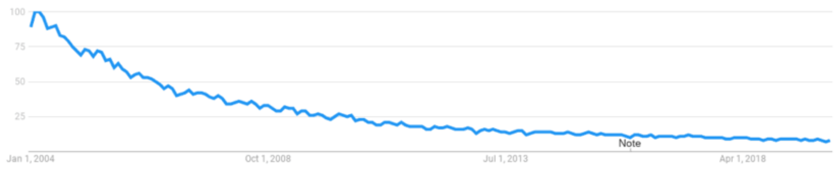


Figure 1.1: Picture showing interest over for the SNMP from 2004 till 2019 [9]

For example, Microsoft announced the deprecation of SNMP on Windows servers in 2012, as detailed in their documentation. Similarly, in 2018, Google highlighted SNMP's obsolescence in their paper "SNMP is Dead."

The concerns raised by Microsoft and Google are valid, as SNMP is a 30-year-old protocol with many flaws and security issues. Despite these problems, SNMP remains in use because it is familiar, lightweight, easy to use, free, and supported by many vendors. Its long history and wide acceptance have helped it persist, even as newer protocols are developed to aim to replace it.

While it is clear that SNMP is becoming less relevant, its complete phase-out will take time. The rise of the Internet of Things (IoT) may even prolong its use by introducing new applications for the protocol.

Researching more about this protocol, here are some benefits and also limitations we found:

Benefits:

- **Interoperability and Support:** SNMP is widely supported and works with many devices and vendors.
- **Resource Efficiency:** It requires minimal resources and bandwidth, making it suitable for various network environments.
- **Standardization:** SNMP provides a standardized way to collect and configure network information, helping automate and integrate network management tasks.

Limitations:

- **Complex Operations:** SNMP is not suitable for complex operations like configuration management, software updates, or remote execution.
- **Reliability Issues:** It can be affected by network congestion, packet loss, and latency, reducing the reliability and timeliness of SNMP messages.
- **Security Risks:** If not configured correctly, SNMP can pose security risks, exposing sensitive information or allowing unauthorized changes to network devices.

Despite its limitations, SNMP's simplicity in development and operation makes it ideal for educational purposes, even if it may not be the most practical solution in real-world scenarios. Currently, there are numerous widely-used SNMP network manager applications available, both open-source and paid, such as iReasoning, PRTG, ManageEngine, SolarWinds, etc. We will leverage the best features of these software solutions to develop our own server for educational purposes.

1.3 Research Objectives and Conceptual Framework

The main goal of this project is to manage network devices by collecting and analyzing data using the SNMP protocol. To achieve this, we will create an SNMP browser application using Java. This application will perform SNMP operations, retrieve responses, process the data, and display it to users. This project will demonstrate how SNMP can be used in network management, despite its limitations.

1.4 Organization of Thesis

The remainder of this report will be organized as follows:

Chapter 2: Theory of SNMP

This chapter explores SNMP in depth, covering its definition, mechanisms, and detailed structure. Special emphasis is placed on the MIB (Management Information Base) tree structure, as it directly relates to the application we have developed.

Chapter 3: Application

In this chapter, we provide a detailed usage of application features as well as explanation of the structure of the application we have developed. We discuss the implemented classes and their interactions with each other, illustrating how they contribute to the functionality of the application.

Chapter 4: Conclusion

This final chapter summarizes all the findings and results obtained from the project. It provides a comprehensive overview of the key outcomes and insights gained. Additionally, it offers several suggestions for future work, highlighting potential areas for further research and development.

CHAPTER 2. Theoretical Analysis

2.1 Introduce to SNMP

SNMP stands for **Simple Network Management Protocol**. It is an Internet Standard protocol used for collecting and organizing information about managed devices on IP networks and for modifying that information to change device behavior. Despite being first introduced in 1988, over 37 years ago, SNMP remains widely used today in both individual and enterprise environments. This enduring popularity persists even though many modern protocols have been developed to replace it. The reasons why SNMP is still used commonly include its standardization, reliability, simplicity, scalability, flexibility, wide range of support, real-time monitoring capabilities, and enhanced security features in the latest version. Particularly in the era of IoT (Internet of Things), SNMP plays a crucial role in network management.

Since SNMP is a protocol, it includes a set of specific rules that all participants must follow in order to communicate with each other. Specifically, these rules define the structure and format of the data stream, as well as the sequence of operations. If one participant sends data in an incorrect format or out of sequence, other participants will not understand or may refuse to exchange information. A device that understands and adheres to the SNMP protocol is referred to as "SNMP supported" or "SNMP compatible."[14]

What does SNMP use for ?

SNMP was primarily developed for network management, allowing administrators to monitor network state and remotely modify setting, configuration on network equipment such as: switcher, routers, server, printers or some others uninterruptible power supplies devices in general. It enables monitoring, information retrieval, notifications, and operational control to ensure systems operate as desired. For example, SNMP software can:

- Monitor the data transmission speed of a router and track the total bytes transmitted/received.
- Retrieve information about server storage capacity, including available disk space
- Automatically receive alerts when a switch port goes down.
- Control and disable ports on a switch.

- . . .

SNMP is used for network management, meaning it is designed to operate over TCP/IP and manage TCP/IP-connected devices. These network devices can include not only computers but also switches, routers, firewalls, ADSL gateways, and various software platforms that support SNMP administration. There are no limitations to what SNMP can manage as long as the device is network-connected and supports SNMP.

SNMP is also engineered to function independently of the architectures and mechanisms of SNMP-supported devices. While different devices may operate differently, their responses to SNMP are standardized, regardless of whether they run on Windows, Linux, macOS or other operating systems.

SNMP is a straightforward protocol, designed with simplicity in message structure and operational procedures, as well as simplicity in security (excluding SNMP version 3, which will be explored later). This design facilitates rapid and cost-effective development of SNMP software solutions, also require minimum processor on devices to operate effectively.

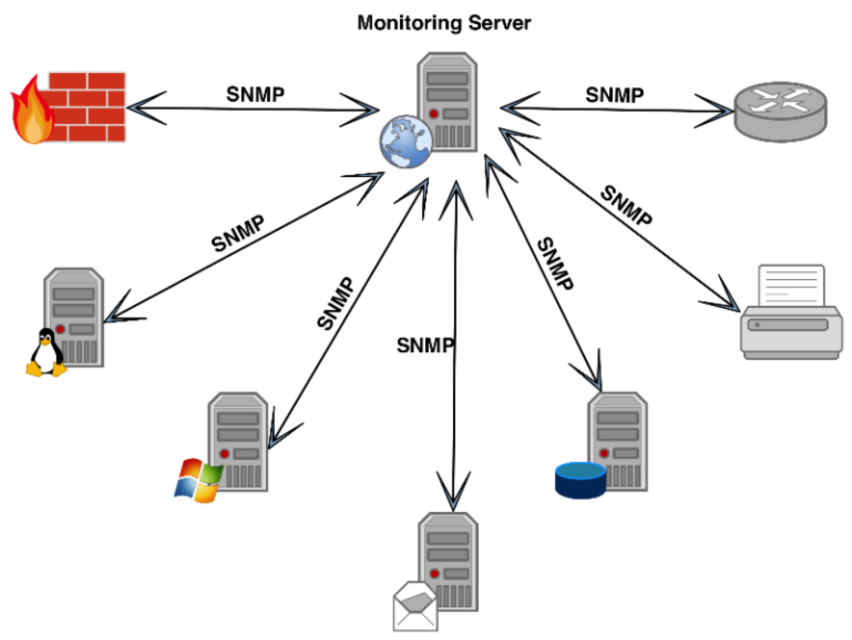


Figure 2.1: SNMP is capable of managing a wide range of devices

SNMP Components

SNMP has two main components:

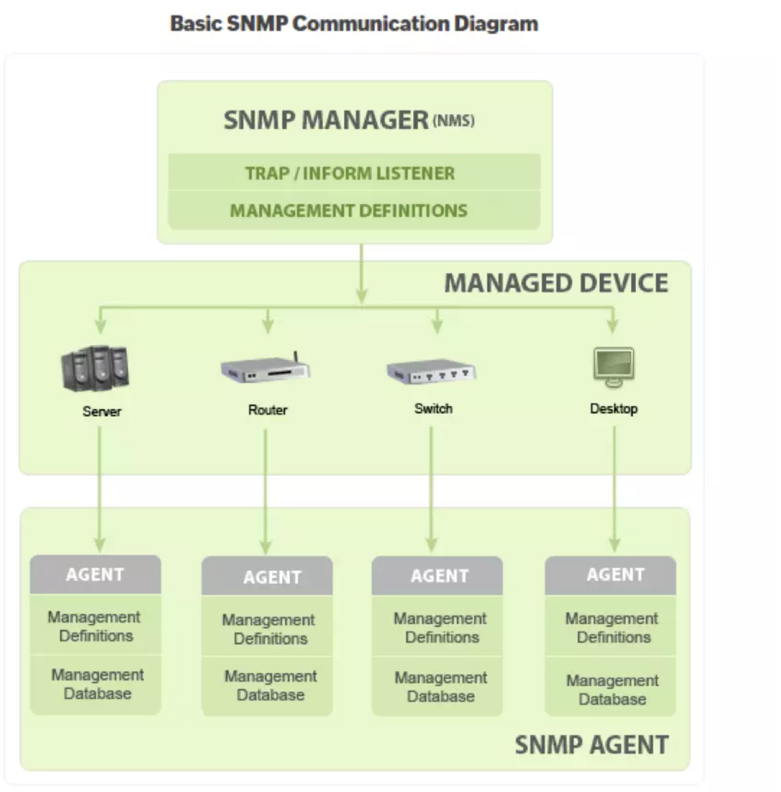


Figure 2.2: SNMP components

a. Manager

A Network Management System, or NMS, for short, is an independent entity responsible for communicating with network devices equipped with SNMP agents. Typically, this is a computer used to run one or more network management systems

b. Agent

An SNMP agent is a program embedded within network devices such as computers, servers, firewalls, and others hardware devices like routers and switches. Activating the agent allows it to collect Management Information Database (MIB) data from the local device and provide it to an SNMP manager upon request. These agents can be standard (e.g., Net-SNMP) or vendor-specific (e.g., HP Insight Agent).

Key functions of SNMP agents include:

- Collecting management information on operational metrics of the device.
- Storing and retrieving management information as defined in the MIB.
- Signaling events to the management system.
- Acting as a proxy for some SNMP-unmanaged network nodes.

Characteristics of an Agent:

An agent can interact with multiple objects/sensors, some of which adhere to industry standards while others are vendor-specific. Each object is identified by an OID (Object Identifier), which is stored alongside some of its attributes like name, data type, status, description, etc. in a MIB file.

For example, an agent might have objects like sysDesc, ifPhysAddress, hrMemorySize, etc. Each identified by an OID (e.g., .1.3.6.1.2.1.1.1 for sysDesc).

How these components work together

SNMP operates over UDP/IP for data transport services. An SNMP Manager application must identify the specific Agent it needs information from. An Agent application is recognized by its IP address. Messages sent between them use the Agent's IP address and the Manager's UDP port (often port 161 or 162).

SNMP employs three fundamental commands: Read, Write, and Trap, alongside additional customize commands for device management. Through these commands, SNMP manages and monitors devices by modifying or gathering information stored in variables on the devices.

Agents installed on devices interact with SNMP-supporting control chips to facilitate reading from or writing to device contents.

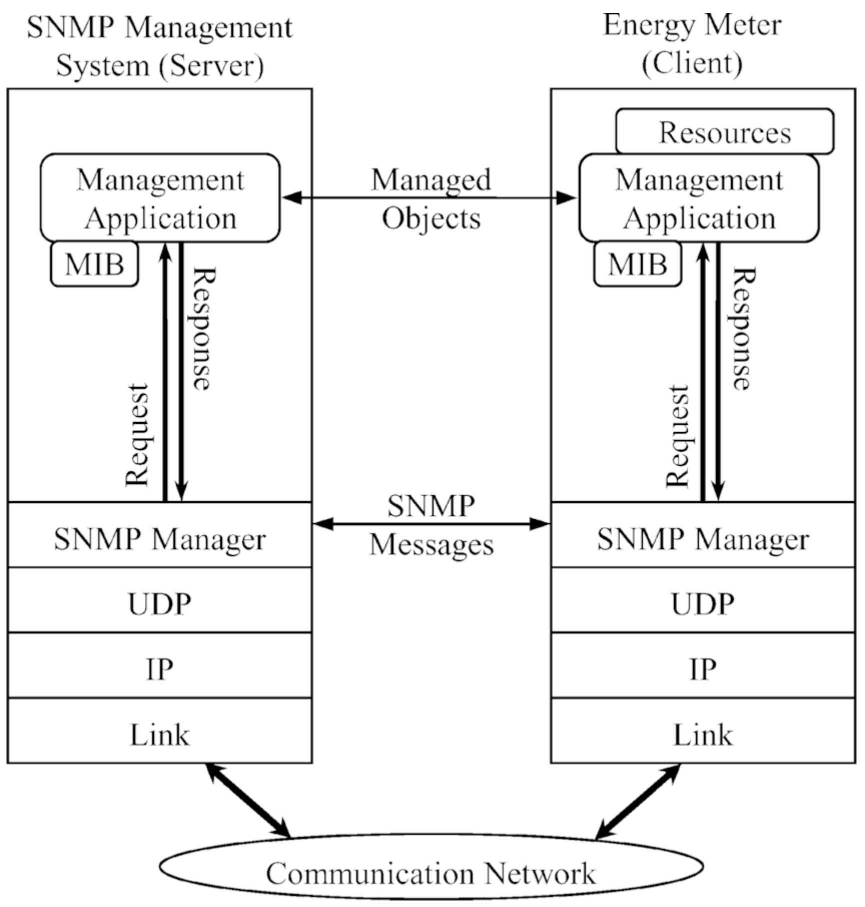


Figure 2.3: SNMP components work

2.2 Network Monitoring With SNMP

SNMP-capable devices provide various types of information, each represented as an Object with a unique identifier known as an Object Identifier (OID). These OIDs may follow industry standards or be defined by individual device manufacturers. Understanding OIDs and MIB (which will be elaborated on in the next section), is crucial for retrieving information from an Agent by a Network Management System (NMS). Here's an illustration of how this process works:

1. **Request Initiation**: The SNMP Manager initiates a request to the SNMP Agent, specifying the OID of the Object it wishes to retrieve (e.g., OID "1.3.6.1.2.1.1.5.0" to fetch the device name).

2. **Agent Response**: Upon receiving the request, the SNMP Agent processes the OID:

   - If the Agent supports the OID, it retrieves the corresponding information associated with that OID.

   - If the Agent does not recognize or support the OID, it does not respond.

3. **Response Transmission**: The SNMP Agent sends a response message back

to the SNMP Manager, containing the requested information. This response includes the OID and the associated data, such as the device name or other specified information.

This process is visualized in the following diagram:

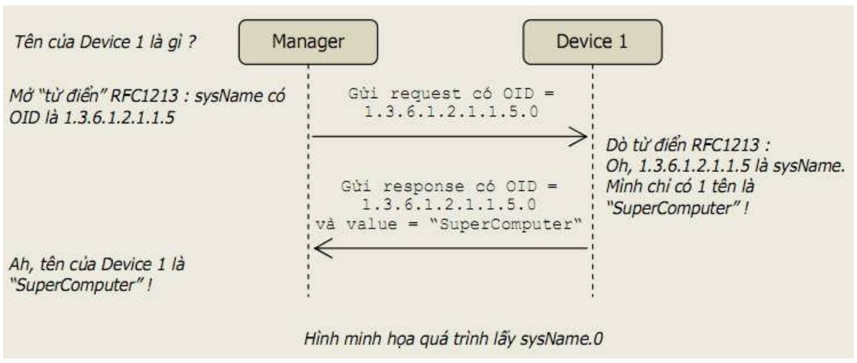

Tên của Device 1 là gì ?
Manager
Device 1
Mở "từ điển" RFC1213 : sysName có OID là 1.3.6.1.2.1.1.5
Gửi request có OID = 1.3.6.1.2.1.1.5.0
Dò từ điển RFC1213 : Oh, 1.3.6.1.2.1.1.5 là sysName. Mình chỉ có 1 tên là "SuperComputer" !
Gửi response có OID = 1.3.6.1.2.1.1.5.0 và value = "SuperComputer"
Ah, tên của Device 1 là "SuperComputer" !
Hình minh họa quá trình lấy sysName.0

Figure 2.4: Communication between NMS and Agent

*Note that in the second step, the action that NMS can interpret to an agent is based on the access level of that OID. If it is **READ_ONLY**, NMS can only retrieve the information from it without any modification allowed. However, if it is **READ_WRITE**, NMS can perform both retrieval and modification actions. For example, in practical scenarios: the device name (sysName) is typically **READ_WRITE** as it is just digital representation of the device, while attributes like the number of ports (ifNumber) are **READ_ONLY** since it is the physical exits of the device*

SNMP Request

Previously, we explored how NMS and agents communicate through "requests." Let's delve deeper into this terminology. SNMP requests include commands like Get, Get Next, Walk, Trap, etc., typically categorized into two main types:

1. Polling Method

- **Operation:** The NMS actively requests information from the agent, receiving the data over UDP port **161**.
- **Principle:** This method involves the NMS regularly querying devices for updates. The agent responds only when queried, ensuring the NMS has current information.

2. Alert/Notification Method

- **Operation:** Agents independently send messages to the NMS to report events without prior requests. This communication occurs via UDP port **162**.
- **Principle:** In this method, devices automatically alert the NMS when spe-

cific events occur, such as critical issues or errors, without needing ongoing requests.

Devices transmit notifications selectively for events and do not send continuous data updates unless polled by the NMS. This dual method ensures efficient network management and real-time monitoring capabilities through SNMP.

These methods of communication ensure efficient network management and real-time monitoring of devices through SNMP, particularly the alert methods. They are extremely useful for simplifying network monitoring by automating notifications of critical events such as router downtime, thereby eliminating the need for continuous 24/7 checks. This reduces monitoring overhead and optimizes resource efficiency.

a, SNMP Requests in Detail

Let's delve deeper into specific messages within the two main method groups.

Polling Methods

- **Get** - Retrieve the current value of a specific variable or set of variables from an SNMP agent by specifying their OID (a Get request can have one or many OIDs in one time sending). The response from the agent will contain with the current value(s) associated with the specified OID.
- **GetNext** - retrieves the variables that come immediately after the provided OID in the MIB tree structure inside the device (This tree structure is embedded inside the device, containing all OIDs that it can support)

  Why is GetNext necessary? A MIB consists of sequentially ordered but not contiguous OIDs. If we know one OID, you cannot determine the next OID by increasing the last index. For example, in RFC1213, the object ifSpecific and the object atIfIndex are next to each other but their OIDs is 1.3.6.1.2.1.2.2.1.22 and 1.3.6.1.2.1.3.1.1.1 respectively. The only way is by exploring the tree using GetNext to find the next OID
- **Walk** - retrieves a sequence of variables from an SNMP agent, starting from a specified OID and traversing all subtrees until the entire tree is fully explored. It achieves this by iterative fetching variables using Get Next requests based on the OID returned in each response. While efficient for comprehensive data gathering in network monitoring and management, SNMP Walk can be resource-intensive if the device's MIB tree structure is large-scale or complex.
- **GetBulk** - Introduced in SNMP Version 2, this is an enhanced version of GetNext request designed for efficiency. The response includes bulk data as re-

quested, optimizing the retrieval process for multiple GetNext at once, means retrieving a large set of variables from an SNMP agent in a single transaction, reducing network overhead and improving performance, especially with complex MIB structures or large datasets like interface or routing tables. By specifying a starting OID and the maximum number of variables to fetch, it streamlines the process of gathering extensive monitoring data crucial for network operations.

- **SetRequest** - enables the NMS to remotely configure or modify variables on SNMP-enabled devices. It plays a pivotal role in network management by allowing administrators to adjust device parameters, initiate actions, or update configurations. This bidirectional communication ensures devices operate according to network requirements and allows for centralized control over distributed network elements. *Note: Only objects with READ_WRITE permissions can have their values modified.*

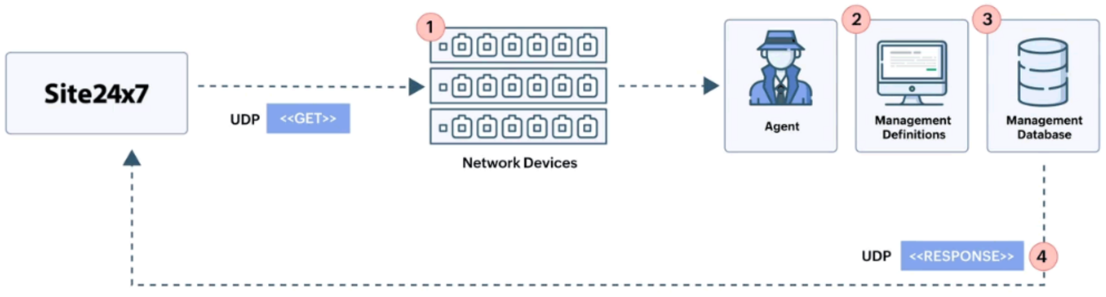


Figure 2.5: GET/GET NEXT/GET BULK/SET *Image from [3]*

Alert Group

- **Trap** - A Trap message is initiated by an agent and sent to a manager when a significant event occurs without waiting for status requests from the manager itself. For example, when a port goes down, unsuccessful user login attempts, or device restarts, an agent sends a trap to notify the manager. Not every event triggers a trap, and not every agent sends traps for the same event. Whether traps are sent for specific events depends on the device or agent manufacturer's specifications.

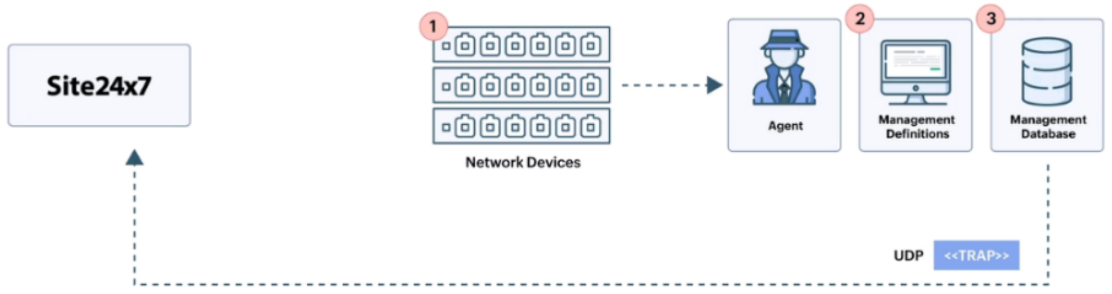


Figure 2.6: Trap message operate *Image from [3]*

Types of Traps Based on that, there are two main types of traps:

– **Generic Traps**: Defined by SNMP standards.

– **Specific Traps**: User-defined by device manufacturers.

Generic Traps (SNMPv1)

Generic traps are standardized in SNMPv1 with seven types:

Trap Type	Description
coldStart (0)	Device is reinitializing, potentially with changed configuration after startup.
warmStart (1)	Device is reinitializing, retaining previous configuration.
linkDown (2)	Error detected in communication link. Includes ifIndex of faulty connection.
linkUp (3)	Communication link restored. Includes ifIndex of restored connection.
authenticationFailure (4)	Unsuccessful authentication attempt across protocols like telnet, ssh, snmp, ftp, etc.
egpNeighborloss (5)	EGP neighbor considered down, peer relationship lost.
enterpriseSpecific (6)	User-defined trap message, not listed under generic types.

Specific Traps Users can define additional specific traps (e.g., boardFailed, configChanged) to suit their needs. These traps are identified by integers in the trap message and require both sender and receiver to support the same MIB for interpretation.

- **Inform**: Similar to a trap, but expects acknowledgment from the manager; will resend if acknowledgment is not received.

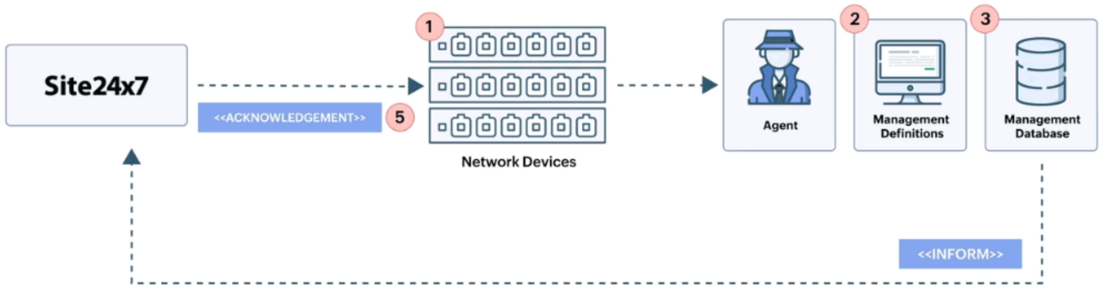


Figure 2.7: Resend the alert message if the manager does not confirm receiving it, rather than sending it and forgetting like Trap message

To summary, GET retrieves the value of a specific object, GET NEXT fetches the next object in sequence, WALK starts from a defined point and traverses all subsequent objects. For this project, we will concentrate solely on GET, GET NEXT, and WALK requests. These operations necessitate specifying:

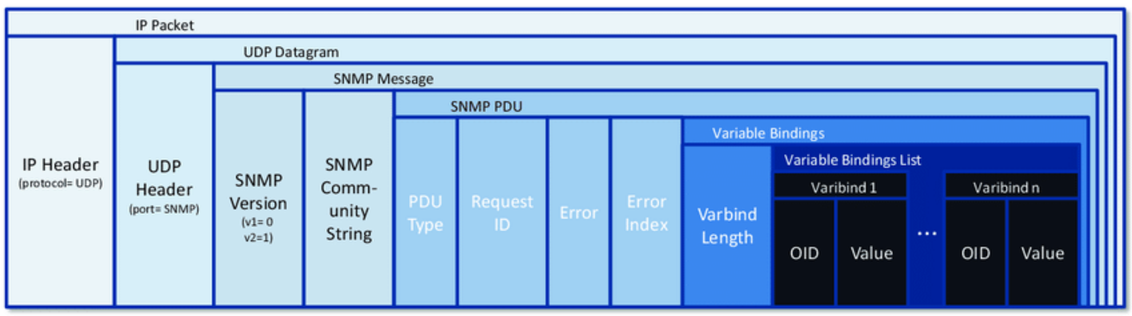
- The target IP address (to identify the device we want to manage)
- The community string (used like a password to access device values)
- OID (specifying which information/object we want to retrieve from the device)

The remaining SNMP request types will be addressed in future phases of the project.

Architecture of an SNMP Request:

Refer to Figure 2.3 in the previous section to visualize how SNMP primarily utilizes UDP for transport. (Although TCP is also an option, its reliability features are not typically useful for SNMP communication due to the request/response nature of the protocol)

The structure of an SNMP is designed to facilitate effective communication between an SNMP Manager and an SNMP Agent. Here's a breakdown of the typical structure of an SNMP message:



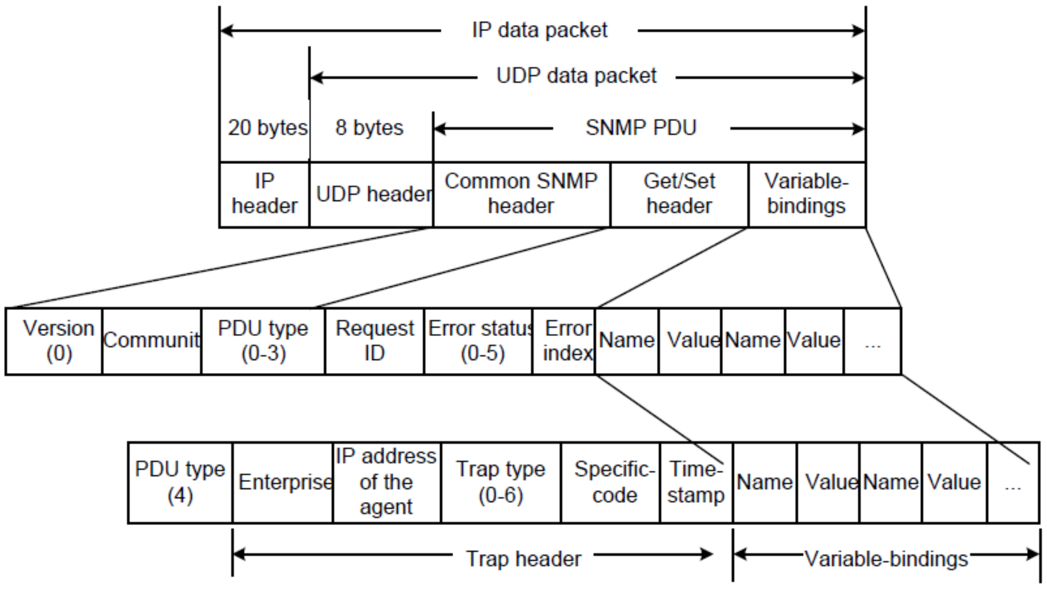


Figure 2.8: SNMP package structure

Ignore metadata from UDP and IP, we focus only on SNMP PDU:

- **Version:** Indicates the version of SNMP being used, such as SNMPv1, SNMPv2c, or SNMPv3
- **Community String:** Acts as a password to authenticate the source or destination of the PDU (mainly in SNMPv1 and SNMPv2c).
- **PDU Type**: Defines the operation to be performed, such as GET, GET-NEXT, SET, TRAP, etc.
- **Request ID**: A unique identifier for each request to match requests with responses.
- **Error Status**: An Integer set to 0x00 in the request sent by the SNMP manager. The SNMP agent places an error code in this field in the response message if an error occurred processing the request. Some error codes include:
  - 0x00 – No error occurred
  - 0x01 – Response message too large to transport
  - 0x02 – The name of the requested object was not found
  - 0x03 – A data type in the request did not match the data type in the SNMP agent
  - 0x04 – The SNMP manager attempted to set a read-only parameter
  - 0x05 – General Error (some error other than the ones listed above)

- **Error Index**: If an Error occurs, the Error Index holds a pointer to the object that caused the error, otherwise the Error Index is 0x00.
- **Variable Bindings**: A list of pairs of object identifiers (OIDs) and their values. This is where the actual data for the request or response is contained. With:
  - Get Request PDU – Value is a Null that acts as a placeholder for the return data. (from NMS to Agent)
  - Get Response PDU – The returned Value from the specified OID of the SNMP agent (agent back to NMS)

Each of these components plays a crucial role in the management and monitoring of network devices via SNMP. The PDU contains all the necessary information needed for the SNMP entities to understand each other's requests and responses. It's like a well-organized envelope containing not just the message, but also metadata that aids in the sorting and delivery of the communication.

More detail about the SNMP message architecture can be found here

2.3 Management Information Base (MIB)

The Management Information Base (MIB) is the cornerstone of this project. Our application relies on reading MIB files to:

1. **Identify and interpret:** Understand the structure and meaning of data objects within managed devices. Then we extract this information about these objects in a user-friendly format.
2. **Translate:** Convert raw SNMP responses into human-readable representations.

MIB Overview

A MIB is a structured collection of managed objects used to manage devices on TCP/IP networks. It serves as a common framework for network management protocols, including SNMP.

MIBs are represented as plain text files (with `.mib` or `.my` extensions) and can be visualized as a hierarchical tree structure. They come in two primary types:

- **Standardized MIBs**, defined by organizations like the CCITT (Consultative Committee for International Telephone and Telegraph), ISO (International Organization for Standardization). Common examples include RFC1213, IF-MIB.

- **Vendor-specific MIBs**, created by device manufacturers to manage their unique features. For instance, Cisco provides MIBs like CISCO-CVP-MIB, CISCO-ENVMON-MIB.

Each object in a MIB file follows a standardized structure, whether it's a standard or vendor-specific MIB. Network Management Systems (NMS) use MIBs to interpret data from devices, which allows for effective monitoring, control, and troubleshooting of network components. In the below section, we will discover more about this "standardized structure".

SMIv1

MIB files are written in SMI (Structure of Management Information), which is a subset of ASN.1 (Abstract Syntax Notation One) and is specified in RFC1155, known as SMIv1. The structure of MIB was later expanded to SMI version 2.[13]

Before we examine the structure of MIB, let's briefly review ASN.1:

- ASN.1 is a standard that outlines encoding rules for digital communication systems. One of the three encoding systems in ASN.1 is BER (Basic Encoding Rules), which SNMP uses for data encoding. Therefore, SNMP-related RFCs often include the note “use of the basic encoding rules of ASN.1”.
- BER specifies various data types such as BOOLEAN, INTEGER, ENUMERATED, OCTET STRING, CHOICE, OBJECT IDENTIFIER, NULL, SEQUENCE, etc. We will delve into some of the most common types in the following section.

Returning to RFC1155, each object consists of three parts: Name, Syntax, and Encoding.

- **Name:** The Name is the identifier of the object, which is of the type OBJECT IDENTIFIER. An OBJECT IDENTIFIER is a sequence of integers that represent the nodes of a tree from root to leaf. It may look something likes this: 1.3.6.1.2.1.0. These sequences are not arbitrary, but rather form by travel across a path in a tree structure.
- **Syntax:** Syntax describes the type of the object. It is derived from ASN.1 standards, but not all types are supported. SMIv1 only supports five primitive types from ASN.1 and six additional defined types. Primitive types include: INTEGER, OCTET-STRING, OBJECT-IDENTIFIER, NULL, SEQUENCE and six additional defined types: NetworkAddress, IpAddress, Counter, Gauge, TimeTicks, Opaque. We will examine more detail about these type when we look at an Object represent in Mib.

- **Encoding:** The Encoding mechanism, as mentioned, follows the BER standard within ASN.1.

OBJECT-TYPE Structure

RFC1155 defines the structure of a record for a “managed object definition,” known as OBJECT-TYPE. ***Other MIB documents must adhere to the SMI specifications when defining a managed object***. An OBJECT-TYPE managed object definition includes fields such as:

- **SYNTAX:** The type of the object, which is one of the primitive, additional defined in SMIv1, SMIv2 or even self defined by vendors.
- **ACCESS:** The access level of the object, which can be read-only, read-write, write-only, or not-accessible. This indicates the action NMS can perform on that object of the device.
- **STATUS:** Indicates whether the object is mandatory (must be supported), optional (may or may not be supported), or obsolete (replaced). An agent that supports a particular MIB standard must support all objects with status=mandatory, while status=optional objects may or may not be supported.
- **DESCRIPTION:** A line explaining the meaning of the object.

For example, the definition of the object **sysDescr** in RFC1213 is as follows:

```
sysDescr OBJECT-TYPE
          SYNTAX  DisplayString (SIZE (0..255))
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
              "A textual description of the entity.
              This value should include the full name
              and version identification of the system's
              hardware type, software operating-system,
              and networking software.  It is mandatory
              that this only contain printable ASCII
              characters."
          ::= { system 1 }
```

As previously noted, in this project, we exclusively utilize the **JSON format**. This is because working directly with the ASN.1 format of MIB can be too complex that overwhelming our ability. Let’s examine the equivalent object in JSON file format, which includes some additional fields when converted by pysmi:

```
"sysDescr": {
    "name": "sysDescr",
    "oid": "1.3.6.1.2.1.1.1",
    "nodetype": "scalar",
    "class": "objecttype",
    "syntax": {
      "type": "DisplayString",
      "class": "type",
      "constraints": {
        "size": [
          {
            "min": 0,
            "max": 255
          }
        ]
      }
    },
    "maxaccess": "read-only",
    "status": "mandatory",
    "description": "A textual description of the entity.
    This value should include the full name and version
    identification of the system’s hardware type, software
    operating-system, and networking software. It is
    mandatory that this only contain printable ASCII characters.
```

Here a some fields and their purposes in an object:

- **sysDescr (Outer Key)**: This field serves as the unique identifier for the Management Information Base (MIB) object within the JSON structure. It’s commonly referred to as the ‘Outer Key’ because it distinguishes each MIB object.
- **name**: This is the common name of the MIB object, typically matching the ‘Outer Key’. It acts as a human-friendly label that makes it easier to reference the MIB object.
- **oid**: Standing for Object Identifier, the OID is a globally unique identifier string that precisely identifies the MIB object across all systems.
- **nodetype**: This attribute indicates whether the node is a ‘scalar’ or something else, like ‘table‘. Scalar nodes correspond to individual data points,

whereas table nodes are arrays of related data points. The nodetype is essential for determining the actual OID needed to access the object on a device. To reach the specific instance of a scalar object, you append a 'scalar instance index' (also known as a 'sub-id') to the OID, often the '.0' to the end of the original OID [5]. For example, if the device name is represented by the MIB object `sysName` with an OID of `1.3.6.1.2.1.1.5` and it is a scalar object, then to retrieve of this field, we will need to use the OID as: `1.3.6.1.2.1.1.5.0`. In case, there are two names, they would be accessed as `sysName.0` and `sysName.1`, with respective OIDs of `1.3.6.1.2.1.1.5.0` and `1.3.6.1.2.1.1.5.1`. This rule is crucial when developing SNMP manager software, as it allows for precise targeting and manipulation of MIB objects.

- **class:** This field categorizes the MIB node. Below are some typical classifications you may come across:
  - **objecttype**: Identifies the node as an object type, a fundamental component in the MIB hierarchy. Object types are variables accessible through SNMP, allowing for read or write operations.
  - **type**: Denotes a type definition, which clarifies the data type for a MIB variable. This definition ensures that the data conforms to expected formats. *Example*: Data types like `Integer32`, `OCTET STRING`, or `OBJECT IDENTIFIER`.
  - **objectgroup**: Signifies a collection of associated objects, which streamlines the MIB's organization and enhances manageability. *Example*: A set of scalar objects or table elements that are logically grouped.
  - **notificationgroup**: Represents a set of related notifications, such as traps or informs. Grouping notifications can simplify administration and response strategies. *Example*: A batch of notifications assembled for more efficient oversight.
  - **compliance**: Specifies the compliance requirements, detailing the necessary MIB components for adherence to certain standards. *Example*: A list of essential objects that must be supported for standard compliance.
- **type**: This specifies the data type of the object. The various data types include:
  - **Integer**: Represents an integer value within the range of -2e31 to 2e31 - 1. It supports enumerations, which are constraints that map a set of related

constants to integer values.

- **Integer32**: Similar to Integer, but it does not support enumeration constraints.
- **Counter**: A non-negative integer that monotonically increases until it reaches a maximum value, after which it rolls over to zero. It is typically used to measure the quantity of items transmitted over time, such as the number of packets per second.
- **Gauge**: A non-negative integer that can increase or decrease, but retains the last reported value if it exceeds its defined range. It is commonly used to measure immediate values, like temperature.
- **IpAddress**: A 32-bit internet address (IPv4), consisting of four consecutive octets.
- **OBJECT IDENTIFIER**: Uniquely identifies managed objects, such as devices, interfaces, or processes, within a hierarchical structure.
- **Octet String**: A sequence of bytes that can contain either binary or textual data. This data type includes subtypes (textual conventions) such as DisplayString, DateAndTime, PhysAddress, MacAddress, SnmpAdminString, ... as suggested in this link
- **TimeTicks**: Represents a time interval in hundredths of seconds.
- **Opaque**: This data type allows the transmission of arbitrary values that are encapsulated as an OCTET STRING according to the ASN.1 standard. It is often used for vendor-specific information or complex data types that are not adequately represented by standard SNMP data types. However, it is important to note that the `Opaque` data type has been deprecated since SNMPv2c, and its use is generally discouraged in favor of more specific and well-defined textual conventions and data types.

• **constraint**: define the permissible range or set of values that an object can take. Constraints are essential for ensuring that the data adheres to expected formats and behaviors. Some common constraints are: enumeration, size, range, ... Here is an example about enumeration constraints:

```
"ifAdminStatus": {
"name": "ifAdminStatus",
"oid": "1.3.6.1.2.1.2.2.1.7",
"nodetype": "column",
    "class": "objecttype",
    "syntax": {
      "type": "INTEGER",
      "class": "type",
      "constraints": {
        "enumeration": {
          "up": 1,
          "down": 2,
          "testing": 3
        }
      }
    }
```

Constraint, together with Type are two key points to format the raw response from SNMP operations to human-readable format.

- **maxaccess:** specifies the maximum access level for the MIB object. Common values are read-only, read-write, or no-access. It indicates whether the value can be read and/or modified.
- **status:** indicates its implementation status. Common values are current, deprecated, or mandatory. This helps in understanding whether the object is actively used or is being phased out
- **description:** A textual description of the MIB object. It provides detailed information about the object, including its purpose and any important notes. This field helps administrators understand what the MIB object represents and how it should be used.

So, RFC1155, also known as SMIv1, establishes the structural framework for object representation within MIB files. It does not directly implement the objects; instead, it sets the groundwork for how they should be represented. The actual object implementations are found in supplementary MIB files, such as RFC1213 and HOST-RESOURCE-MIB. A single MIB file is not required to contain the entire tree structure, but may describe only a specific subtree. Any such subtree, along with all its leaves, can be considered a MIB. When compiled, these individual MIB files collectively create the full MIB tree structure, providing a detailed map of all the objects and their relationships.

MIB Tree Structure

The MIB is organized in a hierarchical tree format. Each level is assigned a number, which forms the numerical sequence found in an OID. Due to the extensive nature of the MIB tree, it's impractical to depict the entire structure in a single image. Therefore, this report only illustrates select paths of the tree. A more comprehensive representation of the tree structure can be found is the `MIBTreeStructure.drawio` file attached inside Docs directory

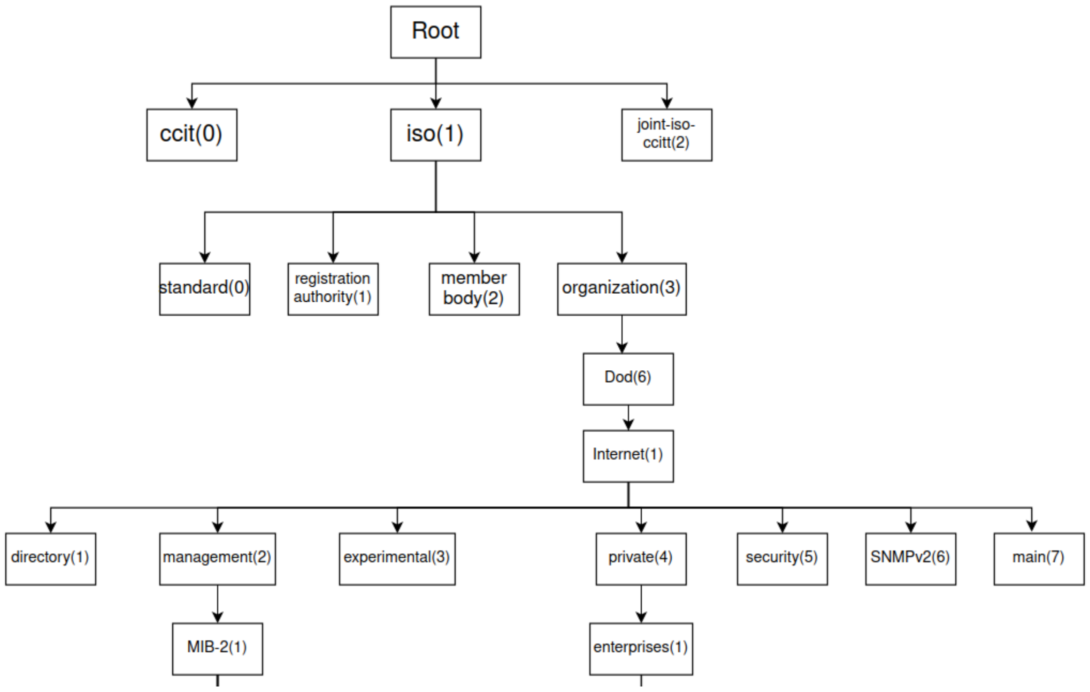


Figure 2.9: MIB tree structure from root node

The root node of the MIB tree does not have a name. Under the root, there are 3 subtrees:

- **ccitt(0)**: managed by CCITT (Consultative Committee for International Telephone and Telegraph).
- **iso(1)**: managed by the ISO organization (International Organization for Standardization).
- **iso-ccitt(2)**: managed by both ISO and CCITT.

Deeper into the **iso** branch:

Under the iso(1) branch, the ISO organization designed a branch for other organizations called org(3). Under org(3), there are many sub-branches, one of which is reserved for the US Department of Defense, dod(6). Under dod(6) is a branch for today's internet community, called internet(1). Everything related to the Internet community is under .iso.org.dod.internet, and all objects of TCP/IP devices start with the prefix .1.3.6.1 (the first dot represents that .iso is a subtree of the root, which is unnamed). With the goal of managing protocol groups in the TCP/IP model and the Internet network, the Internet branch is divided into 4 major groups: Directory, Management, Experimental, and Private:

- **Directory** group: Supports directories in OSI X.500
- **Management group**: Includes objects of the Internet also known as the MIB-II tree (RFC1213)
- **Experimental** group: Used for the testing process before moving to the management group.
- **Private group**: Includes specifications of vendors defined for their own devices and value-increasing regions.

Deeper into the **Management** branch:

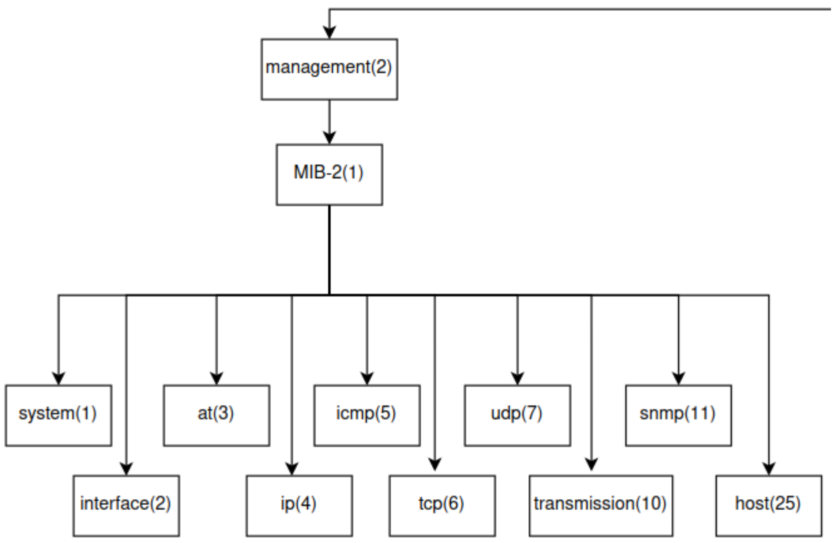


Figure 2.10: Management branch in MIB tree

MIB-II organizes management variables into many functional subtrees, as shown about. These subtrees are further divided into lower-level subtrees such as system objects and corresponding tables for leaves. Leaves mark managed variables of a certain type. Some leaves, like sysDesc, record a single value for a managed variable and require only one storage element (this is called a **"scalar"** object, which we will discuss more in the next chapter). Others, like tcpConnState, may need to record connections at various times. These different times are organized into columns, forming a table, where each row represents the parameters of an entity at different times.

Specific details about some common group are as follows:[11]

- **System group (1.3.6.1.2.1.1)**: Describes the managed system in ASCII text, including object identification, uptime since the last network management entity reboot, and other management details.
- **Interface group (1.3.6.1.2.1.2)**: Consists of 23 object identifiers providing information on performance, configuration, and status for all interface types.
- **Address translation group (1.3.6.1.2.1.3)**: Provides a table for translating between IP addresses and physical (hardware) addresses.
- **IP protocol group (1.3.6.1.2.1.4)**: Contains managed objects for the IP subsystem of a node.
- **ICMP group (1.3.6.1.2.1.5)**: A mandatory component of IP defined in RFC 792, providing internet control message operations within the managed entity.
- **TCP group (1.3.6.1.2.1.6)**: Mandatory, providing information related to TCP operations and connections.
- **UDP group (1.3.6.1.2.1.7)**: Mandatory, providing information related to UDP operations.
- **EGP group (1.3.6.1.2.1.8)**: Mandatory for systems implementing EGP, which conveys information between autonomous systems, detailed in RFC904.
- **CMOT group (1.3.6.1.2.1.9)**: Currently, RFC 1214 is classified as an "historical" protocol.
- **Transmission group (1.3.6.1.2.1.10)**: Contains objects related to data transmission. RFC 1213 does not clearly define these objects.
- **SNMP group(1.3.6.1.2.1.11**: Provides information about SNMP objects.
- **Host group (1.3.6.1.2.1.25**: Defined in HOST-RESOURCES-MIB, provides a standardized set of objects for managing the hardware and software resources on host computers, applicable across various computer architectures. It includes essential system information, performance data, and operational status for effective network management of host resources.

Deeper in to the **Private** branch:

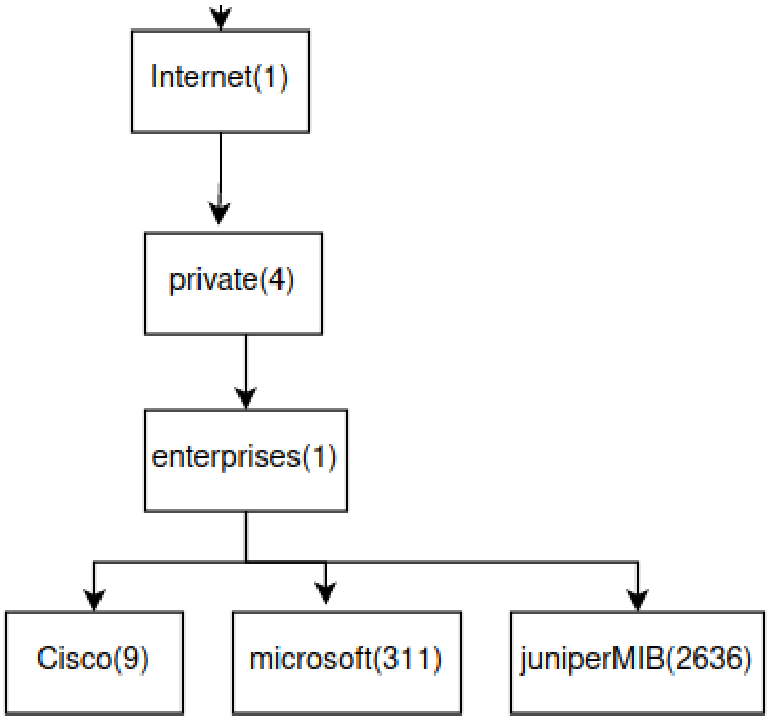
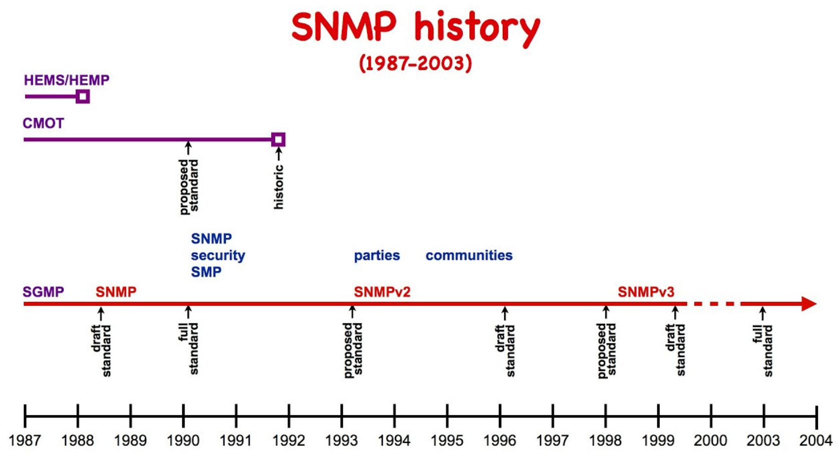


Figure 2.11: Private branch in MIB tree

The OIDs designed by individual companies fall under iso.org.dod.internet.private.enterprise. For instance, Cisco's OID is 1.3.6.1.4.1.9, and Microsoft's is 1.3.6.1.4.1.311, where the numbers 9 and 311 are unique identifiers assigned by IANA to these companies. If Cisco or Microsoft releases a new device, it must first support predefined standard MIBs (like mib-2), then may be its own custom-designed MIBs. Any company-specific MIBs must be located under that company's OID, here is a list of company with their corresponding enterprises OID: http://oidref.com/1.3.6.1.4.1.1

2.4 SNMP Version

For the scope of this project, we opted to deploy our application based on **SNMPv2** due to its simplicity and broad compatibility with a wide range of devices.



There are three main SNMP version **SNMPv1**, **SNMPv2c**, **SNMPv3**:[7] [1]

- **SNMPv1:** This is the original version of SNMP, which has significant security and performance limitations. SNMPv1 uses clear-text passwords (community strings) for authentication, making the managed devices vulnerable to unauthorized access, especially if IP access control lists (ACLs) are not properly configured. The data exchanged between the Network Management System (NMS) and the agents is not encrypted.
- **SNMPv2c:** This is an improved version of SNMPv1. SNMPv2c introduces 64-bit counters, which can better handle high-speed interfaces (since 32-bit counters do not provide enough capacity and must wrap quickly), and the GetBulk operation, which allows the NMS to retrieve large amounts of data more efficiently than using GetNext. It also adds a new type of SNMP communication called the Inform request, which guarantees the NMS can receive the alert message. However, like SNMPv1, SNMPv2c still uses plain text community string for authentication and also no encryption when sending message. *When users encounter SNMP v2, it is usually referring to "SNMP v2c", which the letter ‘c‘ stand for community. There are two other SNMP v2 versions - v2p and v2u - but they are rarely deployed. SNMPv2u is a user-based security model defied in RFC1909 and RFC1910. SNMP 2u offers per-user authentication, similar to SNMPv3, but it never really took off in the wild. Anyone who wants per-user authentication would rather use SNMPv3 instead.*
- **SNMPv3:** SNMPv3 is the latest version of SNMP that addresses the security limitations of earlier versions. It provides message integrity, authentication, and encryption through the use of three key components:
  - SNMP View: Defines what a specific SNMPv3 user can access. For example, a user can be restricted to only view the interface index (OID

1.3.6.1.2.1.2) and anything below it in the MIB tree structure.

- SNMP Group: Associates the SNMP View with a type of access (read-only or read-write) and the security level:
  * noauth: No authentication or encryption
  * auth: Authentication only, no encryption
  * priv: Authentication and encryption
- SNMP User: Added to the group with the specified security level. The security model (e.g., priv) must match the group settings, and includes details like the hash algorithm (e.g., SHA), password, encryption algorithm (e.g., AES), and a shared secret for generating encryption keys.

To summary:

Feature	SNMPv1	SNMPv2c	SNMPv3
Get, GetNext, Set, Walk, Trap	Yes	Yes	Yes
GetBulk, Inform	No	Yes	Yes
Authentication	Clear-text password (community string)	Clear-text password (community string)	Username and Password
Encryption	None	None	AES, DES
Performance	32-bit counters	64-bit counters, GetBulk operation	64-bit counters, GetBulk operation
Access Control	-	-	Views-based

Additionally, SNMPv2 and SNMPv3 can be co-existed within a same network. In standard network management practices, the management system may interact with SNMP agents that operate on various versions. An agent that is multilingual and supports SNMP versions 1, 2, and 3 can function alongside agents that are limited to a single version, as outlined in RFC 25.

Typically, this simultaneous operation is utilized during the transition phase from SNMPv2 to SNMPv3. After completing the migration, it's advisable to deactivate the previous version(s) to maintain network security and integrity.

Some definition about some terminology we used:

- **ACL**: SNMP Access Control List (ACL) is a list of IP addresses that are allowed to manage or monitor an SNMP agent. If a manager with an IP not listed in the ACL sends a request, the agent will not process it, even if the request has the correct community string. This prevents completely block unauthorized SNMP managers. ACLs are implemented on the agent and are supported by most SNMP-compatible devices.

- **SNMPv3 View:** A manager with a read-community can read all OIDs of an agent. However, an agent can restrict this to only certain related OIDs, or a part of the MIB, known as a view. An agent can define multiple views, such as interfaceView for interface-related OIDs, storageView for storage-related OIDs, or AllView for all OIDs. A view is associated with a community string. Depending on the received community string, the agent processes the corresponding view. Many systems do not support the view feature.
- **Community String**: A community string is a character sequence set identically on both the SNMP manager and SNMP agent, acting as a "password" for data exchange between the two. There are three types of community strings:
  - **Read-Community**: When the manager sends a request to the agent, the message contains the Read-Community. The agent compares the Read-Community sent by the manager with its own setting. If they match, the agent responds; if not, it does not.
  - **Write-Community**: Used only in Set Request messages. The agent accepts data changes only if the Write-Community matches on both sides.
  - **Trap-Community**: Included in trap messages from the trap sender to the trap receiver. The trap receiver only stores the trap message if the Trap-Community matches on both sides, although some are configured to accept all trap messages regardless of the Trap-Community.

  While there are three types of community strings, multiple different strings can exist within the same category. This means an agent can declare multiple Read-Communities and Write-Communities.

  On most systems, the default Read-Community is "public," the default Write-Community is "private," and the default Trap-Community is also "public."

Currently landscape:

While SNMPv2c is still widely used due to compatibility with older devices, SNMPv3 is strongly recommended for new implementations due to its superior security. Many organizations are transitioning to SNMPv3 to safeguard their network management data.

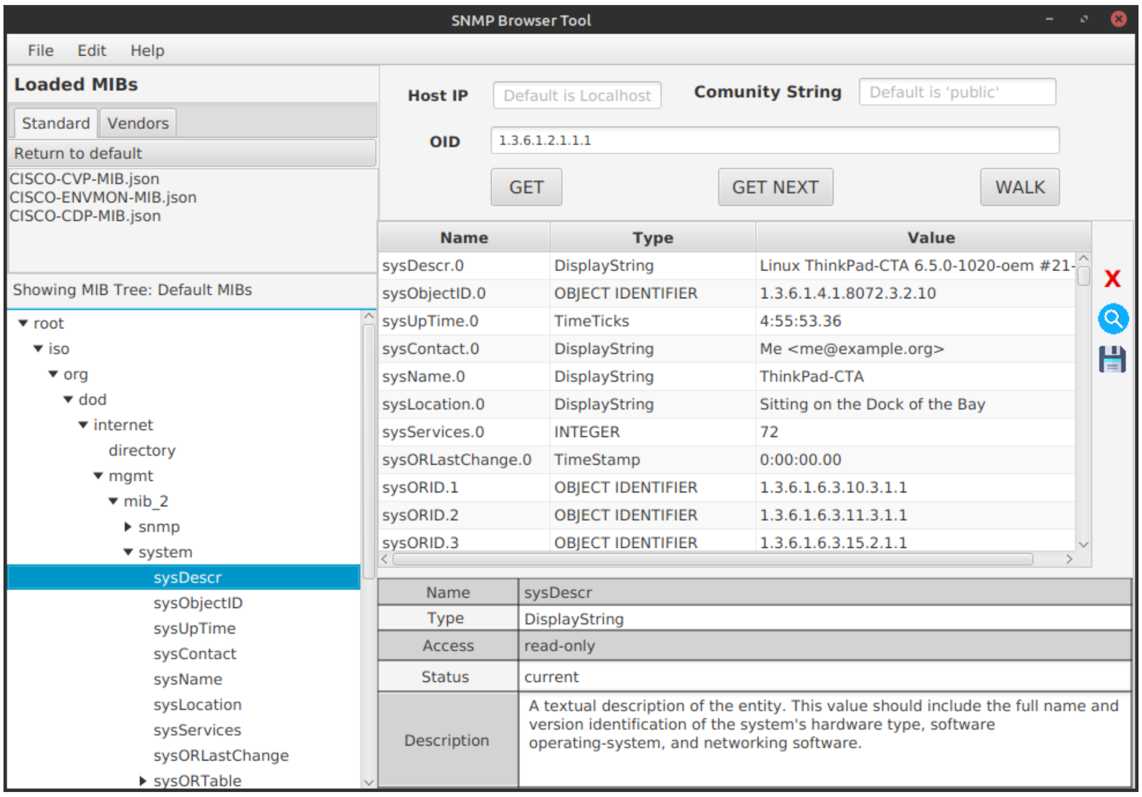
Industry experts and security guidelines emphasize the importance of migrating to SNMPv3 to address the inherent vulnerabilities in older SNMP versions. The performance enhancements in SNMPv3 also make it more suitable for modern network environments.[10]

Although the transition to SNMPv3 may require additional effort, the improved security and performance benefits make it a crucial step for organizations looking to secure their network infrastructure and management data. As the industry continues to emphasize strong security practices, the adoption of SNMPv3 is expected to increase.

3.1 Overview

We have developed an SNMP browser application using Java and the JavaFX framework for the user interface. Our app currently supports SNMP Get, GetNext, and Walk operations using the snmp4j library, working on SNMPv2c. It reads MIB files in JSON format to display the MIB tree structure. Additionally, these files help extract relevant information from provided OIDs to format raw SNMP responses. The primary goal of this app is to demonstrate how SNMP can be used to collect data from server and network devices, effectively managing them. The app also supports Dark Mode that is made through a CSS file. Right now, this feature is still being worked on, and the CSS file will be tweaked further to make improve the app appearance.

Here's a first look at what the app looks like:



CISCO-CVP-MIB.json
CISCO-ENVMON-MIB.json
CISCO-CDP-MIB.json

Showing MIB Tree: Default MIBs

```
▾ root
  ▾ iso
    ▾ org
      ▾ dod
        ▾ internet
            directory
          ▾ mgmt
            ▾ mib_2
              ▸ snmp
              ▾ system
                  sysDescr
                  sysObjectID
                  sysUpTime
                  sysContact
                  sysName
                  sysLocation
                  sysServices
                  sysORLastChange
                ▸ sysORTable
```

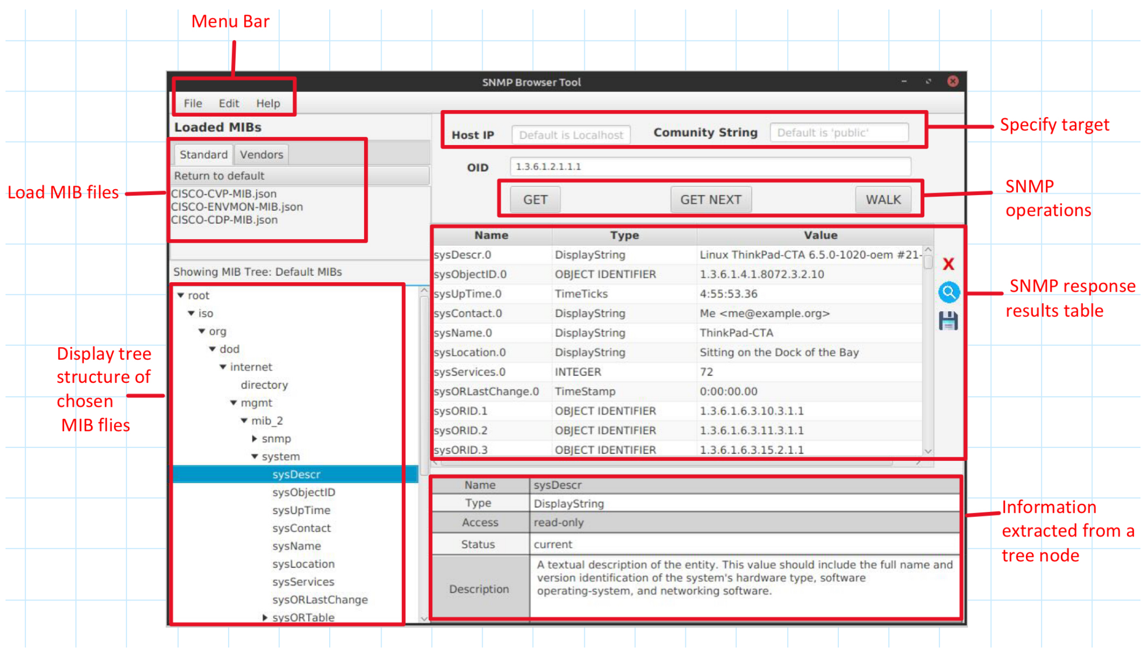
OID: 1.3.6.1.2.1.1.1

Name	Type	Value
sysDescr.0	DisplayString	Linux ThinkPad-CTA 6.5.0-1020-oem #21-
sysObjectID.0	OBJECT IDENTIFIER	1.3.6.1.4.1.8072.3.2.10
sysUpTime.0	TimeTicks	4:55:53.36
sysContact.0	DisplayString	Me <me@example.org>
sysName.0	DisplayString	ThinkPad-CTA
sysLocation.0	DisplayString	Sitting on the Dock of the Bay
sysServices.0	INTEGER	72
sysORLastChange.0	TimeStamp	0:00:00.00
sysORID.1	OBJECT IDENTIFIER	1.3.6.1.6.3.10.3.1.1
sysORID.2	OBJECT IDENTIFIER	1.3.6.1.6.3.11.3.1.1
sysORID.3	OBJECT IDENTIFIER	1.3.6.1.6.3.15.2.1.1

Name	sysDescr
Type	DisplayString
Access	read-only
Status	current
Description	A textual description of the entity. This value should include the full name and version identification of the system's hardware type, software operating-system, and networking software.

Figure 3.1: App Demo

3.2 App Features



CISCO-CVP-MIB.json
CISCO-ENVMON-MIB.json
CISCO-CDP-MIB.json

root
iso
org
dod
internet
directory
mgmt
mib_2
snmp
system
sysDescr
sysObjectID
sysUpTime
sysContact
sysName
sysLocation
sysServices
sysORLastChange
sysORTable

Host IP: Default is Localhost
Comunity String: Default is 'public'
OID: 1.3.6.1.2.1.1.1

Name	Type	Value
sysDescr.0	DisplayString	Linux ThinkPad-CTA 6.5.0-1020-oem #21-
sysObjectID.0	OBJECT IDENTIFIER	1.3.6.1.4.1.8072.3.2.10
sysUpTime.0	TimeTicks	4:55:53.36
sysContact.0	DisplayString	Me <me@example.org>
sysName.0	DisplayString	ThinkPad-CTA
sysLocation.0	DisplayString	Sitting on the Dock of the Bay
sysServices.0	INTEGER	72
sysORLastChange.0	TimeStamp	0:00:00.00
sysORID.1	OBJECT IDENTIFIER	1.3.6.1.6.3.10.3.1.1
sysORID.2	OBJECT IDENTIFIER	1.3.6.1.6.3.11.3.1.1
sysORID.3	OBJECT IDENTIFIER	1.3.6.1.6.3.15.2.1.1

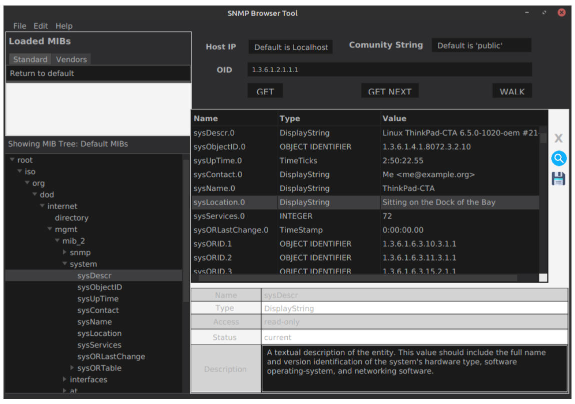
Name	sysDescr
Type	DisplayString
Access	read-only
Status	current
Description	A textual description of the entity. This value should include the full name and version identification of the system's hardware type, software operating-system, and networking software.

Figure 3.2: App UI breaks into parts

Below, we highlight some key features of the application:

- **Menu bar:** there are three items in the menu bar:
  - File: it contains three items:
  - * Open MIB: let the user select a MIB file from their system to load into the program, which will appear in the MIB Load section.
    * Import MIB: Similar to Open MIB, but it also saves a copy of the MIB file to the application's MIB Databases directory.
    * Unload all MIBs: Clears the MIB Load section below.

    In the MIB Database directory, approximately 70 MIBs are currently available in JSON format. These have been sourced from the MIB databases of the iReasoning MIB browser software and subsequently converted to JSON format.
  - Edit: Allows toggling between light mode and dark mode. Note that the dark mode is currently under development as we are still fine-tuning the CSS file.

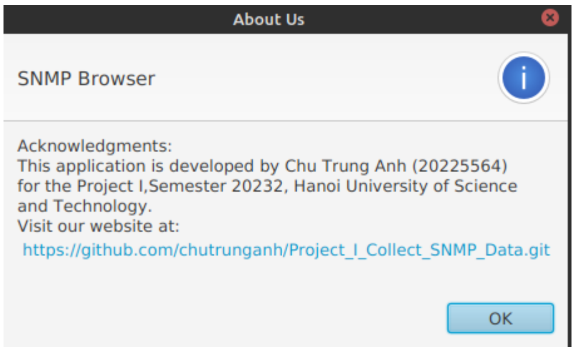


Host IP: Default is Localhost
Comunity String: Default is 'public'
OID: 1.3.6.1.2.1.1.1

Name	Type	Value
sysDescr.0	DisplayString	Linux ThinkPad-CTA 6.5.0-1020-oem #21
sysObjectID.0	OBJECT IDENTIFIER	1.3.6.1.4.1.8072.3.2.10
sysUpTime.0	TimeTicks	2:50:22.55
sysContact.0	DisplayString	Me <me@example.org>
sysName.0	DisplayString	ThinkPad-CTA
sysLocation.0	DisplayString	Sitting on the Dock of the Bay
sysServices.0	INTEGER	72
sysORLastChange.0	TimeStamp	0:00:00.00
sysORID.1	OBJECT IDENTIFIER	1.3.6.1.6.3.10.3.1.1
sysORID.2	OBJECT IDENTIFIER	1.3.6.1.6.3.11.3.1.1
sysORID.3	OBJECT IDENTIFIER	1.3.6.1.6.3.15.2.1.1

Name	sysDescr
Type	DisplayString
Access	read-only
Status	current
Description	A textual description of the entity. This value should include the full name and version identification of the system's hardware type, software operating-system, and networking software.

– Help: Provides information about the app, including contributors and acknowledgments.



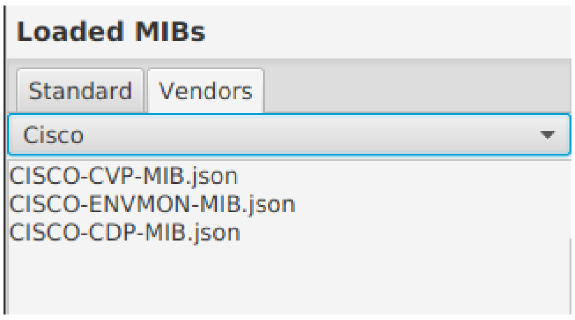
SNMP Browser

Acknowledgments:
This application is developed by Chu Trung Anh (20225564)
for the Project I,Semester 20232, Hanoi University of Science
and Technology.
Visit our website at:
https://github.com/chutrunganh/Project_I_Collect_SNMP_Data.git

- **Load MIBs Panel**: There are two tabs

  – Standard: This option includes a 'Reset to Default' button that restores the MIB tree to its original state, just like when the app is first launched. The default MIB tree is composed of the following files: **SNMPv2-SMI.json**, **RFC1213-MIB.json**, **HOST-RESOURCES-MIB.json**, **SNMPv2-MIB.json**, and **IF-MIB.json**.

  – Vendor: This allows the user to select a vendor from a drop-down menu. Each vendor is associated with a predefined list of available MIBs.



CISCO-CVP-MIB.json
CISCO-ENVMON-MIB.json
CISCO-CDP-MIB.json

All MIBs that are opened, imported, or selected from a vendor will be loaded into the application and displayed in their file name in the below the flow pane section.

- **MIBTreeDisplay pane:** When user click on a file name in the MIB Load pane above, the tree structure of that file will be displayed. This tree is created using the TreeView object in JavaFX. A single click on any node in the tree will reveal that node's details in the bottom right pane of the app. Although each node contains numerous fields, we only show the most important ones in the UI: **Name**, **Type**, **Access**, **Status**, **Description**, and **OID**. Double-clicking on a node will execute an SNMP Get request for that selected node.
- **Host IP, Community String**: Enter the IP address of the target device and the community string to authenticate the communication. The community string is entered into a Password Field in JavaFX. If left blank, the default IP address is set to localhost (127.0.0.1) and the community string defaults to "public". note that currently our application only support SNMP Get, Get Next and Walk operation, so the port used is fixed to UDP 161.
- **Get, Get Next, Walk button**:The functions of these buttons are indicated by their names. They use the current IP address, community string, and OID displayed in the UI to perform their actions.
  - Get: retrieves the value of the passed in OID. Double-click on a tree node can perform similar action.
  - Get Next: retrieve the value of the OID that comes after the one we pass in. If users select a node and then click Get Next several times, it will sequentially retrieve the values of subsequent OIDs (incremental from the first OID). The OID currently being used will only be reset if you click on a different node in the TreeView or if you press the Clear Table button.
  - Walk: retrieve values by exploring the subtree starting from the entered OID. Using Walk on a large and complex MIB tree can be demanding on resources. In our tests, starting from node 1.3.6.1 with around 70 MIB files in the database, the Walk operation took approximately 1.5 seconds to complete. Currently, Walk searches for OIDs in MIBs by comparing the entered OID with the root index of a MIB file, and moves to the next file if there's no match. We plan to implement more efficient techniques in the future to speed up execution.
- **Result Table:** Display the SNMP response, which includes the **name**, **data type**, and **formatted values**. If the information required to format an OID is

unavailable, append "(raw response)" after the value. Double-clicking on a row will highlight it and navigate to the corresponding node in the MibTreeDisplay panel. On the right side, there is a toolbox panel that allows users to perform several actions: clear the results table, prompt for a name to locate a specific row, and export all results in the table to a CSV file. Hovering over any tool will display a tooltip describing its usage.

Here is the summary of all important features in the User case diagram:

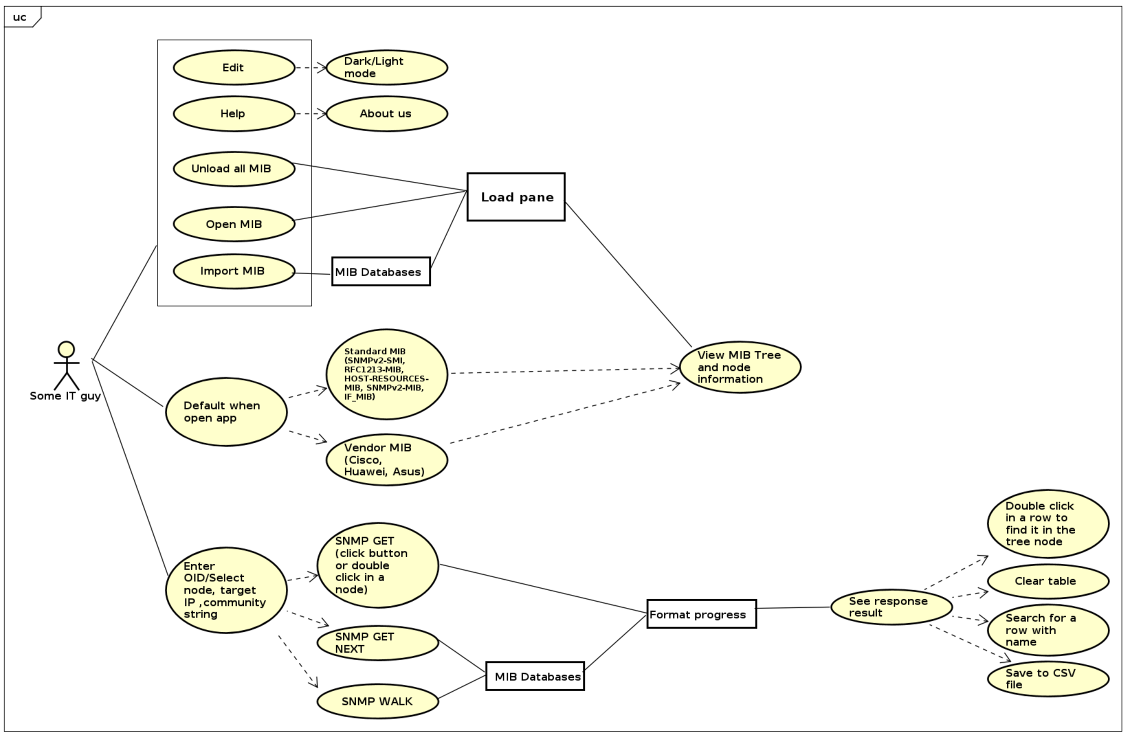
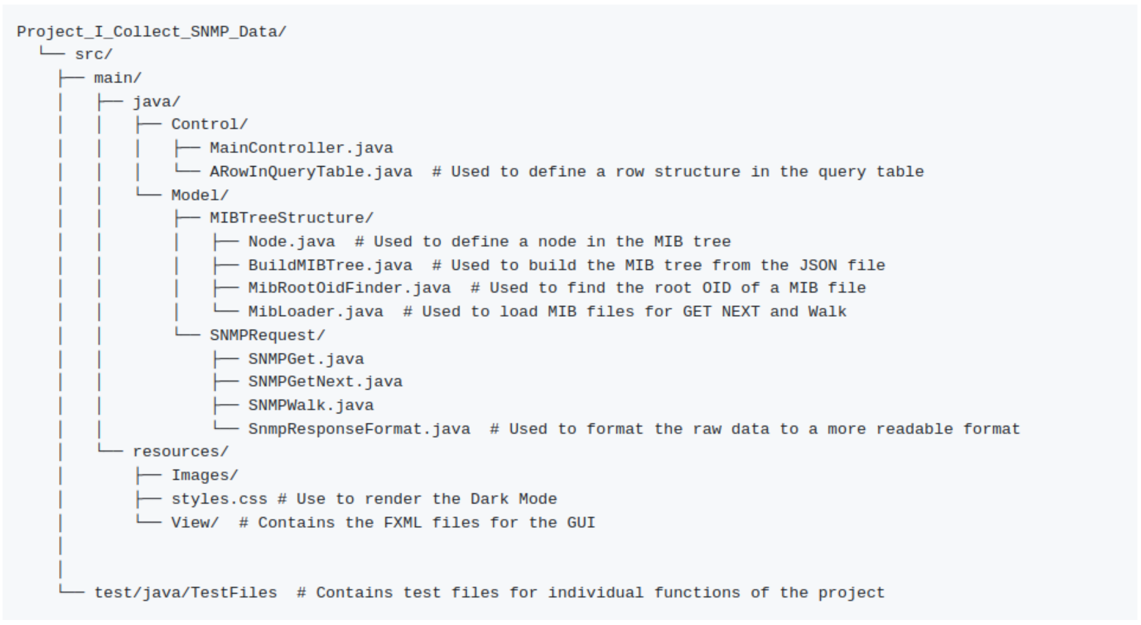

uc
Edit
Dark/Light mode
Help
About us
Unload all MIB
Open MIB
Import MIB
Load pane
MIB Databases
Some IT guy
Default when open app
Standard MIB (SNMPv2-SMI, RFC1213-MIB, HOST-RESOURCES-MIB, SNMPv2-MIB, IF_MIB)
Vendor MIB (Cisco, Huawei, Asus)
View MIB Tree and node information
Enter OID/Select node, target IP ,community string
SNMP GET (click button or double click in a node)
SNMP GET NEXT
SNMP WALK
MIB Databases
Format progress
See response result
Double click in a row to find it in the tree node
Clear table
Search for a row with name
Save to CSV file

Figure 3.3: User Case Diagram

3.3 Code Structure of Application

We designed our code following the Maven standard structure and the Model-View-Controller (MVC) pattern. Here is the main structure and their short purpose:



```
Project_I_Collect_SNMP_Data/
  └── src/
    ├── main/
    │   ├── java/
    │   │   ├── Control/
    │   │   │   ├── MainController.java
    │   │   │   └── ARowInQueryTable.java  # Used to define a row structure in the query table
    │   │   └── Model/
    │   │       ├── MIBTreeStructure/
    │   │       │   ├── Node.java  # Used to define a node in the MIB tree
    │   │       │   ├── BuildMIBTree.java  # Used to build the MIB tree from the JSON file
    │   │       │   ├── MibRootOidFinder.java  # Used to find the root OID of a MIB file
    │   │       │   └── MibLoader.java  # Used to load MIB files for GET NEXT and Walk
    │   │       └── SNMPRequest/
    │   │           ├── SNMPGet.java
    │   │           ├── SNMPGetNext.java
    │   │           ├── SNMPWalk.java
    │   │           └── SnmpResponseFormat.java  # Used to format the raw data to a more readable format
    │   └── resources/
    │       ├── Images/
    │       ├── styles.css # Use to render the Dark Mode
    │       └── View/  # Contains the FXML files for the GUI
    │
    │
    └── test/java/TestFiles  # Contains test files for individual functions of the project
```

Figure 3.4: Code structure

The project also incorporates the following components:

- **MIB Databases** directory: contains all the MIB files that the application utilizes to gather information. There are two instances of this directory in the project: the first is located within **Project_I_Collect_SNMP_Data/out/artifacts/SNMP_Browser/ MIB Databases** and is used to package with the JAR file, while the second is located right under the **Project_I_Collect_SNMP_Data/MIB Databases** root directory and is accessed by the source code when running from the IDE.
- **GetJSONFiles** script: This script is employed to transform MIB ASN1 files into JSON format files. If you wish to recompile the MIB files, adjust the *MIB_DIR* and *DES_DIR* variables in the script to match your case.
- **ProjectCodeStructureDiagram.asta** : Astah UML project contains all Class diagrams and Use Case diagram.

Here is the general class diagram

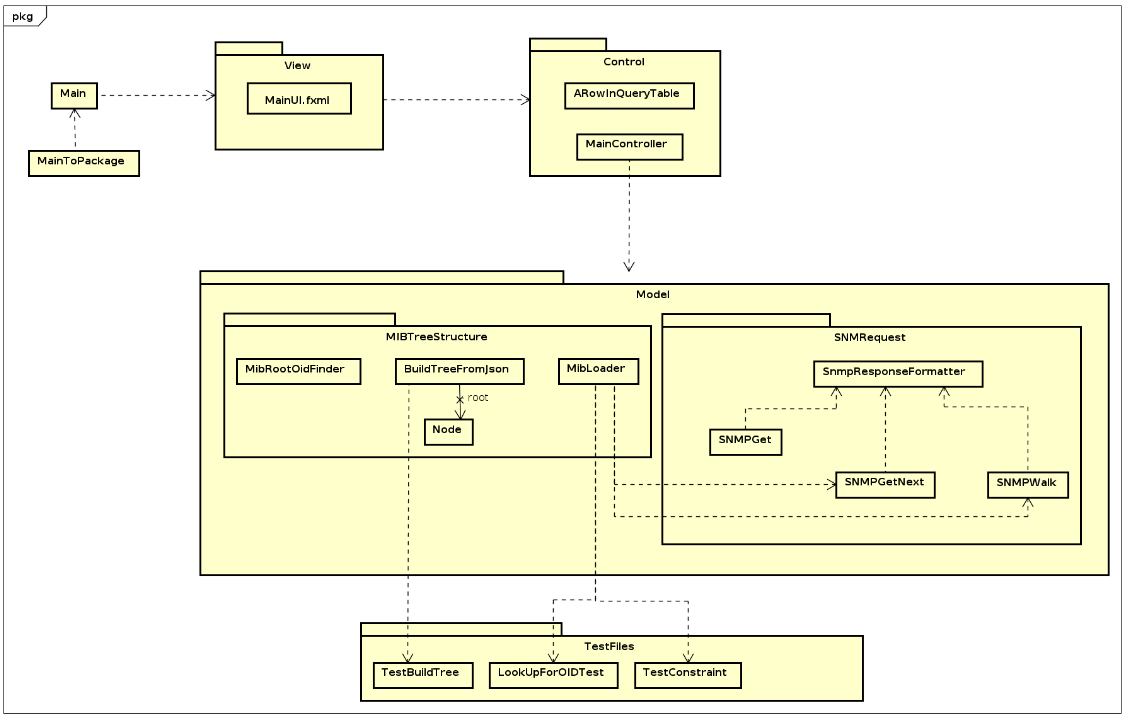


Figure 3.5: General Class Diagram

The `Main` class is responsible for launching the main window (JavaFX Stage) and must therefore extend the `Application` class. However, when packaging the application, it is often recommended the main entry point class of the artifact does not have any external dependencies (like extending another class). Therefore, we use `MainToPackage` class, which just simply call the `main` method from `Main` class, as the main class when making the Artifact.

Two of the most important packages in the project are:

- **Control:** contains two classes: **ARowInQueryTable**, which defines the structure of a row in the display table, and **MainController**, which is responsible for managing all events from **MainUI.fxml**. The MainController class monitors user actions and interacts with the Model. Below is a detailed class diagram of the Control package:

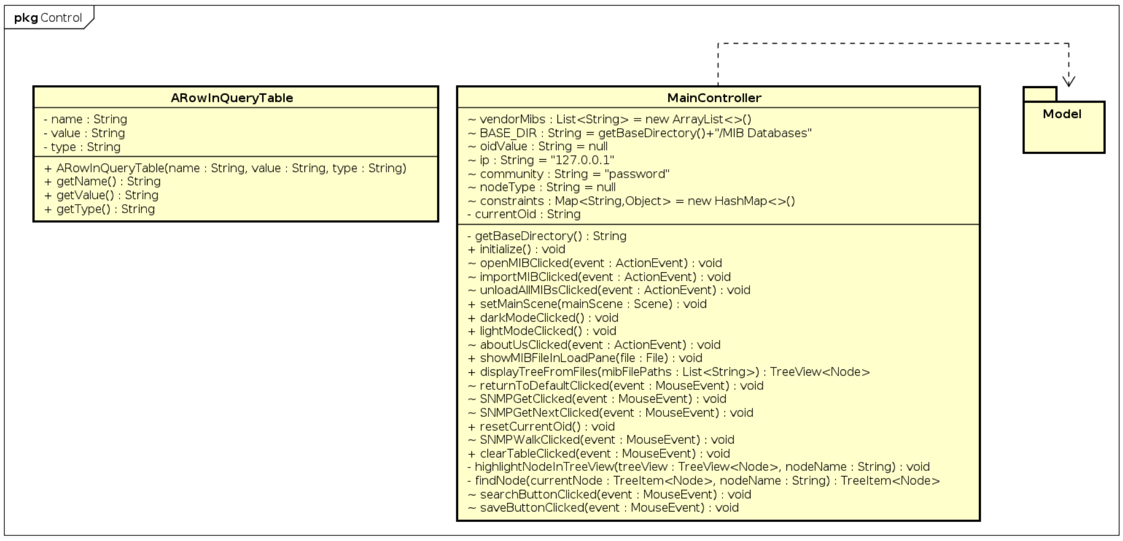

pkg Control
ARowInQueryTable
- name : String
- value : String
- type : String
+ ARowInQueryTable(name : String, value : String, type : String)
+ getName() : String
+ getValue() : String
+ getType() : String
MainController
~ vendorMibs : List<String> = new ArrayList<>()
~ BASE_DIR : String = getBaseDirectory()+"/MIB Databases"
~ oidValue : String = null
~ ip : String = "127.0.0.1"
~ community : String = "password"
~ nodeType : String = null
~ constraints : Map<String,Object> = new HashMap<>()
- currentOid : String
- getBaseDirectory() : String
+ initialize() : void
~ openMIBClicked(event : ActionEvent) : void
~ importMIBClicked(event : ActionEvent) : void
~ unloadAllMIBsClicked(event : ActionEvent) : void
+ setMainScene(mainScene : Scene) : void
+ darkModeClicked() : void
+ lightModeClicked() : void
~ aboutUsClicked(event : ActionEvent) : void
+ showMIBFileInLoadPane(file : File) : void
+ displayTreeFromFiles(mibFilePaths : List<String>) : TreeView<Node>
~ returnToDefaultClicked(event : MouseEvent) : void
~ SNMPGetClicked(event : MouseEvent) : void
~ SNMPGetNextClicked(event : MouseEvent) : void
+ resetCurrentOid() : void
~ SNMPWalkClicked(event : MouseEvent) : void
+ clearTableClicked(event : MouseEvent) : void
- highlightNodeInTreeView(treeView : TreeView<Node>, nodeName : String) : void
- findNode(currentNode : TreeItem<Node>, nodeName : String) : TreeItem<Node>
~ searchButtonClicked(event : MouseEvent) : void
~ saveButtonClicked(event : MouseEvent) : void
Model

Figure 3.6: Control Class Diagram

- **Model:** contains two sub packages **MIBTreeStructure** and **SNMPRequest**:

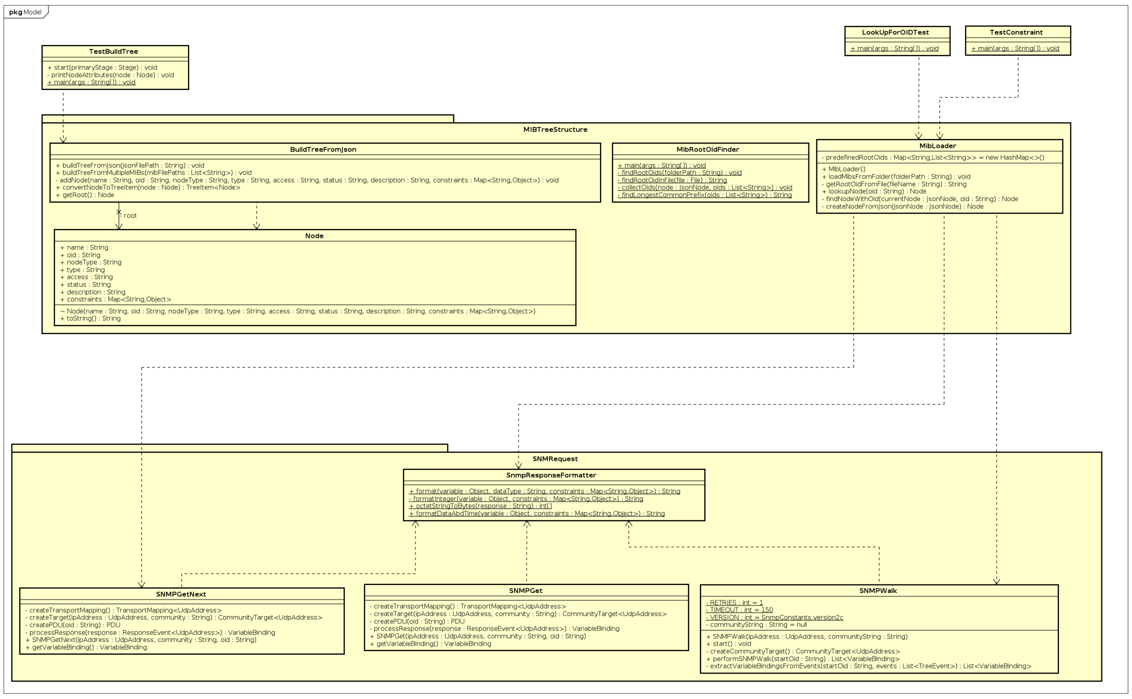
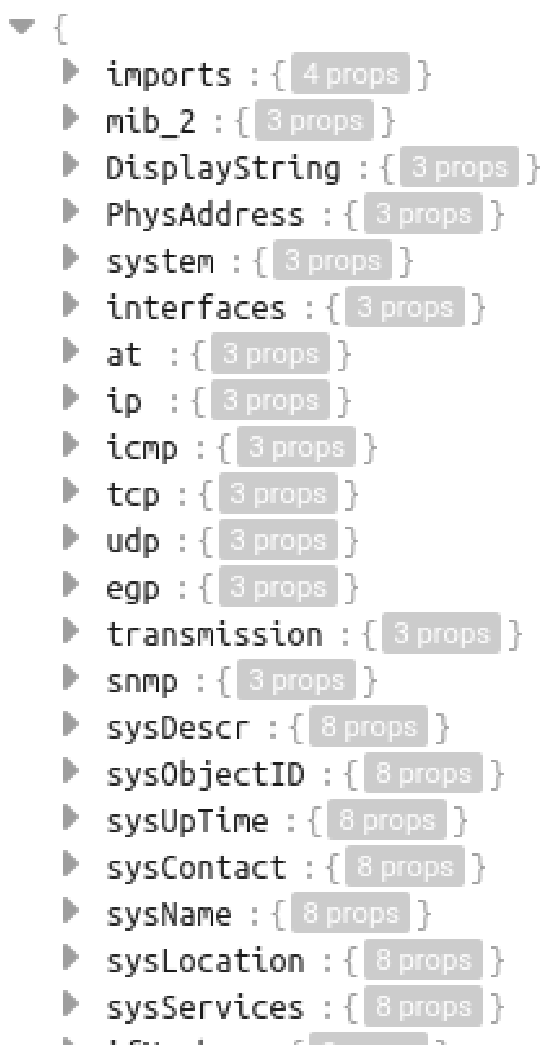

pkg Model
TestBuildTree
+ start(primaryStage : Stage) : void
- printNodeAttributes(node : Node) : void
+ main(args : String[ ]) : void
LookUpForOIDTest
+ main(args : String[ ]) : void
TestConstraint
+ main(args : String[ ]) : void
MIBTreeStructure
BuildTreeFromJson
+ buildTreeFromJson(jsonFilePath : String) : void
+ buildTreeFromMultipleMIBs(mibFilePaths : List<String>) : void
- addNode(name : String, oid : String, nodeType : String, type : String, access : String, status : String, description : String, constraints : Map<String,Object>) : void
+ convertNodeToTreeItem(node : Node) : TreeItem<Node>
+ getRoot() : Node
- root
Node
+ name : String
+ oid : String
+ nodeType : String
+ type : String
+ access : String
+ status : String
+ description : String
+ constraints : Map<String,Object>
~ Node(name : String, oid : String, nodeType : String, type : String, access : String, status : String, description : String, constraints : Map<String,Object>)
+ toString() : String
MibRootOidFinder
+ main(args : String[ ]) : void
- findRootOids(folderPath : String) : void
- findRootOidInFile(file : File) : String
- collectOids(node : JsonNode, oids : List<String>) : void
- findLongestCommonPrefix(oids : List<String>) : String
MibLoader
- predefinedRootOids : Map<String,List<String>> = new HashMap<>()
+ MibLoader()
+ loadMibsFromFolder(folderPath : String) : void
- getRootOidFromFile(fileName : String) : String
+ lookupNode(oid : String) : Node
- findNodeWithOid(currentNode : JsonNode, oid : String) : Node
- createNodeFromJson(jsonNode : JsonNode) : Node
SNMRequest
SnmpResponseFormatter
+ format(variable : Object, dataType : String, constraints : Map<String,Object>) : String
- formatInteger(variable : Object, constraints : Map<String,Object>) : String
+ octetStringToBytes(response : String) : int[ ]
+ formatDataAbdTime(variable : Object, constraints : Map<String,Object>) : String
SNMPGetNext
- createTransportMapping() : TransportMapping<UdpAddress>
- createTarget(ipAddress : UdpAddress, community : String) : CommunityTarget<UdpAddress>
- createPDU(oid : String) : PDU
- processResponse(response : ResponseEvent<UdpAddress>) : VariableBinding
+ SNMPGetNext(ipAddress : UdpAddress, community : String, oid : String)
+ getVariableBinding() : VariableBinding
SNMPGet
- createTransportMapping() : TransportMapping<UdpAddress>
- createTarget(ipAddress : UdpAddress, community : String) : CommunityTarget<UdpAddress>
- createPDU(oid : String) : PDU
- processResponse(response : ResponseEvent<UdpAddress>) : VariableBinding
+ SNMPGet(ipAddress : UdpAddress, community : String, oid : String)
+ getVariableBinding() : VariableBinding
SNMPWalk
- RETRIES : int = 1
- TIMEOUT : int = 150
- VERSION : int = SnmpConstants.version2c
- communityString : String = null
+ SNMPWalk(ipAddress : UdpAddress, communityString : String)
+ start() : void
- createCommunityTarget() : CommunityTarget<UdpAddress>
+ performSNMPWalk(startOid : String) : List<VariableBinding>
- extractVariableBindingsFromEvents(startOid : String, events : List<TreeEvent>) : List<VariableBinding>

Figure 3.7: Model Class Diagram

  – The **MIBTreeStructure** class responsible for all action related to MIB files.

    ∗ Its primary function is to construct the tree structure from the provided MIBs. Initially, the JSON files, converted by pysmi, are flat, as shown below:



```
▼ {
  ▶ imports : { 4 props }
  ▶ mib_2 : { 3 props }
  ▶ DisplayString : { 3 props }
  ▶ PhysAddress : { 3 props }
  ▶ system : { 3 props }
  ▶ interfaces : { 3 props }
  ▶ at : { 3 props }
  ▶ ip : { 3 props }
  ▶ icmp : { 3 props }
  ▶ tcp : { 3 props }
  ▶ udp : { 3 props }
  ▶ egp : { 3 props }
  ▶ transmission : { 3 props }
  ▶ snmp : { 3 props }
  ▶ sysDescr : { 8 props }
  ▶ sysObjectID : { 8 props }
  ▶ sysUpTime : { 8 props }
  ▶ sysContact : { 8 props }
  ▶ sysName : { 8 props }
  ▶ sysLocation : { 8 props }
  ▶ sysServices : { 8 props }
```

Figure 3.8: The sysDescr, sysObjectID are on the same level with the system instead of being sub objects inside it

To establish the tree structure, we utilize the OID path of each node. By parsing the JSON file, we create the tree with an entity stored as a Node, as defined in the Node class. Each Node possesses the following attributes:

```
public String name;
public String oid;
public String nodeType;
public String type;
public String access;
public String status;
public String description;
public Map<String, Object> constraints;
//Constraints,include the name of the constraints
```

```
and the actual constraints, since we do not know in
advance the type of the constraints, we use the Object
type to store the constraints object.

Map<String, Node> children = new HashMap<>();
```

Leveraging the OID path for construction allows this class to merge multiple JSON files into a single, comprehensive MIB tree. To verify the successful build and display of the tree, use the **TestBuildTree** class within the `test` package.

  * The second responsibility is to locate information for a given OID across all MIB files. Upon invocation, it loads MIB files into memory and searches for the node with the matching index, returning it if found. Currently, we compare whether the given OID starts with the same sequence as the root OID of a MIB file, continuing the search if it does, or skipping the file if it does not. Identifying the root OID of each file is solving the Longest Common Sequence problem (we find the longest OID path between all OIDs in a JSON file), which is accomplished using the **MibRootOidFinder** class. This class is called once to retrieve the root OID information and does not run each time the application is executed. All these responsibilities are handled by `MibLoader` class. To check with a given OID, this class can correctly find the node with matching OID, use the **LookUpForOIDTest** in the `test` package. *Note that: We need to use the MibRootOidFinder class to find the root OID of each MIB file, rather than simply taking the first OID in the file as the root. The first OID is not guaranteed to be the root. For a more detailed explanation and counterexamples, please refer to the comments in the MIibRootOidFinder class*

- The **SNMPRequest**: component handles the retrieval and formatting of SNMP data. It consists of the following key elements:
  * SNMPResponseFormatter: Take in the raw response from SNMP operations, together with data type and constraint to convert to human-readable format. Most of SNMP response result have been already in human-readable format already, we just handle some type such as: DateAndTime, Enumeration of Integer, ... To check with a given re-

sponse with enumeration constraint, this class can correctly handle, use **TestConstraint** in `test` package.

* SNMP Get: Upon user selection of a Node, which provides the OID, name, data type, and constraints, an SNMP Get operation is performed using that OID. The raw response, along with the data type and constraints, is then processed by SNMPResponseFormatter to produce the final result for display in the table. For objects with a node type of 'scalar', ".0" is appended to the OID; for other types, ".1", ".2", etc., are tried until a 'noSuchObject' response is encountered. This is a temporary implementation and may require refinement later.
* SNMP Get Next: When a Node is selected by the user, the OID immediately following the selected one is retrieved. As the information about this new OID is unknown, MibLoader is used to look up its details, which are then passed to SNMPResponseFormatter. Note that with the OID we must remove the post fix (".someNumber ") at the end of OID so that it exists in the MIB files.
* SNMP Walk: retrieves a list of OIDs and their corresponding values, starting from the selected OID and continuing all the way down the subtree. Each retrieved OID is stripped of its post-fix, then searched using MibLoader before being processed by SNMPResponseFormatter.

3.4 Installation and Setup

The application has been packaged as a ZIP file containing all necessary JAR file and dependencies. This means it can run on any operating system with a Java Virtual Machine (JVM) installed. **Java version 21 or newer is required; older versions will not be able to launch the application.**

To get started:

1. Visit the Releases page on the project's GitHub repository:

   https://github.com/chutrunganh/Project-I-Collect-SNMP-Data/releases/

2. Download the latest release, a ZIP file named `SNMP_Browser`.

3. Unzip the file. You'll find a JAR file and a "MIB Databases" directory inside.

```
SNMP_Browser/
    SNMP Brower.jar
    MIB Databases
```

4. Double-click the JAR file to launch the application.

In case you want to **build the project from source**:

1. Clone the project with
   ‘ git clone https://github.com/chutrunganh
   /Project-I-Collect-SNMP-Data.git‘.
2. Build the artifact to generate the `SNMP Browser` artifact directory (using IntelliJ IDE), then make a copy of the `MIB Databases` directory into the same location as the JAR file within the `SNMP Browser` artifact. This ensures everything works smoothly, no matter where you’re running it.

Note that: The path to MIB Databases directory is specified in the BASE_DIR variable in MainController.java. If you encounter any issues related to the MIBs’ relative path, please refer to this location.

4.1 Future Improvement

The SNMP application we developed successfully demonstrates the use of SNMP for collecting data from servers and network devices, enabling effective management. However, there is still a lot of work to improvement in terms of user experience, performance, and functionality.

Current Issues and Solution:

- **Deployment:** Although the application is implemented using Java, with the slogan "Write Once Run Anywhere," we provided a JAR file that users can simply double-click to run the app. However, it requires a Java Virtual Machine (JVM) with a compatible Java version to be installed first. Therefore, we plan to deploy the application to platform-specific installers (e.g., `.exe`, `.dmg`, `.deb`) for better optimization and system integration, or provide a Docker image to simplify installation.
- **Apply machine learning algorithms and statistical techniques** to automatically retrieve device data, analyze it, and predict potential failures or misbehavior. .
- **UI/UX:** Some UI elements need refinement, and the Dark Mode implementation can be enhanced. Solution:
  - Address visual inconsistencies and improve the overall look and feel of the application
  - Ensure proper color contrast and readability in Dark Mode.
- **Handle ASN1**: Try to implement the code to handle data directly in ASN1 format, instead of converting it to JSON first. In past attempts, we tried integrating an API called Mibble [4]. This API can process ASN1 format directly and has many built-in methods to handle these files. However, we encountered an issue when trying to get the data type of object. The API did not return the specific textual conventions we expected, such as DisplayString, DateAndTime, or Timeticks. Instead, it returned general data types like OCTET STRING.
- **MIB Database:** The database lacks comprehensive MIB coverage, and the search process could be more efficient. Solution:
  - Add more MIB modules to the database for better device support

  - Explore the possibility of integrating with online MIB repositories or allowing community contributions to expand the database.
  - Implement more efficient search algorithms (e.g., indexing, filtering) to improve the speed and accuracy of MIB lookup.

- **Data Handling:** Additional data types and constraints need to be supported for broader compatibility.
- **Limited Functionality:** The application only implements basic SNMP Get, GetNext, and Walk operations. Solution:
  - Implement SNMP GetBulk operation for retrieving large amounts of data efficiently
  - SNMP Set for modifying device configurations
  - Trap handling to enable the application to receive and process SNMP traps
  - SNMP Inform to provide a more reliable trap notifications with acknowledgements
- **Security:** SNMPv3 support is missing, leaving the communication vulnerable. Also implement user roles and permissions to restrict access to sensitive SNMP operations.
- **Error Handling:** Improve error handling and logging to provide more informative feedback to users and administrators.
- **Performance Optimization:** Profile and optimize code to reduce resource usage and improve response times.
- **Testing:** Develop a comprehensive testing strategy (unit tests, integration tests, etc.) to ensure the stability and reliability of the enhanced application.
- **Automated Quality Assurance:** Utilize Jenkins for continuous integration and deployment (CI/CD) and Integrate SonarQube for continuous code quality analysis.

We acknowledge that this project report may have limitations due to time constraints and the scope of our current knowledge. We welcome constructive feedback and contributions from the academic community and industry professionals. Your insights and expertise will be invaluable in refining and improving the work presented here.

References

[1] A guide to understanding snmp. https://www.solarwinds.com/assets/solarwinds/swresources/tech-tip/a_guide_to_understanding_snmp.pdf. Accessed: 2024-05-02.

[2] Jackson github repository. https://github.com/FasterXML/jackson. Accessed: 2024-05-02.

[3] Manage engine - what is snmp? https://www.site24x7.com/network/what-is-snmp.html. Accessed: 2024-05-01.

[4] Mibble api document. https://www.mibble.org/doc/. Accessed: 2024-05-15.

[5] Oracle - mib reference guide. https://docs.oracle.com/cd/E95618_01/html/sbc_scz810_mibguide/GUID-050076D7-8339-4AB8-8CF5-3D07DB3754DC.htm. Accessed: 2024-06-22.

[6] Pysmi document. https://pysmi.sourceforge.net/user-perspective.html. Accessed: 2024-05-30.

[7] Snmp evolution and version differences. snmp security models/levels details. https://www.noction.com/blog/snmp-versions-evolution-security. Accessed: 2024-05-30.

[8] Snmp4j api document. https://agentpp.com/doc/snmp4j-api-index.html. Accessed: 2024-05-01.

[9] Top snmp alternatives because snmp is dying (updated 2023). https://bestmonitoringtools.com/top-snmp-alternatives-because-snmp-is-dying/. Accessed: 2024-06-30.

[10] Mouhammd Alkasassbeh. The critical role of snmp in enabling network security. 10 2023.

[11] Nguyen Van Giang. Nghiên cứu giao thức snmp và ứng dụng xây dựng hệ thống giám sát mạng, gửi cảnh báo bằng tin nhắn. Master’s thesis, Da Nang University, 2013.

[12] Diep Thanh Nguyen. Giao thỨc snmp toÀn tẬp chương 1. https://mdungblog.wordpress.com/2020/01/06/ly-thuyet-giao-thuc-snmp-toan-tap/. Accessed: 2024-06-20.

[13] Diep Thanh Nguyen. Giao thỨc snmp toÀn tẬp chương 3. https://www.

`academia.edu/17659760/75731529_SNMP_toan_tap_Diep_Thanh_Nguyen_Chuong_3`. Accessed: 2024-06-23.

[14] Phat Huu Nguyen and Dung Duc Vu. Đề xuất xây dựng hệ thống quản lỳ, giám sát, và cảnh báo tập trung trong mạng vnpt: Array. *Journal of Science and Technology on Information and Communications*, 1(3):31–38, 2020.