Informatics

Device and Link Discovery in Industrial Ethernet Networks

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Computer Engineering

by

Thomas Marangoni
Registration Number 1634007

to the Faculty of Informatics

at the TU Wien

Advisor:	Univ.Prof. Dipl.-Ing. Dr.techn. Wolfgang Kastner
Assistance:	Projektass. Dipl.-Ing.(FH) Dieter Etz, MBA
	Projektass. Dipl.-Ing. Dr.techn. Thomas Frühwirth, BSc

Vienna, 11th February, 2022

Thomas Marangoni   Wolfgang Kastner

Erklärung zur Verfassung der Arbeit

Thomas Marangoni

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 11. Februar 2022

Thomas Marangoni

Acknowledgements

First and foremost, I'd like to thank my thesis advisors Wolfgang Kastner, Dieter Etz and Thomas Frühwirth for making it possible for me to write about this fascinating subject. When problems arose, it was also always possible to seek assistance. Theodor Mittermair was another helpful person; he provided additional hardware to test and was always willing to listen. Last but not least, I'd like to thank Julia Kren for her assistance and proofreading.

Abstract

Today production facilities are optimized for single products with high quantities. Switching the focus of production requires a lot of preparation, planning, and workforce. With Industry 4.0, interoperability and flexibility of sensors, actuators, and computing units are getting more important. The increased amount of devices makes the reconfiguration of the Industrial Ethernet structures more complex, which can result in long downtimes. Therefore, an automatic discovery mechanism is desired, which allows reconfiguring the network with a single button press.

This thesis implements a prototype of such discovery, it detects devices and their ports with Link Layer Discovery Protocol (LLDP) and prepares a network graph with the collected data for the Time-Sensitive Networking (TSN) Standard. The prototype was tested in a laboratory environment. It turned out that the functionality was limited by the manufacturers of the network devices used.

Contents

CHAPTER 1

Introduction

Industrial systems have been changing for years to become more efficient and dynamic. With the occurrence of computers and Programmable Logic Controllers (PLCs), the era of the third industrial revolution was started. The production lines got more automated and more computers got integrated. The sensors and actuators are connected over different bus systems with a computing unit. This unit is then connected over Ethernet to other units. The network then can be connected to a management system.

With the fourth industrial revolution, which is happening in the last years, even more, automated processes got added. The bus systems got reduced and the sensors got integrated into the network of the computing units. Now the network reaches from the sensors and actuators to the ERP systems. There has also been a shift from proprietary network systems to systems that resemble Ethernet. These networks often also have support for the Internet Protocol version 4 and version 6 (IPv4 and IPv6). Because of the widespread availability of Ethernet and IP networks, the industrial implementations of them are easy to understand for the technician and also easy to implement for the manufacturers.

In order to attain its goals, Industry 4.0 systems should be interoperable, easily virtualized, decentralized, be capable of real-time processing, be service-oriented and be appropriately modular for increased reconfigurability and reusability. An open and vendor-independent network standard for real-time communication is Time-Sensitive Networking (TSN). It works similar to standard Ethernet but needs special switches and routers to function. In a TSN network, routes get calculated to guarantee timings and bandwidths. There are centralized and decentralized approaches to calculate these routes. The centralized approach is simpler and easier to implement, but if the central component fails routes can't be calculated anymore. An important element of complex network infrastructures is the automatic discovery of its parts, such as devices and the links in between, because it is the basis for rapid reconfiguration. [HC19]

1.1 Motivation

Modern industrial assembly lines need to be able, to change their manufacturing focus as fast as possible. The time that is needed to change the production lines defines how fast the production can react to sudden events or new requirements. Open and vendor-independent protocols can help improve the independence of single hardware manufacturers.

TSN needs to be aware of the network topology to calculate network routes. In current implementations, this information must be manually added. This can be very time intensive depending on the size of the network and the changes. An automatic way to collect all needed information can reduce the time needed for changes drastically. It also allows to detect changes while the production is running and react to them. This makes it possible to calculate and apply new network routes on the fly. To increase the performance of searching better network routes, additional parameters are needed. This thesis assumes the following parameters to be helpful for such a task: the length of a connection, the medium of a connection, and the link speed. These few parameters can already improve the pathfinding on the network by a lot, because the limits of every connection are known exactly.

By discovering the direct neighbors from each device, the topology of the network can be generated. For discovering the direct neighbors, the vendor-independent and open Link Layer Discovery Protocol (LLDP) is utilized. To collect the connection and additional information, the also vendor-independent and open Simple Network Manager Protocol (SNMP) is used. To allow changing the production line while in operation, the system must be aware of changes on the network and they should be detected in under one second. Short detection times allow faster reconfigurations of the network, which reduces the downtime of the production. SNMP can detect changes on network interfaces of devices and send a notification to a specific host.

1.2 Structure of the work

The remaining chapters are structured in the following.

- **Chapter 2 - State of the Art:** This chapter gives an introduction about the technologies and standards used. It covers MIB, SNMP, LLDP and NETCONF.
- **Chapter 3 - Concept:** It describes the theoretical approach of the application to get the data from the network.
- **Chapter 4 - Implementation:** This chapter describes the practical approach of the implementation and its structure.
- **Chapter 5 - Evaluation:** This chapters describes how the application was tested, how to setup a test environment and what requirements must be met by the network devices.
- **Chapter 6 - Conclusion and Future Work:** It describes what issues have been found, the conclusion from them and possible improvements that can be done in future implementations.

CHAPTER 2

State of the Art

Various technologies are needed for detecting devices on the network. In this chapter, four technologies are explained: MIB, SNMP, LLDP, and NETCONF.

Management Information Base (MIB) describes a data structure and is used in different technologies like SNMP. The data is structured hierarchically in a tree. A saved object can be addressed via a sequence of numbers or an ASCII string sequence. It is using Formal Abstract Syntax Notation One (ASN.1) to describe the database and its entries.

Simple Network Management Protocol (SNMP) is a network monitoring and management protocol, developed in 1988 by the Internet Engineering Task Force (IETF). Over the years, multiple versions of SNMP have been released, but only three of them are mainly used: v1, v2c, and v3. The communication between network devices is handled by agents. There is one master agent, called AgentX, which communicates with other master agents. SNMP supports multiple instructions for getting and setting data from the devices.

Link Layer Discovery Protocol (LLDP) is an open and vendor-independent Layer 2 Protocol. Its purpose is to advertise the identity and capabilities to the connected network neighbors. Each device runs an agent, which collects the data sent by other devices. Because LLDP is a Layer 2 Protocol, it does not need an IP address to transmit its information.

Network Configuration Protocol (NETCONF) is a network monitoring and management protocol, published in 2006 as a successor of SNMP. It exchanges data encoded with XML with remote procedure calls. The advantages over SNMP are a modern, secure , and session-based communication, detection of capabilities, and the modern data description language YANG.

2.1 MIB - Management Information Base

The Management Information Base (MIB) is used to describe network entities. It was created to monitor, manage, and control network entities over a remote management protocol. It was first defined in RFC 1155[RM90b], RFC 1156[RM90a] and RFC 1157[FSDC90] by IEFT. Its main purpose was to provide a standardized Data Modeling Language for SNMP (Section 2.2), but it can also be used for other remote management protocols. The current valid standards for MIB are RFC 1155[RM90b] and RFC 1235 [RM91].

The MIB represents a hierarchical database, that is a tree. The root node of the tree is a MIB called MIB-I [RM90b], which can be extended by submodules defined in other MIBs. Elements in the tree are called *managed objects* and have a unique id on their level, which is called Object Identifier (OID). With the OID, it is possible to address one specific object in the tree, by describing the path from the root object to the object itself. The OID can be expressed as a sequence of numbers (.1.3.6.1.4.1) or as an ASCII string sequence (.iso.org.dod.internet.private.enterprise), however, they can also be mixed (.iso.org.dod.1.4.1). A simplified graphical representation of such a MIB tree structure can be found in Figure 2.1.

MIBs are using Formal Abstract Syntax Notation One (ASN.1)[ISO15] as a formal description language and are normally saved in text files with the file extension .mib. Other MIBs can be imported into one MIB and their definitions and types can be reused. MIBs are normally defined and maintained by IEFT and IEEE, but also by hardware and software vendors. There are multiple sources to acquire the MIBs, they can be found in RFC documents, on websites from vendors, as well as on general websites like `http://mibdepot.com`.

2.1.1 Names

A *name* is used to uniquely identify an object, regardless of the semantic association (standard document, network device, etc.), this concept is called Object Identifier (OID). An OID is a sequence of integers traversing a global tree. The root of the tree is connected to a number of labeled nodes via edges. Each node may have labeled children, also called a subtree. This concept may be repeated arbitrarily. A label is defined as a pairing of a brief textual description and an integer number. The root node itself is unlabeled and has at least three children (see Figure 2.1). These children are defined in [RM90b] and may be extended by other standards. The three main nodes are:

- ccitt(0) by the International Telegraph and Telephone Consultative Committee
- iso(1) by the International Organization of Standardization
- joint-iso-ccitt(2) by ISO and CCITT

The iso(1) node has a designated child node with the name org(3), that can be used by other international organizations. These organizations have their own subtrees below

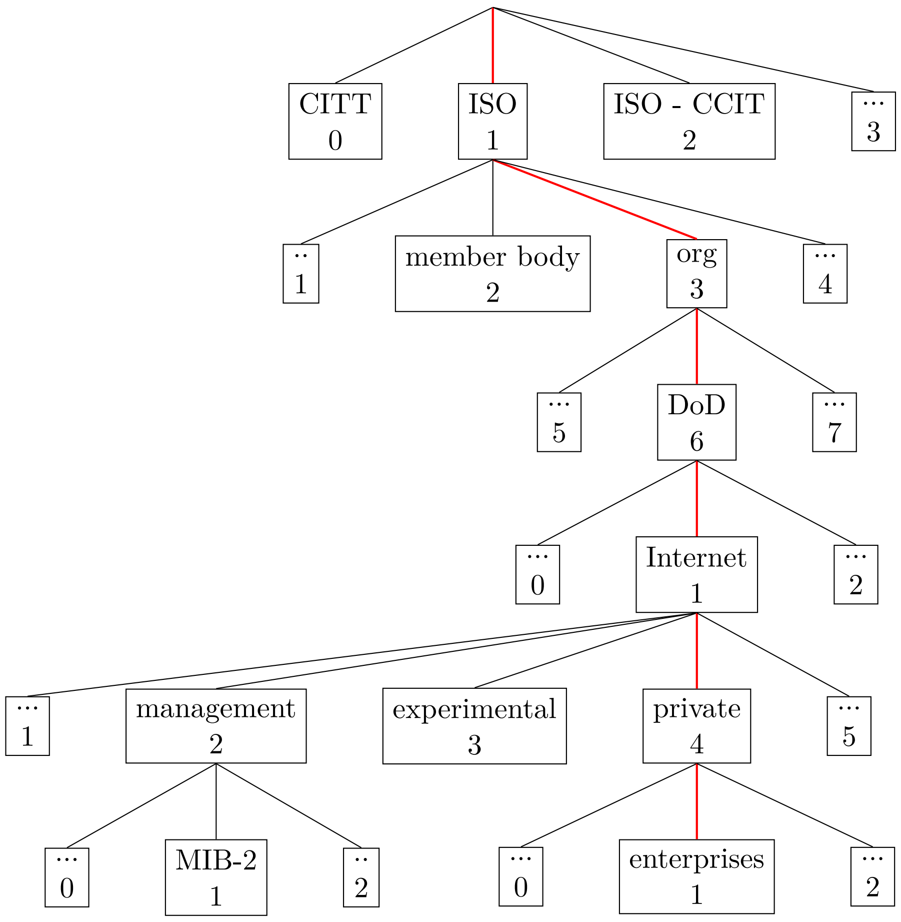


Figure 2.1: Shows a simplified MIB-Tree, where red marks the path for .1.3.6.1.4.1 or .iso.org.dod.internet.private.enterprise (Adaption based on [Mü06])

their child task. Two subtrees got assigned to the U.S. National Institutes of Standard and Technology and one of them has been transferred to the U.S. Department of Defense, dod(6). The Department of Defense has allocated a node to the *internet* community, to be managed by the Internet Activities Board (IAB). The definition of the *internet* can be seen in Listing 2.1. The *internet* object can be identified as 1.3.6.1 or as .iso.org.dod.internet. The IAB also defined four subnodes of internet(1), this can be seen in Listing 2.2.

```
internet   OBJECT IDENTIFIER ::= { iso org(3) dod(6) 1 }
```

Listing 2.1: Definition of a node, internet node used as an example (from [RM90b])

```
directory     OBJECT IDENTIFIER ::= { internet 1 }
mgmt          OBJECT IDENTIFIER ::= { internet 2 }
experimental  OBJECT IDENTIFIER ::= { internet 3 }
private       OBJECT IDENTIFIER ::= { internet 4 }
```

Listing 2.2: Definition of subnodes, internet subnodes used as an example (from [RM90b])

The directory(1) subtree has not been defined in [RM90b], but it discusses how OSI Directory may be used on the Internet. OSI Directory is also known as X.500 and is a system to lookup named objects.

The mgmt(2) subtree is used for IAB-approved documents. The subnode mgmt(1) is used for newer approved versions of the Internet standard Management Information Base, defined by RFC.

The experimental(3) subtree is used for Internet experiments and is managed by the Internet Assigned Numbers Authority of the Internet. They may define requirements on how this tree has to be used.

The private(4) subtree is used to define unilateral objects. It is managed by the Internet Assigned Numbers Authority(IANA) of the Internet. It has an enterprise(1) subtree, which can be used by companies to define their subtrees. To add a subtree in this subtree, an enterprise identification number needs to be registered.

2.1.2 Syntax

The structure of objects is described using the ASN.1 syntax, but the full generality of ASN.1 is not permitted. For primitives, only the following ASN.1 primitives are allowed: integer, octet string, the object identifier, and null. If an enumerated integer is used, there shall be no element with value 0.

Multiple non-primitive types can be used. The ASN.1 constructor sequence can be used to generate lists and tables. New types can be defined, but remain ASN.1 primitive types, lists, tables and, other application types. Other application types are:

- **NetworkAddress:** This choice type represents an address from one of several Internet protocol families.
- **IpAddress:** This type represents an internet address, represented as an octet string in network-byte order.
- **Counter:** This represents a non-negative integer value that can be increased. It wraps around if the maximum value is reached and starts incrementing from zero again.
- **Gauge:** This represents a non-negative integer value that can be increased and decreased, but latches at the minimum and maximum value.
- **TimeTicks:** This represents a non-negative integer value, that is getting incremented every hundredth of a second since an epoch. The epoch needs to be defined and explained in the description of this type.
- **Opaque:** This represents data with special encoding. Details can be found in RFC 1155 ([RM90b]).

2.1.3 Objects

Objects must be defined in a MIB. A MIB can contain a collection of object definitions. An object may refer to another object defined in the same or another MIB. The MIB defines the following items for each object:

- **Name:** Consists of a textual name and Sub-OID, and is also known as an object descriptor. It's not allowed to use 0 as a Sub-OID.
- **Syntax:** Defines the type of the object (see Section 2.1.2).
- **Definition:** Describes the usage of the object.
- **Access:** Defines the access rights of the object. Access permissions can be one of: "read-only", "read-write", "write-only", "not-accessible".
- **Status:** The object can be classified as "mandatory", "optional" or "obsolete".

A management protocol that is using MIBs must provide a way to access a simple non-aggregate object. The management protocol needs to define if an aggregate object can be accessed. If an object refers to multiple instances, it also needs to specify which instance of an object has been returned. An example of an object definition can be seen in Listing 2.3.

```
OBJECT:
-------
    atIndex { atEntry 1 }

Syntax:
    INTEGER

Definition:
    The interface number for the physical address.

Access:
    read-write.

Status:
    mandatory.
```

Listing 2.3: Example of an object definition (taken from [RM90b])

2.1.4 Versions

Every MIB document makes the previous version of the same document obsolete. It is forbidden to change the OID of an object between versions of a MIB document. The standard RFC1155 (see [RM90b]) defines rules to remain constant regarding semantics and simplify implementing support for multiple versions of a MIB [RM90b]:

"New versions may:

- declare old object types obsolete (if necessary), but not delete their names;
- augment the definition of an object type corresponding to a list by appending non-aggregate object types to the object types in the list; or,
- define entirely new object types.

New versions may not:

- change the semantics of any previously defined object without changing the name of that object."[RM90b]

2.1.5 MIB Files

Depending on the needed MIB file, the location where they are available varies. The core MIBs are defined in RFC1213 [RM91]. If MIBs from network hardware vendors are needed, they can be found on the websites of these vendors. But there are also more general sources: a search engine for MIBs can be found at `http://mibdepot.com` and some network monitoring tools also provide a collection of MIB files.

2.2 SNMP - Simple Network Management Protocol

The Simple Network Management Protocol (SNMP) was introduced in 1988 by the Internet Engineering Task Force (IETF). The reason for this was that at this time, computers got more popular and the need to connect them increased. The result has been that more devices have been included in IP networks. One goal of SNMP was to simplify the management of bigger networks and allow remote management of devices for system administrators, but the main goal was to monitor network components. The protocol was designed to provide a simple set of instructions. The most used implementation for SNMP is called "Net-SNMP"[1] and is available on all major operating systems. Over the years, multiple versions of SNMP have been released, but only three of them are mainly used: v1, v2c, v3. The versions are not backward compatible and the SNMP devices need to implement the detection of the used protocol.

[1] `http://www.net-snmp.org`

The communication between network devices is handled by agents. There is one master agent which communicates with other master agents. The master agent is getting the needed information from other agents on the same device. The agents are exposing data from MIBs (see Section 2.1) as variables. They also handle a set of instructions, if data in their MIB tables are affected.

The most important features of SNMP are *get*, *set* instructions and *traps*. With the get instruction, it is possible to request data from devices by specifying the OID of the data set. The *set* instruction can modify the data saved on a specified OID. There is also a *walk* instruction, that allows querying multiple OID at once. To accomplish this, the SNMP agent sends multiple *get* requests. There is also an advanced version of *walk*, called *bulkwalk*, that can improve the performance when querying multiple OIDs.

The simplest and first version of SNMP was v1. It has no security mechanisms besides community strings and only supports up to 32-bit counters while the communication is in plain text. In 1993, SNMP v2c was designed, which is a subversion of v2. It introduced an inform command, which is similar to a trap, but with acknowledgments. The focus of v2c has been performance improvements and better error handling. With v3 security mechanisms, encryption and authentication have been introduced.

SNMP is now also used to configure network devices, which is a new purpose for the protocol. Because of this change in use-case, the IEFT introduced a successor standard called NETCONF (see Section 2.4) in 2006. Until today, SNMP is still the most used open protocol to configure and monitor network devices. For simplicity, the next sections will only contain the most recent SNMP definitions.

2.2.1 Architecture

The architecture of SNMP is defined in RFC 3411 [HWP02]. Its goal is to make SNMP modular to allow evolution over time.

The SNMP management system is a network with multiple devices that can be managed by SNMP. These devices are called SNMP entities and contain a command responder application and a notification originator application. These applications need to have access to management instrumentation, also called agents. The network must contain at least one manager, which is a device that manages other devices. It is an SNMP entity containing a command generator application or a notification receiver application, or both applications can be contained. A compatible management protocol between communicating SNMP entities is needed.

Managers are devices that monitor and control other SNMP entities. They are often implemented on servers or end-user devices for temporary usage. Managed devices are hosts, routers, switches, servers, etc, that are controlled and monitored via their management information base.

If a security model is used for SNMP, it should protect against modification of information and masquerade. There should also be protection to prevent message stream modifications and disclosure. A security model does not need to protect against denial of service and traffic analysis. Denial of service floods the network with messages, which results in a failure of the network. In that case, SNMP can not operate anymore. The attack method traffic analysis uses package analysis to predict operations. These predictions are easy due to the design of SNMP. For that reason, it was decided that protection against this attack vector is not needed.

SNMP messages are encapsulated in Protocol Data Units (PDUs). Because SNMP messages can have different use cases, they are classified into seven categories. Every PDU type can be classified into multiple categories. The following classes exist according to RFC 3411 ([HWP02]):

- Read Class - For operations that retrieve management information, for example: GetRequest-PDU, GetNextRequest-PDU, and GetBulkRequest-PDU.
- Write Class - For operations that modify management information, for example: SetRequest-PDU.
- Response Class - For operations which are a response of a previous request, for example: Response-PDU, Report-PDU.
- Notification Class - For operations that are sending notifications to a notification receiver, for example: Trapv2-PDU, InformRequest-PDU.
- Internal Class - For operations that exchange information between SNMP engines, for example: Report-PDU.
- Confirmed Class - For operations that are causing the receiving SNMP engine, to send back an acknowledgment, for example: GetRequest-PDU, GetNextRequest-PDU, GetBulkRequest-PDU, SetRequest-PDU, and InformRequest-PDU.
- Unconfirmed Class - For operations that are not causing the sending of an acknowledgment, for example: Report-PDU, Trapv2-PDU, and GetResponse-PDU.

SNMP Entity

The purpose of the SNMP engine is to provide services to send, receive, authenticate and encrypt messages, while also handling the access control to managed objects. For these tasks, it contains a Dispatcher, a Message Processing Subsystem, a Security Subsystem, and an Access Control Subsystem. Only one SNMP engine exists per SNMP entity. An overview of the subsystems, their naming, and hierarchy can be found in Figure 2.2.

Every SNMP engine is identified by a snmpEngineID. This identifier must be unique within an administrative domain. Because only one SNMP engine exists per SNMP entity, this identifier can also be used to identify the SNMP entity.

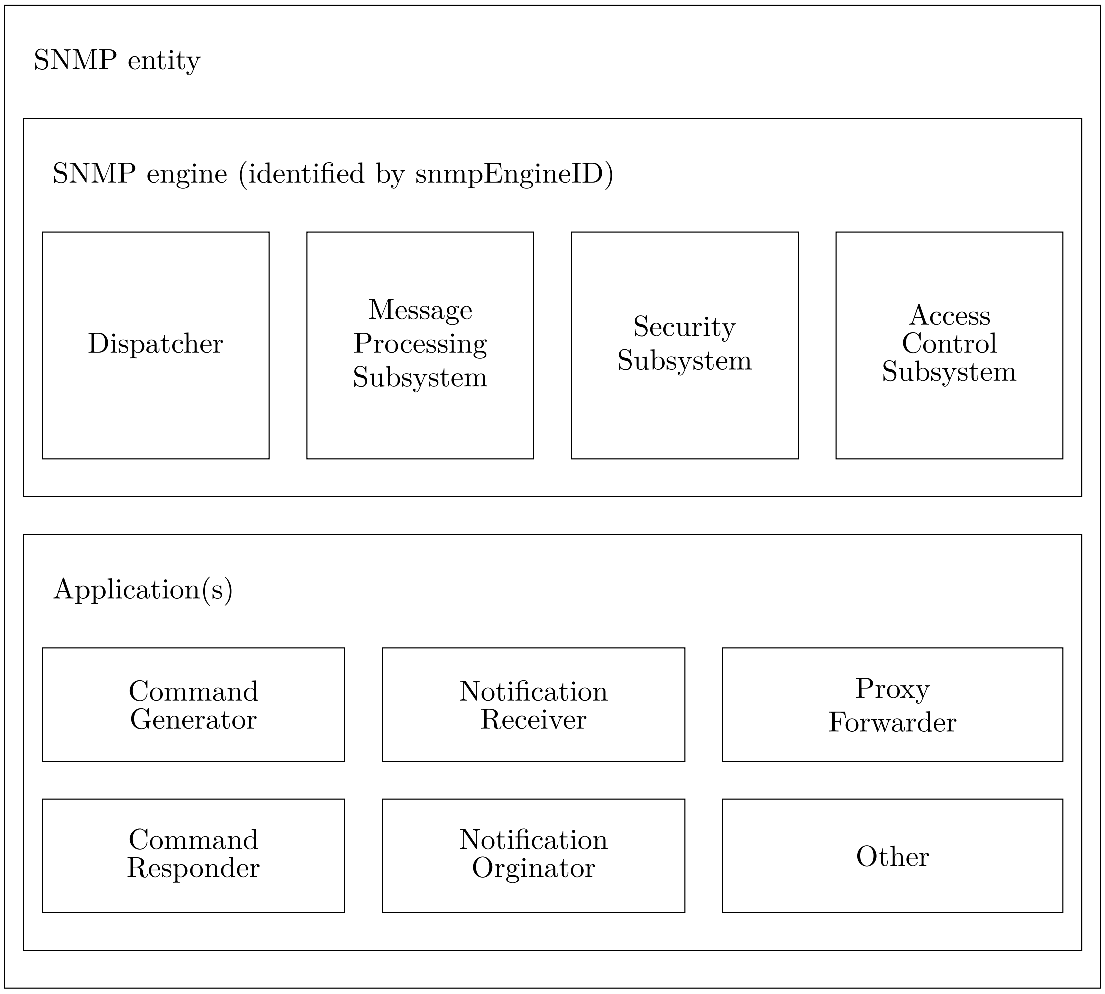


Figure 2.2: Naming and hierarchy of single elements from a SNMP entity (Adapted from [HWP02])

The Dispatcher is used to send and receive SNMP messages. Therefore, it must be able to determine the SNMP version of a message. It provides an abstract interface for the SNMP application to send and receive messages over the network. Only one Dispatcher can exist per SNMP engine.

An SNMP engine also contains one Message Processing Subsystem, which is responsible for preparing messages for sending and extracting data from received messages. One Message Processing Subsystem can contain multiple Message Processing Models (e.g. SNMPv3, SNMPv2c, SNMPv1), whereby model processes its matching SNMP message.

The Security Subsystems is part of the SNMP engine and provides services like authentication and privacy. The subsystem may contain one or more models. Only one Security Subsystem can exist per SNMP engine.

Lastly, another subsystem included in an SNMP engine is the Access Control Subsystem, which provides one or more Access Control Models. The default Access Control Model is View-Based, only a predefined set of MIB Objects can be viewed or edited. Only one of these subsystems can exist per SNMP engine.

Figure 2.3 shows a typical setup of an SNMP Manager. It has one or more command generators and notification receivers. An instruction is generated in a command generator application and the instruction is sent to the Dispatcher. In the Dispatcher the instruction is encapsulated in a PDU. After that, the Message Dispatcher transfers the data to the Message Processing Subsystem, where the correct SNMP version and its properties are appended. The Message Processing System transfers the message to the Security Subsystem, where security-related information is added. In the next step, the message is passed to the Transport Mapping. This step adds transport-related and transport-protocol-related information to the message. After that, the message is embedded into the Transport Protocol and is delivered over the network. Receiving a message and parsing it works in reverse order. The only difference is that a notification receiver application is receiving the message instead of a command generator application. For further information, see Section 4.6.1 in RFC 3411 on page 38 [HWP02].

Figure 2.4 shows a typical setup of an SNMP Agent, which has one or more command receiver applications and notification originator applications. The process of receiving and decoding a message is the same as for an SNMP manager. After the message is decoded, it can be sent to a proxy forwarder application, which is forwarding the message to another SNMP entity, or to the notification originator application and command responder application. These applications may check if the sender is authorized to receive or write the content of the requested MIB entry. They can check if the access is granted by communicating with the access control unit. If access is granted, the content of the MIB entry may be sent back to the origin of the request in reverse order of receiving the message. For further information, see Section 4.6.2 in RFC 3411 on page 39 [HWP02].

The snmpEngineID only describes the SNMP engine and the SNMP entities. Because of that, every context (instance of MIB Objects) also needs to be uniquely identified. Every context has a readable name (e.g. bridge0, bridge1), and additionally, the contextID is provided. If the contextID is the same as the snmpEngineID, the context is only valid on this SNMP entity. If the context is valid over multiple SNMP entities, another contextID may be used, but the combination of contextName and contextID must be unique in the administrative domain. It is also possible that multiple combinations of contextID and contextName exist to identify the same object.

With SNMPv3 three levels of security have been introduced:

- noAuthNoPriv - without authentication and without privacy
- authNoPriv - with authentication and without privacy
- authPriv - with authentication and with privacy

The security levels are sorted like this: noAuthNoPriv < AuthNoPriv < AuthPriv. Every message has an associated security level and all subsystems and applications are required to provide a value of security level or to hold the same security level while processing data.

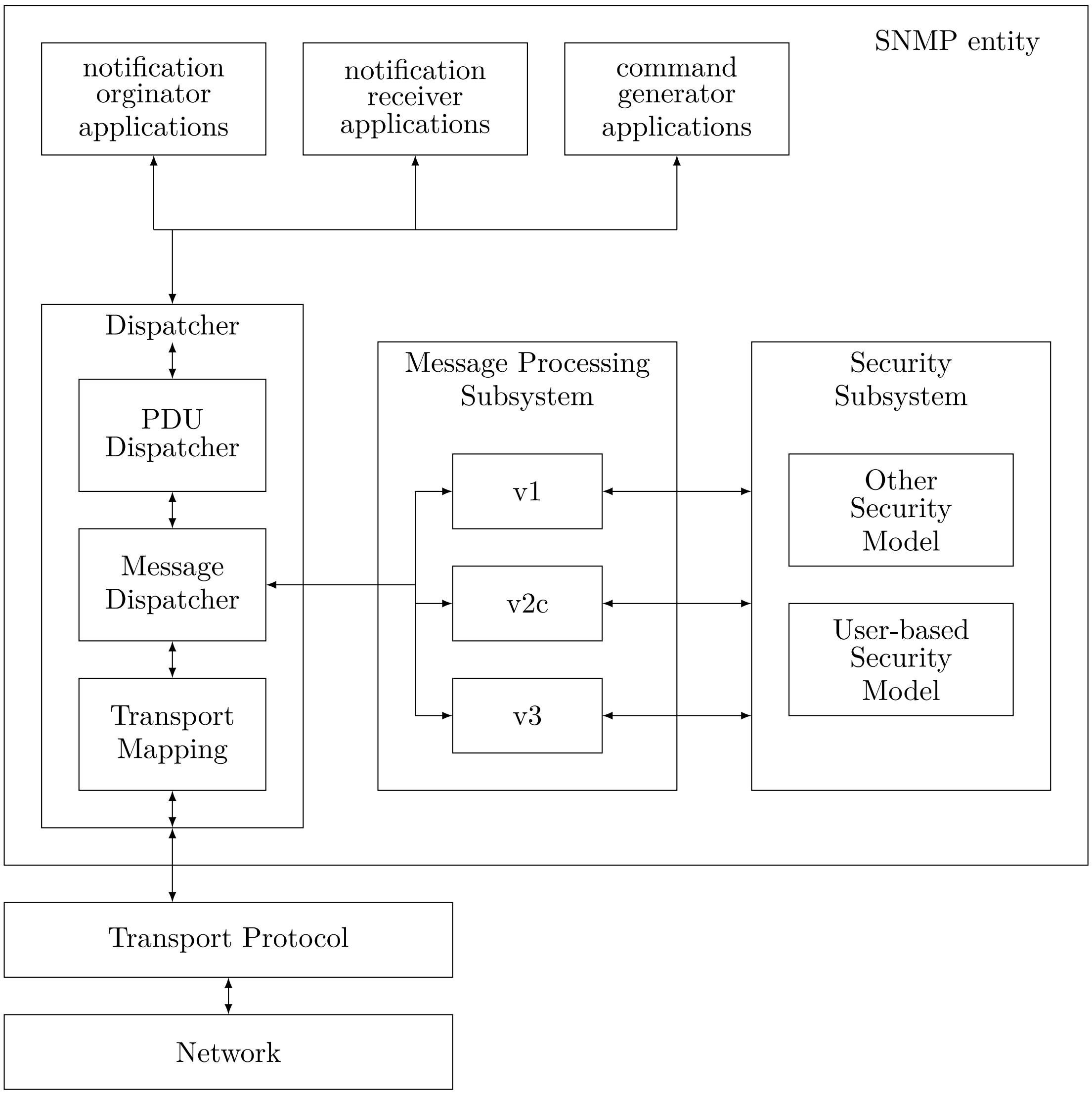

SNMP entity
notification orginator applications
notification receiver applications
command generator applications
Dispatcher
PDU Dispatcher
Message Dispatcher
Transport Mapping
Message Processing Subsystem
v1
v2c
v3
Security Subsystem
Other Security Model
User-based Security Model
Transport Protocol
Network

Figure 2.3: Typical structure of an SNMP manager (Adapted from [HWP02])

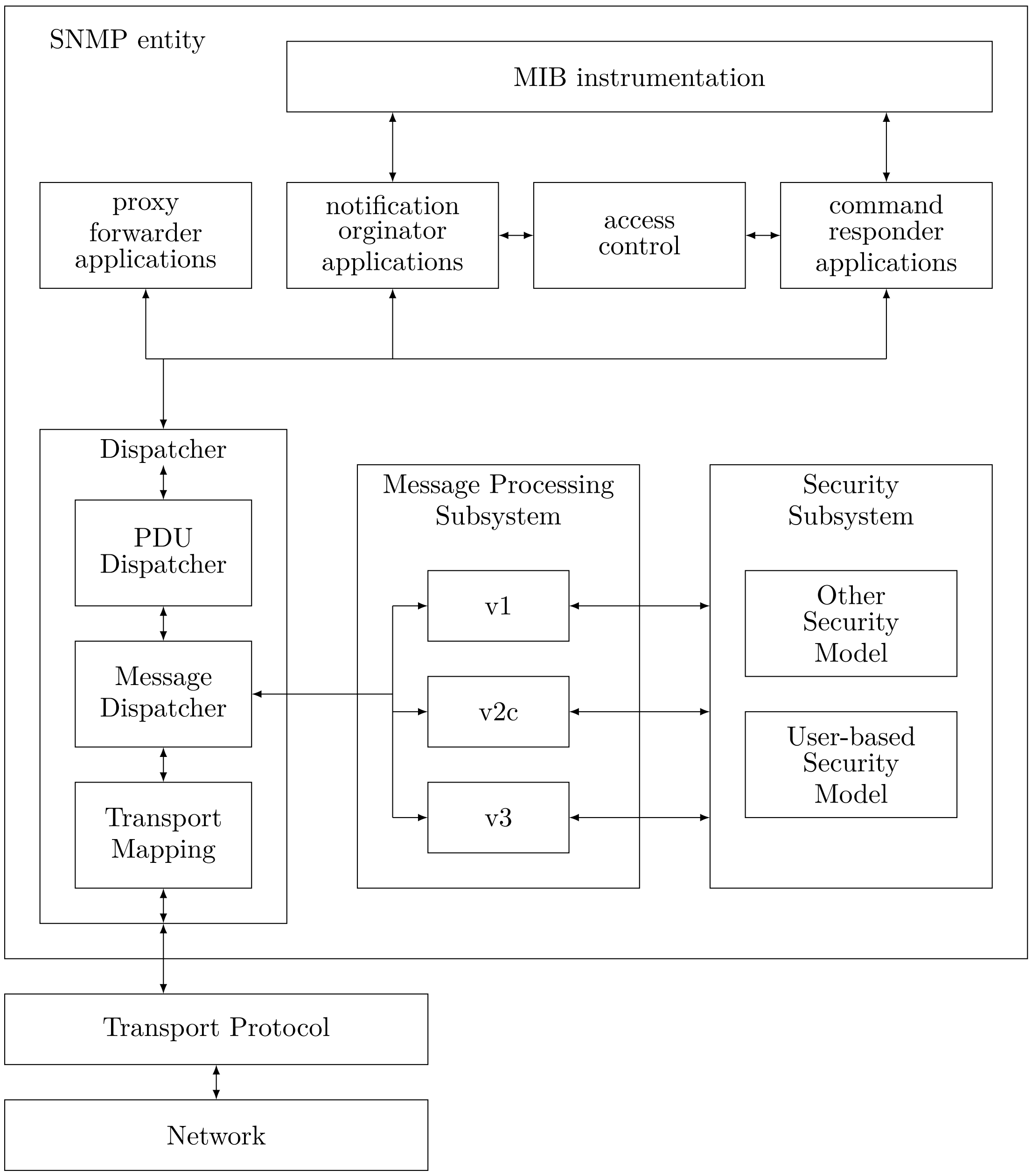


Figure 2.4: Typical structure of a SNMP agent (Adapted from [HWP02])

2.2.2 Protocol

The basic functions of the protocol are defined in RFC3416 [Pre02b]. It is defined that it should be possible to retransmit a request. Under normal circumstances, the receiver is required to send a response to the originator of the message. If no response is received, the originator can decide if the message should be retransmitted, but the application needs to act responsibly with the retransmission frequency and duration.

The message size is limited to the capabilities of all SNMP entities that are part of the communication. The maximum size of a message is defined by the smallest value of the maximum size that both the receiving SNMP entity can accept and the sending SNMP entity can generate. The maximum size an entity can receive may be known. Otherwise the size is defined by the transport domain when sending the message. The maximum size an entity can generate is defined by local constraints. Each transport mapping defines a minimum size, which every entity needs to be able to consume and produce.

The goal of the GetBulkRequest-PDU is to reduce the number of messages. Therefore, messages should be as large as possible. To achieve this, the PDU is embedding the limits of supported message sizes from the command responder and command generator into the request. It can happen that the maximum message size is bigger than the MTU of the network, leading to fragmentation of the message. Message fragmentation can decrease the reliability of the transfer.

SNMP only requires an unreliable datagram. The protocol encourages the use of UDP to send messages, but other transport mappings can also be used.

PDUs contain a request-id field. The request-id is generated by the sending SNMP entity and added to the Request-PDU. If a response for that request is sent by the receiving entity, the request-id from the received request is used. This makes it easy to match sent requests with the incoming response. To calculate round-trip time, the request-id needs to be changed if the PDU is retransmitted. A non-zero value of the error-status field in the Response-PDU indicates that an error occurred. The error-index field represents which variable-binding caused the error. Thereby, the error-index is the same as the index of the variable-binding in a variable-binding list.

The different PDUs can contain fields that are not relevant for the specific PDU type. These fields are ignored while processing the PDU by the receiving entity. Every field, even the ignored ones, must have a valid ASN.1(2.1.2) syntax and encoding. There can be as many variable bindings as fit into the message size limit, but no more than 21474483647.

The GetRequest-PDU is generated and is transmitted by the request of an application. Each variable binding in the variable-binding list is getting processed by the receiving SNMP entity. Every field of the Response-PDU has the same value as the corresponding sent PDU. They are processed as follows: If the variable name exactly matches the name of the accessible variable, then the variable field is set to the value of the accessible variable. If the name does not match exactly and does not have an OID prefix that matches the requested OID prefix, the value is set to "noSuchObject". Otherwise, the variable field is set to "noSuchInstance".

If any other error occurs, the Response-PDU is set to the same value as the received fields and the error-status field is set to "genError". Otherwise, the error-status field is set to "noError" and the error-index field is set to zero. If the message is shorter than or equals the maximum message size, the Response-PDU is transmitted to the originator. Otherwise, an alternative Response-PDU is generated, with the same values as the received PDU, the error-status field set to "tooBig" and the error-index field set to 0. If this alternate Response-PDU is bigger than the maximal message size, the message is dropped and the snmpSilentDrops counter is increased.

The GetNextRequest-PDU is generated and is transmitted by the request of an application. The variables are processed as follows: The successor variable of the lexicographically order variable names is located. The found name and the value are set in the Response-PDU to the same index as the variable of the GetNextRequest-PDU. If the requested variable has no successor, then the value is set to "endOfMibView" and the name is set to the name of the variable from the GetNextRequest-PDU. The errors are handled in the same way as for the GetRequest-PDU.

The GetBulkRequest-PDU is generated and transmitted by the request of an application. The purpose of this PDU is to request a large amount of data, but it also can be used to retrieve large tables rapidly and efficiently. Every variable is processed on receive and a Responds-PDU is produced with the request-id, but the variables are not mapped one-to-one like in the GetRequest-PDU, GetNextRequest-PDU, and SetRequest-PDU. This PDU has a non-repeaters and max-repetitions field, which is not included in other PDUs. The non-repeaters field contains the number of variables in the variable list where a single-lexicographic successor can be returned, while the max-repetitions field contains the number of lexicographic successors for the remaining variables.

In the following description, we define these variables: N is the value of the non-repeaters field, M is the value of the max-repetitions field, and R is the maximum of variables minus the value of non-repeater fields N. The variables get requested as follows: One variable of the Respond is assigned to the first N variables, M variables are requested for the R variables. This results in a total number of $N + (M * R)$ requests. A more detailed explanation is given in RFC 3416 Section 4.2.3 on page 15 [Pre02b]. The errors are handled the same as for the GetRequest-PDU.

The SetRequest-PDU is generated and is transmitted by the request of an application. The receiving entity parses the PDU and produces a response PDU. In the first step, the variables are validated, and if all validations succeed, the variables get altered. These validations are performed:

- Access for the variable name is validated. If no access is granted, the error-status is "noAccess".
- If the OID prefix does not match any variable that can be created or modified, the error-status is set to "notWriteable".
- If the ASN.1 type is inconsistent with the required type, the error-status is set to "wrongType"
- If the required length is inconsistent, the error-status is set to "wrongLength"
- If the encoding does not match ASN.1, the error-status is set to "wrongEncoding"
- If a value is specified that can not be set, the error-status is set to "wrongValue".
- If a specified variable does not exist or can not be created, the error-status is set to "noCreation".
- If a specified variable does not exist and can not be created under circumstances, the error-status is set to "inconsistentName".
- If a variable is specified that can not be created or modified, no matter of the new value, the error-status is set to "notWriteable".
- If a value can not be set or modified, but is getting modified, the error-status is set to "inconsistentValue".
- If the variable needs allocation of a resource that is not available, the error-status is set to "resourceUnavailable".
- If the processing fails for another reason, the error-status is set to "genErr".

If a requested variable does not exist, it is created and the value is assigned to it. When the same variable should be set multiple times with different values in one request, the behavior is implementation-specific. If an assignment fails, all other assignments from the same request get reverted and the error-status is set to "commitFailed". If it is not possible to revert all assignments, the error-status is set to "undoFailed". It is recommended for implementations to avoid both cases.

The SNMPv2-Trap-PDU is generated and is transmitted on behalf of a notification originator application. These PDUs are often used to inform remote entities about events that occurred. The PDUs are not getting confirmed and the destinations for these PDUs are implementation-dependent. The first variables in an SNMPv2-Trap-PDU are sysUpTime.0 and snmpTrapOID.

The InformRequest-PDU is generated and is transmitted on behalf of a notification originator application. Like SNMPv2-Trap, it informs about an event without guarantee of delivery, but adds confirmation. The destination to which it is sent is defined by the notification originator. The first two variables in the PDU are sysUpTime.0 and snmpTrapOID. If an InformRequest-PDU is received, the entity validates the received PDU. If the validation succeeds, the entity presents its content to the appropriate application, generates a Respond-PDU to confirm the PDU, and transmits it.

2.2.3 Transport Subsystem

The goal of the Transport Subsystem ([SH09]) is to provide access to lower-layer security mechanisms. This can include authentication of the sender, encryption, data integrity, and timeliness checking. Examples for these techniques are TLS, Simple Authentication and Security Layer (SASL) as well as SSH. Transport Mappings need to use security protocols the way they were designed. Modified usage often implies that the protocol can not deliver the expected security characteristics. They also should not modify the underlying security protocol, which might change it is security characteristics.

Transport Subsystems must meet the following requirements:

- They must follow the architectural modularity requirements.
- If establishing the connection between two entities for every message is too expensive, sessions can be used.
- The security model must use the same or a higher security level than provided by the transport model.
- A message processing model might unpack security-specific parameters from incoming messages, before calling a specific security model.
- Authentication and authorization are separated. Authentication is handled by the security model and authorization by the access control model.

A security name is a human-readable value, that describes the used security protocol and is used by SNMP applications and the Access Control Subsystem. The transport mapping security name is a human-readable value and is used by the transport and security models to identify the used transport security protocol.

The security value consists of two values: the requested security level, and the transport security level. The first value represents the minimum security level required for transport models. It can be used to prevent sending messages via sessions that are not secure enough. The second value represents the security level offered by the session. The security model can use this to ensure the minimum security level for incoming messages for a session.

2.2.4 Transport Mapping

Transport Mappings are used to describe the way how data is transferred between multiple SNMP entities. An SNMP entity can use multiple transport mappings, but every entity should provide access via UDP over IPv4. If an entity supports IPv4 as Transport Mapping, it must also implement UDP over IPv4. The definition for Transport Mappings can be found in RFC 3417 [Chr01].

It should be mentioned that popular Transport Mappings are "SNMP over UDP over IPv4" [Chr01], "SNMP over UDP over IPv6" [Chr01] and "SNMP over IEEE 802 Networks (Ethernet)" [SJ06]. To be able to communicate with SNMPv1 entities, a proxy is needed [FWLR00].

2.3 LLDP - Link Layer Discovery Protocol

LLDP (Link Layer Discovery Protocol) is an open and vendor-independent Layer 2 (Data Link Layer) Protocol, which is defined in IEEE Std 802.1AB [IEE16]. Its purpose is to advertise the identity and capabilities to the connected network neighbors. Each device needs to run an LLDP Agent, which collects the information of the neighbors and publishes its information periodically.

LLDP data are transferred via Ethernet frames, with a specific multicast destination addresses and a specific EtherType. These frames with the specific multicast destination addresses are not forwarded by switches and routers, but other addresses are permitted.

LLDP is a one-way protocol, the agent can receive and transmit information, but does not directly communicate with other agents. Because of that, LLDP allows to separately enable the receive or transmit functionality. The LLDP agent itself does not provide any functionality to process information. Received information is stored in multiple MIB tables and can be read per SNMP (see Section 2.2) or other provided interfaces [IEE16].

2.3.1 Communication

For communication, Ethernet frames are used. They contain the destination address, the source address, the EtherType, and the LLDP Data Unit (see Table 2.1). The destination address is a multicast MAC address. It can be a specified and reserved MAC address by IEEE Std 802.1AB [IEE16], but other MAC addresses are also permitted. The Ethernet frame must contain the EtherType "0x88CC". If the Agent supports EtherType encoding, it shall be encoded in the LLDP Data Unit header. The source address must be the MAC Address of the sender or the sender port.

Destination Address	Source Address	EtherType	LLDPDU
Multicast Address	MAC Address of LLDP Device	0x88CC	See 2.3.2

Table 2.1: Simplified structure of a Ethernet Frame for LLDP

Supported destination Addresses

IEEE Std 802.1AB ([IEE16]) contains predefined MAC address groups, that allow filtering the proposal of the Ethernet frames by different network components. These network components are Two-Port MAC Relay (also known as cross-connect Ethernet devices), Service Virtual Local Area Network (S-VLAN), Customer Virtual Local Area Network (C-VLAN), and MAC Bridges (Network Switch).

Nearest bridge group (01:80:C2:00:00:0E): Packages with this destination address are not propagated by Two-Port MAC Relay (TPMR) components, S-VLAN components, C-VLAN components, and MAC Bridges. This means, packages only contain information about devices attached to the same LAN segment as the recipient.

Nearest non-TPMR bridge (01:80:C2:00:00:03): Frames using this group as destination are not propagated by S-VLAN components, C-VLAN components, and MAC Bridges. But they are propagated by Two-Port MAC Relay (TPMR) components.

Nearest custom bridge (01:80:C2:00:00:00): LLDPDU sent with this destination address group are not propagated by C-VLAN components and MAC Bridges. But they are propagated by S-VLAN components and Two-Port MAC Relay (TPMR) components.

Custom groups and custom addresses: Custom groups or individual addresses could be used, but it can happen that the custom address or group is not implemented on the recipient.

Table 2.2 shows the needed implementation for each destination address group and network device category [IEE16].

2.3.2 LLDP Data Unit (LLDPDU)

The LLDP Data Unit contains a sequence of TLVs (see Section 2.3.3). These contain the information sent by the Agents. The Data Unit frame contains three mandatory TLVs and can contain multiple optional ones.

Every LLDPU needs to contain a number of octets. The octets are starting with 1 and the bits in the octets are numbered from 1 to 8, where 1 is the low-order bit. When bits

Address	C-VLAN Bridge	S-VLAN Bridge	TPMR Bridge	End Station
Nearest bridge	Mandatory	Mandatory	Mandatory	Mandatory
Nearest non-TPMR bridge	Mandatory	Mandatory	Not Permitted	Recommended
Nearest customer bridge	Mandatory	Not Permitted	Not Permitted	Recommended
Any other group MAC address	Permitted	Permitted	Permitted	Permitted
Any individual MAC address	Permitted	Permitted	Permitted	Permitted

Table 2.2: Support for MAC addresses (Adaption based on [IEE16])

Name	**Description**	**Requirement**
Chassis ID TLV	An unique identifier from the device sending this package	Mandatory
Port ID TLV	An identifier for the port the receiving device is connected to.	Mandatory
Time to Live TLV	The time in seconds how long the data can be identified as valid	Mandatory
Optional TLV	See 2.3.3	Optional
...	...	...
Optional TLV	See 2.3.3	Optional
End of LLDPDU TLV	Marks the end of a TLV sequence.	Optional

Table 2.3: Structure of a LLDPDU Frame (Adaption based on [IEE16])

in an octet are represented as a binary number, the highest bit is the most significant bit. If octets are used to represent a binary number, the lowest octet contains the most significant number. When the LLDPU or parts of it are represented using a diagram, the lowest octet is shown most left of the page and within an octet the highest numbered bit is shown most left [IEE16].

2.3.3 Type, Length, Value (TLV)

The TLV format represents information and contains its type and length. The first TLV section (see Table 2.4) contains the type of the information and is 7 bits long. The next section includes the length of the information string, it is 9 bits long and represents the total length of the information string in octets. The last section of the TLV contains the information string, which can be 0 to 511 Octets long. The information string can contain binary or alphanumeric data. For encoding alpha-numeric data ,UTF-8 [Yer03] shall be used.

TLV type	TLV information string length	TLV information string
7 bits	9 bits	$0 \leq n \leq 511$ Octets

Table 2.4: Structure of a TLV (Adaption based on [IEE16])

TLV types can be grouped into two categories. The first category contains basic information for network management and is required by every LLDP implementation. Each TLV of this category has its type specified. The second category is organizationally specified. Each TLV of this category has the same type, but includes an organizationally unique identifier (OUI) and an organizationally-specific TLV subtype. This category is specified by standard groups like IEEE 802.1 and IEEE 802.3. The different LLDP TLV Types as defined in [IEE16] are briefly summarized in the following.

type	name	usage	reference
0	End Of LDDPDU	Optional	2.3.3
1	Chassis ID	Mandatory	2.3.3
2	Port ID	Mandatory	2.3.3
3	Time to Live	Mandatory	2.3.3
4	Port Description	Optional	2.3.3
5	System Name	Optional	2.3.3
6	System Description	Optional	2.3.3
7	System Capabilities	Optional	2.3.3
8	Management Address	Optional	2.3.3
9-126	Reserved	—	—
127	Organizationally Specific	Optional	2.3.3

Table 2.5: The value of all TLV types (Adaption based on [IEE16])

End Of LLDPDU TLV

This TLV type is optional and uses type value 0. It is 2-octets long and contains only zeros, and it is used to mark the end of a TLV sequence in an LLDPDU frame.

Chassis ID TLV

This TLV type is mandatory and uses type value 1. It is up to 258 octets long and contains an identifier that can be associated with the transmitting agent. It contains an 8-bit subtype at the beginning of the information string because there are several ways to identify an agent. The subtype can classify a chassis component, interface alias, port component, MAC address, network address, interface name, or a locally assigned string. Each LLDPDU shall only contain one Chassis ID TLV and it shall be the first TLV in an LLDPDU frame.

Port ID TLV

This TLV type is mandatory and uses type value 2. It is up to 258 octets long and contains an identifier, which can be associated with the port of the transmitting agent. It contains an 8-bit subtype at the beginning of the information string because there are several ways to identify the port of an agent, like the Chassis ID. The subtype can classify an interface alias, port component, MAC address, network address, interface name, agent circuit ID, or a locally assigned string. Each LLDPDU shall only contain one Port ID TLV and it shall be the second TLV in an LLDPDU frame. The port ID shall be constant while the port remains operational.

Time To Live TLV

This TLV type is mandatory and uses type value 3. It is 4 octets long and represents the number of seconds that the receiving agent can assume the attached information as valid. If the time to live contains a non-zero value, the receiving agent is notified to replace old saved information with the attached information. If the time to live is set to zero, the receiving agent is notified to delete all information associated with this agent. This may be used to indicate a shutdown of the sending port. Each LLDPDU shall only contain one Time to Live TLV and it shall be the third TLV in an LLDPDU frame.

Port Description TLV

This TLV type is optional and uses type value 4. It is up to 257 octets long and contains an alpha-numeric string that describes the sending port. If *ifDesc* object [MK00] is implemented, it should be used for this field. Each LLDPDU shall only contain one Port Description TLV.

System Name TLV

This TLV type is optional and uses type value 5. It is up to 257 octets long and contains an alpha-numeric string of the system name from the sender. If *sysName* object [Pre02a] is implemented, it should be used for this field. Each LLDPDU shall only contain one System Name TLV.

System Description TLV

This TLV type is optional and uses type value 6. It is up to 257 octets long and contains an alpha-numeric string of a textual description from the sender. If sysDescr object [Pre02a] is implemented, it should be used for this field. Each LLDPDU shall only contain one System Description TLV.

System Capabilities TLV

This TLV type is optional and uses type value 7. It is 6 octets long and contains bit mask of functions the system is capable and functions the system has enabled. A system can have multiple capabilities and multiple can also be enabled. The bitmask fields are two octets long and can represent following capabilities:

- repeater
- MAC Bridge component
- 802.11 Access Point
- router
- telephone
- DOCSIS cable device
- station only
- C-VLAN component
- S-VLAN component
- Two-port MAC Relay
- other

Each LLDPDU shall only contain one System Description TLV.

Management Address TLV

This TLV type is optional and uses type value 8. It is up to 169 octets long and contains the management address of a device to reach higher layer protocols. This TLV type also provides space to include the system interface number and an object identifier (OID), if they are known [IEE16]. In Table 2.6 the structure of the information string can be seen.

management address string length	management address subtype	management address	interface number subtype	interface number	OID string length	OID
1 octet	1 octet	1-31 octets	1 octet	4 octets	1 octet	0-128 octets

Table 2.6: Structure of the information string from a Management Address TLV (Adaption based on [IEE16])

The management address subtype contains the type of an address represented in the management address field. It can be one of the types defined in ianaAddressFamilyNumber module [Edi02].

The management address should contain a Layer 3 address. If no management address is available, the field should contain the MAC address of the sender or the MAC address of it is port.

The interface number subtype describes the value of the interface number field. The following values are defined: Unknown, ifIndex, system port number. If the type is Unknown, the interface number field must only contain zeroes.

The object identifier (OID) field contains an identifier of the type of the hardware component or the protocol entity associated with the indicated management address. The OID shall be ASN.1 [ISO15] encoded. If no OID is provided, this field shall be skipped.

At least one Management Address TLV should be included in every LLDPDU, the address provided should be the address with the best management capabilities. If more than one Management Address TLV is provided, each address shall be different.

Organizationally Specific TLVs

This TLV type is optional and uses type value 127. This TLV type was created to extend the information that can be advertised by an agent. This type can be specified by a standardizing organization, but also by individual software and equipment vendors. The information string contains a three octets long organizational unique identifier (Clause 8 of IEEEStd 802-2014), one-octet long unique organizationally defined subtype, and an up to 507 octets long organizationally defined information string.

2.4 NETCONF - Network Configuration Protocol

The Network Configuration Protocol (NETCONF) was first published in December 2006 as RFC 4741[Enn06]. It was created as the successor to SNMP (see Section 2.2) to modernize, simplify, and unify network management. The protocol was updated in June 2011 with RFC 6241[EBBS11]. The updated version defines the protocol as follows.

NETCONF uses remote procedure calls (RPC) to send and receive data. The data are encoded in XML. For communication between client and server, secure and connection-oriented sessions are used. Administrative devices are called clients, while network devices are called servers. A server can support one or more sessions. Configuration changes can only be done by authorized sessions and the effects of these changes must be visible to all active sessions.

It is possible for the client to discover the capabilities of a server. This helps the client to adapt its behavior to the capabilities of the server. Capabilities can have dependencies on other capabilities and a server must support all depending capabilities to support a capability.

The protocol is separated into four layers:

- Secure Transport (Layer 1) - The task of this layer is to provide a path between client and server across the network. The path can be provided via any protocol that fulfills the basic requirements.
- Messages (Layer 2) - This layer provides a mechanism for encoding RPCs and notifications.
- Operations (Layer 3) - Contains a set of base protocol operations.
- Content (Layer 4) - Contains data of the device.

To specify data models, the YANG data model language ([Bjö10]) was created. It covers the operations and content layer.

Transport protocols need to provide reliable and sequenced data delivery connections and automatically release requested resources on connection close. Moreover, authentication, data integrity, confidentiality, and replay protection must be provided. The protocol is responsible for establishing the connection between server and client. A NETCONF peer assumes that the authentication is handled by the protocol. Every NETCONF implementation must provide and support the SSH mapping ([Cul11]).

CHAPTER 3 Concept

An important prerequisite for quick reconfiguration of network devices is an always up-to-date model of the physical network topology. This information is needed to generate a new configuration for the real-time network TSN on demand. To find the best route through the network, it has been decided to include the following properties: cable length, speeds, and the material of the connection on the physical layer. An open and common protocol to detect connected devices on a device is LLDP. LLDP offers the ability, for devices to announce their information over the complete network until a router is separating it. However, this is not supported by most LLDP daemons. A daemon is sending information about the local and the remote port, but additional information needs to be gathered from the device itself. If all daemons would send their information to the whole network, every device would have a huge list of information, that needs to be processed by the device. If every device only gets information from neighbors, it is less work to update the information when the network topology is changing. To query the information, the most common LLDP daemon *lldpd* supports publishing its data to an SNMP daemon (for details, see Section 2.3).

The first step of generating a model of the network topology is to make a deep scan of the network to find all SNMP devices. With the result of the deep scan, it is possible to scan the found devices separately to get more detailed information. If these initialization steps are completed, the application will wait for notifications of SNMP devices on the network.

The deep scan is implemented in a way that it can scan a large network in a short time. When the scan is started, it sends SNMPv1 GET requests to all IPv4 addresses in the given range. Because the scan is using normal requests, a community needs to be specified. To keep the traffic small, the scan requests the MIB field *SNMPv2-MIB::sysDescr*. This field should be implemented by every device and contains only basic information. The request is sent to all given IPv4 addresses, with a delay of 10ms between requests.

The scan waits for responses and logs them. This is possible because SNMP is a stateless protocol. A normal SNMP scan would send a request to one device and wait for the responses. If an answer is received or a timeout reached, the next request to the next device is being sent. The normal approach can take up to 13 minutes depending on the size of the network. Due to the implementation of the deep scan, it is only possible to scan 65535 IPv4 addresses with one scan. The bandwidth needed for this scan is less than 100MB/s. An alternative to the deep scan implementation would be to perform a depth-first or breadth-first search with the following steps:

- Get the neighbors found by LLDP on the scanning device, contact them per SNMP, and get their LLDP neighbors.
- Continue with the newly discovered neighbors until the whole network is known.

The speed of this scan would depend on the structure and size of the network. It would require contacting each device at the initialization phase one by one and parsing the information from each device after each scan. This could result in bad performance on large networks.

Every device found by the deep scan is scanned again by a SNMP v2c WALK request. These tables are requested to get more detailed information:

- *LLDPMIB::lldpLoc* - Contains LLDP information about the local device.
- *LLDPMIB::lldpRem* - Contains LLDP information about all connected devices.
- *IFMIB::if* - Contains information about all network interface of the local device.

After each SNMP walk, the data is processed and inserted into a database. To pair a connection of a local and a remote port, the MAC-Addresses of these ports are used. It is required that the data of both ports have been queried. The remote MAC address can be found at *LLDPMIB::lldpRemChassisId* if *LLDPMIB::lldpRemChassisIdSubtype* is set to a physical address. The local MAC address can be found at *IFMIB::ifPhysAddress*.

The last step is to wait for incoming SNMPv1 traps, which get triggered by devices if the status of an interfaces changes. If such a trap is detected, the sender of the trap is scanned by an SNMPv2c WALK request again. The queried data is parsed and the found structure is saved again.

CHAPTER 4

Implementation

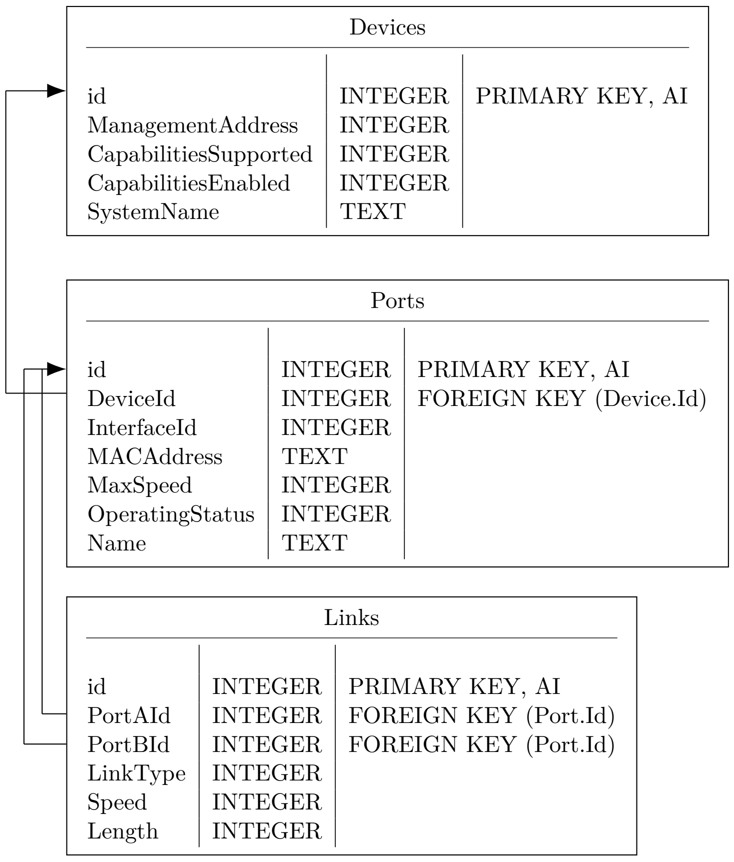
The implementation follows the concept in the previous chapter, a detailed flow diagram of the application can be seen in Figure 4.2. To be able to receive a trap while the application is running, a separate thread is needed. This can be seen in the figure, where the flow splits into two separate paths. If a terminating signal is received, both threads will get terminated.

The application is using snmpwalk to query data from devices. There is an open-source implementation of it by the net-snmp project. It is compiled by the build system and executed by *exec* from the application. Its output is parsed by the application. A more clean way would be using a library and not executing a precompiled executable. A common library for that is net-snmp, but due to the broad platform and version support, it is very complex. For a first prototype, the precompiled executable has been used. This implementation will deliver the queried data as a string over stdout.

For the deep scan, the application is using onesixtyone [1]. It is the most common open-source implementation for such a task. With it, it is possible to scan the network for SNMP devices in a very short time span. This program is precompiled, executed with *exec* and the output parsed by the application.

Due to the usage of a precompiled version of *snmpwalk*, the application needs to parse a lot of strings. Because string operations are limited in C, it was decided to use a dedicated library. The library sds has been selected for the application. Sds manages the memory allocation for the strings and is compatible with default C strings. The library also offers multiple methods, that are faster than the corresponding C string functions. It also offers functions that provide more comfort, e.g. tokenize a string with a specific delimiter. When using sds methods, the developer needs to be aware that the methods need to be used in other ways than default c *string.h* methods.

[1] https://github.com/trailofbits/onesixtyone



Devices		
id	INTEGER	PRIMARY KEY, AI
ManagementAddress	INTEGER	
CapabilitiesSupported	INTEGER	
CapabilitiesEnabled	INTEGER	
SystemName	TEXT	

Ports		
id	INTEGER	PRIMARY KEY, AI
DeviceId	INTEGER	FOREIGN KEY (Device.Id)
InterfaceId	INTEGER	
MACAddress	TEXT	
MaxSpeed	INTEGER	
OperatingStatus	INTEGER	
Name	TEXT	

Links		
id	INTEGER	PRIMARY KEY, AI
PortAId	INTEGER	FOREIGN KEY (Port.Id)
PortBId	INTEGER	FOREIGN KEY (Port.Id)
LinkType	INTEGER	
Speed	INTEGER	
Length	INTEGER	

Figure 4.1: Database schema visualization of the application.

To save the queried data in the structure needed, the application uses sqlite3 [2]. Sqlite was selected because it is easy to use with C. The sqlite3 library has a simple API to create a database and execute SQL statements. With SQL it is possible to easily query data, with specific and complex conditions. Linking entries with others is possible, which makes it easy to represent connections. Sqlite has good portability, which means other applications can interact with the database easily. Data of a Sqlite database can easily be visualized using existing tools. This makes it easy to use in the development process and can be helpful with debugging. In Figure 4.1 the database schema that has been used can be seen.

[2]https://www.sqlite.org/index.html

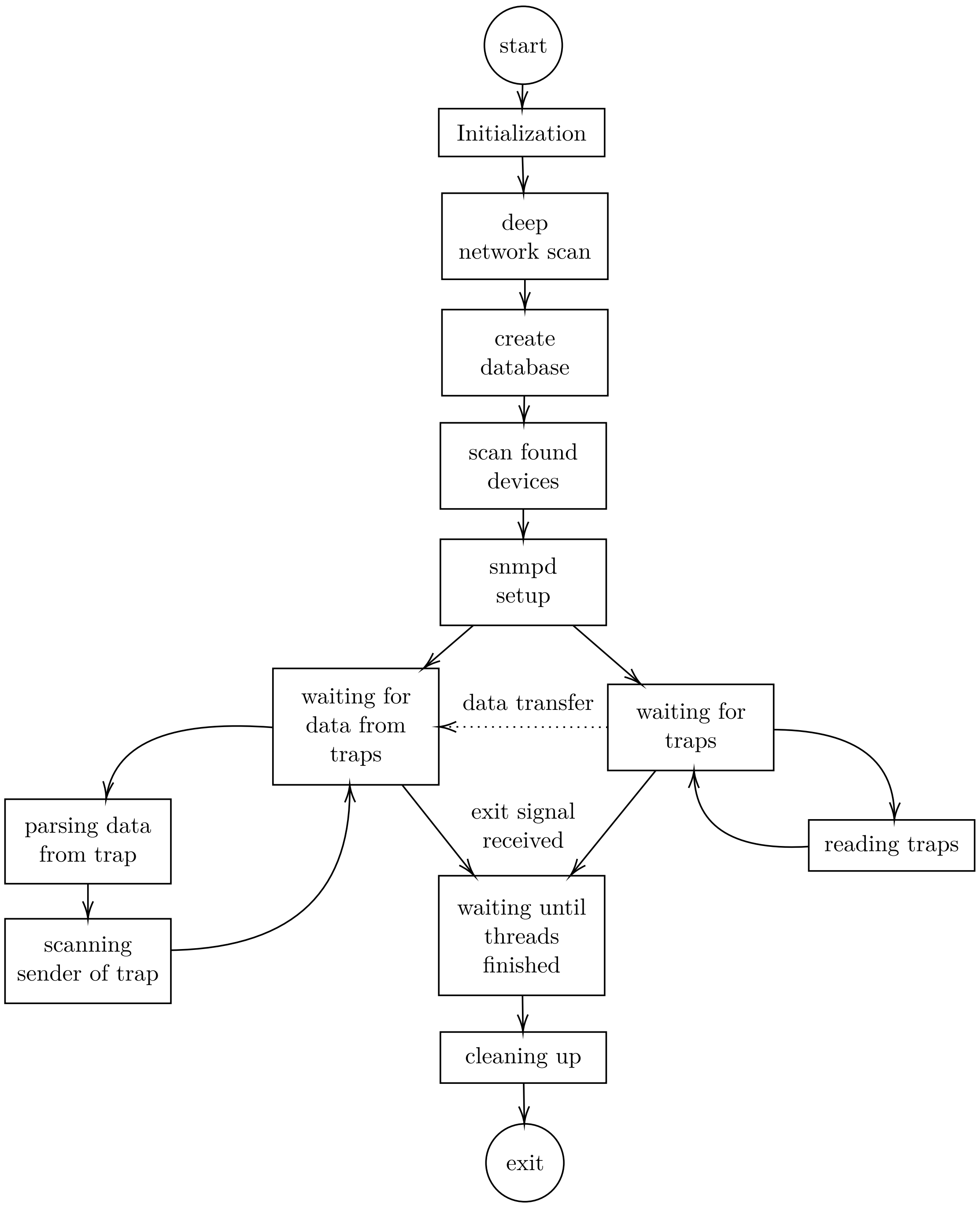


Figure 4.2: Program flow of the application.

CHAPTER 5

Evaluation

To visualize the scenario when a network device is added or removed, Figure 5.1 shows an example network. A similar network has been used in the evaluation process of the application. This network contains only one switch with IPv4 address 10.0.0.2, as well as four additional devices. While the device with IPv4 address 10.0.0.1 holds the application, the other devices do not have a task related to the network discovery. Every device on this network has an SNMP daemon running and only the switch and the application device have configured traps. These traps will notify the application if a connection changes.

The scenario of removing a device from the network can be seen in Figure 5.2. Here, the connection between the switch and the device with IPv4 10.0.0.4 got removed. Because of that, the switch will send a trap to the application. The application parses the received trap and initializes an SNMP scan of the switch. The received scan indicates that the device with IPv4 10.0.0.4 got removed and it will be removed from the internal database.

In Figure 5.3 the event of adding a device can be seen. For this case, a new device with IPv4 10.0.0.6 got added to the figure and a connection between it and the switch got added. The switch detects the connection and sends a trap to the application. The application initializes an SNMP scan of the switch. After parsing the result of the scan, the application detects that a new device got added to the network. It will now make an SNMP scan of the device with the IPv4 10.0.0.6 and add it and the new information of the switch to the database.

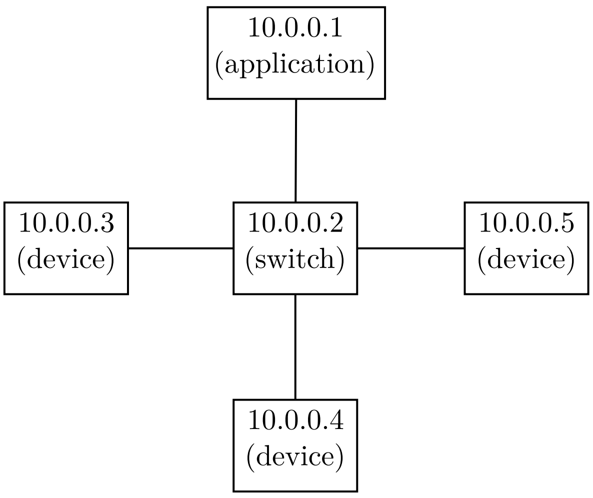


Figure 5.1: The application device has the IP-Address 10.0.0.1

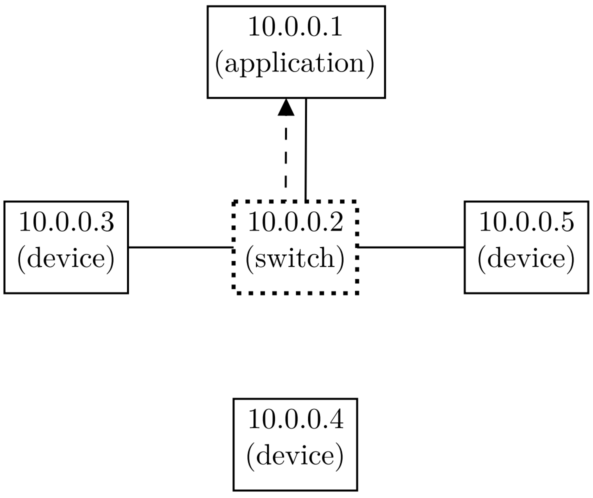


Figure 5.2: Communication is marked as an arrow, scanned devices are highlighted.

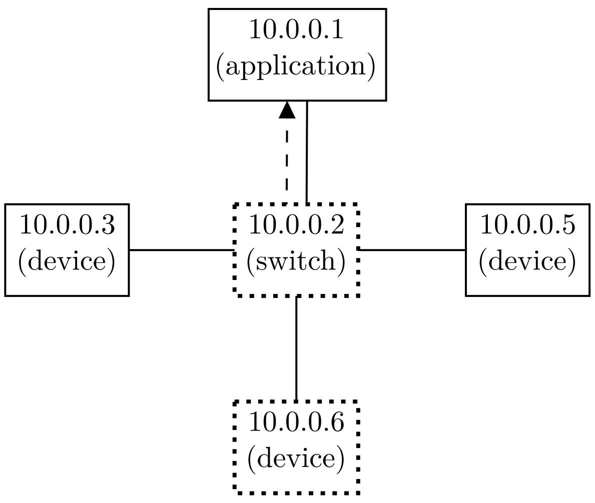


Figure 5.3: Communication is marked as an arrow, scanned devices are highlighted.

5.1 How to setup a device

To make a Linux device work with this application, it is necessary to install an LLDP and SNMP daemon and configure them. More detailed information can be found in the man pages of snmpd, lldpd and lldpdcli: [MAN21] [RB21] [Ber21]. The setup was tested on Debian 10 and may vary depending on the Linux distribution. The first step is to install *lldpd* and *snmpd*. After the installation has been finished, *snmpd* needs to be configured. All configuration needs to be done with superuser permissions. For that, a file needs to be created at */etc/snmp/snmpd.conf* with the content shown in Listing 5.1.

```
# SNMP v3 read-only community "public"
rouser public noauth

# SNMP v1/v2c read-only community "public"
rocommunity public

proc 0

# SNMP device info
syslocation "closet"
syscontact "Alice"
sysservices 12

# Activate agentx
master agentx
```

Listing 5.1: snmpd.conf - Configuration file of the SNMP daemon

With this configuration, the SNMP daemon will listen to SNMP v1/v2c/V3 requests, when community "public" is used. Entries can only be read and not modified. For SNMPv3 no authentication is needed. The location of the system is "closet", the system administrator is called "Alice" and it should provide information about the system like interface names. It also commands the daemon to activate *agentx*, which allows other daemons like *lldpd* to provide its data over SNMP.

The next step is to configure *lldpd* via the command line interface *lldpcli.* If *lldpcli* is called, a special shell will open. After configuration is finished, the shell can be closed with the command *exit.* The commands needed to configure the LLDP daemon are provided in Listing 5.2.

```
configure lldp portidsubtype macaddress
configure system bond-slave-src-mac-type real
configure lldp capabilities-advertisements
configure lldp management-addresses-advertisements
configure lldp agent-type nearest-bridge
```

Listing 5.2: Commands to configure lldpd with lldpcli

These commands will configure the daemon to publish the MAC address of the port as portid, use the real MAC address of the portadvertise the system capabilities, advertise the management address of the device, and only broadcast the information until the next bridge.

Now both daemons need to be started, for *snmpd* it only needs one command (See Listing 5.3).

```
$ snmpd
```

Listing 5.3: Starting snmpd.

The LLDP daemon, on the other hand, needs to be launched with the *-x* as an argument, this will notify the daemon to search for an agentx and communicate with it. There are two ways to start the daemon, the first way is to start *lldp* directly with the argument (see Listing 5.4).

```
$ lldpd -x
```

Listing 5.4: starting lldpd with agentx support.

Using the lldp daemon as a service requires a configuration file. This line needs to be added to */etc/default/lldpd*:

```
DAEMON_ARGS="-x"
```

Listing 5.5: /etc/default/lldpd - Starting the daemon with agent x support.

After adding the line, the daemon can be started with:

```
$ lldpd
```

Listing 5.6: Starting lldpd

5.2 Requirements

The application has some requirements to deliver the correct network topology. Every device on the network must fulfill these requirements:

- Run an SNMP daemon.
- Support SNMPv1 and SNMPv2c.
- Run an LLDP daemon.
- LLDP data must be collectible with SNMP.
- Each interface port needs to have a unique MAC address.
- LLDPd must send the port MAC address as *ChassisID*.
- The remote port data must be connected to a local port in the MIB data.
- Every network node, that isn't an end node, must send an SNMPv1 trap to the application if the status of an interface changes.

CHAPTER 6

Conclusion and Future Work

Due to the many different used technologies, with their manufacturer-specific implementations, a lot of issues have occurred. The issues and their solution, if there have been any, are presented in this section. There are also possible improvements for future implementations depending on new technologies or modified versions of the used ones.

6.1 Issues

If two devices from the same model and vendor have different firmware versions, it can happen that the devices return different types of data at the same MIB entry. To avoid this problem, the devices should have the same firmware and package versions.

Multiple OIDs can point to the same MIB entry. It is common that every manufacturer has its MIB table beside the standardized ones from IEEE. This makes it hard to find the needed MIB entries. A common way to find the needed entry is to perform an SNMP walk of the complete MIB table of the device. If the values of the entries are known, the entries can be found that way. The project Net-SNMP provides an application *snmptranslate* to convert between numbered and labeled OIDs. This translation process can also help to find the correct entry.

By default, MIB entries can have different access permissions, depending on the configuration of the device. A recommended way to fix such issues is to create an *SNMP view* on each device. The view defines which community can view what MIB entries.

Because manufacturers implement MIB tables differently, on some devices it was possible to get the MAC address of the remote port. But it was not possible to identify which local port received this information. All received information was assigned to local port zero, which is representing the bridge and not a specific port.

Some MIB entries of some manufacturers included non-printable characters, which represent different control characters in ASCII. Especially the entry *SNMPv2-MIB::sysDescr* included them, which caused errors when parsing the output of *snmpwalk*.

The application needs SNMPv1 for the deep scan and traps and needs SNMPv2c for scanning devices. Both protocols need to be activated. On some devices, it is required to activate the protocol version separately. Others activate all versions if SNMP is enabled.

To match the connection between ports, each port needs to have an individual MAC address. But some devices only provide one MAC address for all ports.

One device returned a MAC address for a remote port, that could not be found on the device, neither in the command line interface nor via SNMP. With this behavior, it was not possible to match the local and remote ports.

There is no default way to get the length of a connection between two devices. The reason for this is that there is no application in normal Ethernet networks that would require the length of the line. It is possible to get the length of a cable by using the Precision Time Protocol (PTP). It is used to precisely synchronize time between multiple network devices. If a precision time measurement is available between devices, it can be used to measure how long a message took from one device to another. Because the speed of electrical signals in different materials is known, the distance the signal traveled can be calculated. But while researching implementations of it, no daemon has been found that hand this data to the SNMP daemon.

To calculate better routes, the material of the transmission medium is also needed. The reason for that is the difference in latency from the material that is used for a connection. The most common materials are fiber optical and twisted-pair copper for wired connections. There is no standardized way to get this information from the devices. This is due to that the devices often do not have this information. There are MIB tables to get this information from some devices, but they are proprietary and only for specific devices.

To update information on changes on the network, the application is relying on SNMPv1 traps. All nodes of the network that are able to have more than one connection need to send them to the application. In the current state of the application, these traps need to be configured manually. Every device manufacturer has other methods and schemas to configure them and they are often also not well documented.

6.2 Conclusion

In theory, it is possible to implement network discovery on the basis of LLDP and SNMP but in practice, there have been a lot of pitfalls, which ultimately prohibited a full test of the application with existing equipment. A detailed explanation of the issues can be found in Section 6.1.

Most problems occurred, with the provided data by the manufacturer. This includes missing data, wrong data format, and more. Another problem has been the usage of SNMP because it was very complicated to map the data between the data type register and the data register. Under the devices that have been used for testing the application, no network device could be found that fulfills all requirements defined in Section 5.2.

The current state of the implementation is that the data is queried from the devices and it can also receive traps. But because of the occurred issues, the data from the single network devices can't be connected. This means that the data of the devices has been written to the database, but the connection table stays empty.

There also has no way been found, to get the length of the connection and the connection type. There are technical ways to implement such features, but this must be supported by the device manufacturers. If device manufacturers would support a discovery mechanism like described in this thesis, it could be an advantage to create a MIB table for this usage. In this MIB table, all the needed data could be described, which would also speed up the data gathering process.

Most needed data could be gathered with SNMP, but some essential data to connect the found information couldn't be gathered. SNMP provided a lot of pitfalls while implementing the application. From that experience, a future implementation should use NETCONF instead of SNMP. A detailed explanation of possible improvements can be found in Section 6.3.

6.3 Improvements

It is possible to improve the speed of the initial deep scan, by reducing the interval packages are sent. This would require a network with bigger bandwidth capabilities and also a device that can handle this amount of traffic. Another way would be increasing the interval, which would reduce the bandwidth needed and allow to increase the number of devices that can be scanned. It would take longer to scan the whole network, but it enables to cover larger networks.

The current implementation uses snmpwalk as a method to query the needed data. Each needed MIB table is requested by a snmpwalk, to reduce the time needed for the operation. There is an advanced implementation called *snmpbulkwalk*, which uses the SNMP BULKWALK request to gather the data. It is faster by bundling the requested data in one request. With *snmpbulkwalk*, it may be faster to request a higher OID where all needed MIB tables are included. The speed improvement depends on the OID of these MIB tables and their sizes.

Instead of using the precompiled *snmpwalk* executable, it would be better to use the SNMP library provided by the net-snmp project. It makes the requests more flexible and could make it possible to parallelize the snmpbulkwalk. This may increase the speed of the initialization phase of the application significantly.

The communication to SNMP devices is not secure as SNMPv2c is used for the application. In some networks, it could be a requirement to secure the connections. This is possible with SNMPv3. SNMPv3 also allows to set different permissions per user. In combination with authentication, this could restrict the publicly available information. The last possible security measurement is transport security, which allows to create a secure connection or session between two points. However, this security measurement may introduce a new complexity that makes it harder to maintain the network.

IPv6, the successor of IPv4, is getting more popular, the current state of the application does not support IPv6. With IPv6 it would be possible to assign one IPv6 address per port. Then it is possible to use the IPv6 Neighbor Discovery Protocol [SNNS07]. This protocol is similar to LLDP (see Section 2.3) and may be able to replace it.

Because SNMP was designed in 1988 and uses MIB for data representation, it does not follow modern implementation models. Fields that can have data represented in multiple ways need a separate field that saves the representation form. This makes it harder to parse data queried by SNMP. NETCONF, the successor of SNMP, uses YANG as a data modeling language. In YANG, data is represented as objects, whereby each property of an object has a fixed data type assigned. Another advantage is, interrelations between different data points can be represented. Using NETCONF and YANG would reduce the work needed to implement data parsing and interoperability between programs, because not every MIB entry has to be linked manually. YANG models can be shared between applications with JSON or XML. Both file formats are widely used for transferring information.

Acronyms and Abbreviations

ASCII - American Standard Code for Information Interchange
ASN.1 - Abstract Syntax Notation One
DOCSIS - Data Over Cable Service Interface Specification
IEEE - Institute of Electrical and Electronics Engineers
IP - Internet Protocol
IPv4 - Internet Protocol Version 4
IPv6 - Internet Protocol Version 6
LLDP - Link Layer Discovery Protocol
LLDPDU - LLDP Data Unit
MAC - Media Access Control
MB/s - Megabytes per second
MIB - Management Information Base
NETCONF - Network Configuration Protocol
PDU - Protocol Data Units
PTP - Precision Time Protocol
RFC - Request for Comments
SNMP - Simple Network Management Protocol
SSH - Secure Shell
SQL - Structured Query Language
TLV - Type-length-value
TSN - Time-Sensitive Networking
OID - Object Identifier
OSI - Open Systems Interconnection
UDP - User Datagram Protocol
VLAN - Virtual Local Area Network
XML - Extensible Markup Language
YANG - Yet Another Next Generation

List of Figures

List of Tables

Bibliography

[Ber21] Vincent Berna. LLDPCLI(8): lldpcli, lldpctl - control LLDP daemon, November 2021.

[Bjö10] Martin Björklund. YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF). RFC 6020, October 2010.

[Chr01] Philip Christian. Generic Routing Encapsulation over CLNS Networks. RFC 3147, July 2001.

[Cul11] Margaret Cullen. Using the NETCONF Protocol over Secure Shell (SSH). RFC 6242, June 2011.

[EBBS11] Rob Enns, Martin Björklund, Andy Bierman, and Jürgen Schönwälder. Network Configuration Protocol (NETCONF). RFC 6241, June 2011.

[Edi02] RFC Editor. Assigned Numbers: RFC 1700 is Replaced by an On-line Database. RFC 3232, January 2002.

[Enn06] Rob Enns. NETCONF Configuration Protocol. RFC 4741, December 2006.

[FSDC90] Mark Fedor, Martin Lee Schoffstall, James R. Davin, and Dr. Jeff D. Case. Simple Network Management Protocol (SNMP). RFC 1157, May 1990.

[FWLR00] Rob Frye, Bert Wijnen, David B. Levi, and Shawn A. Routhier. Coexistence between Version 1, Version 2, and Version 3 of the Internet-standard Network Management Framework. RFC 2576, March 2000.

[HC19] Maki K. Habib and Chukwuemeka Chimsom. Industry 4.0: Sustainability and Design Principles. In *2019 20th International Conference on Research and Education in Mechatronics (REM)*. IEEE, may 2019.

[HWP02] David Harrington, Bert Wijnen, and Randy Presuhn. An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks. RFC 3411, December 2002.

[IEE16] IEEE Standard for Local and metropolitan area networks - Station and Media Access Control Connectivity Discovery, 2016.

[ISO15] ISO/IEC 8824-1:2015, Information technology Abstract Syntax Notation One (ASN.1): Specification of basic notation, November 2015.

[MAN21] SNMPD.CONF(5): snmpd.conf - configuration file for the Net-SNMP SNMP agent, November 2021.

[MK00] Keith McCloghrie and Frank Kastenholz. The Interfaces Group MIB. RFC 2863, June 2000.

[Mü06] Stefan Müller. SNMP MIB-Tree. Website, June 2006.

[Pre02a] Randy Presuhn. Management Information Base (MIB) for the Simple Network Management Protocol (SNMP). RFC 3418, December 2002.

[Pre02b] Randy Presuhn. Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP). RFC 3416, December 2002.

[RB21] Pierre-Yves Ritschard and Vincent Bernat. LLDPD(8): lldpd - LLDP daemon, November 2021.

[RM90a] Dr. Marshall T. Rose and Keith McCloghrie. Management Information Base for network management of TCP/IP-based internets. RFC 1156, May 1990.

[RM90b] Dr. Marshall T. Rose and Keith McCloghrie. Structure and identification of management information for TCP/IP-based internets. RFC 1155, May 1990.

[RM91] Dr. Marshall T. Rose and Keith McCloghrie. Management Information Base for Network Management of TCP/IP-based internets: MIB-II. RFC 1213, March 1991.

[SH09] Jürgen Schönwälder and David Harrington. Transport Subsystem for the Simple Network Management Protocol (SNMP). RFC 5590, June 2009.

[SJ06] Jürgen Schönwälder and Tony Jeffree. Simple Network Management Protocol (SNMP) over IEEE 802 Networks. RFC 4789, November 2006.

[SNNS07] William A. Simpson, Dr. Thomas Narten, Erik Nordmark, and Hesham Soliman. Neighbor Discovery for IP version 6 (IPv6). RFC 4861, September 2007.

[Yer03] François Yergeau. UTF-8, a transformation format of ISO 10646. RFC 3629, November 2003.