



NATIONAL KAPODISTRIAN UNIVERSITY OF  
ATHENS

SCHOOL OF SCIENCE

DEPARTEMENT OF PHYSIC

DEPARTEMENT OF INFORMATICS AND  
TELECOMMUNICATIONS

**Postgraduate program of Department of Physics and Informatics  
and Telecommunications**

**MSc in Electronics and Radioelectrology**

**DISERTATION THESIS**

**“Control of Network Devices in 5G Communications Systems”**

**Kaponis Georgios**

**2020101**

## Table of Contents

<b>1. Introduction</b> .....	5
<b>2. Definition of the Problem</b> .....	6
<b>2.1 Software Defined Networking</b> .....	6
<b>2.2 Field Programmable Gate Array</b> .....	8
<b>2.3 Communication and Interface with SoC</b> .....	10
<b>2.4 Architecture</b> .....	11
<b>3. Control of Network elements through different Protocols</b> .....	13
<b>3.1 SNMP</b> .....	13
<b>3.2 SCPI</b> .....	22
<b>3.3 TL1 (Transaction Language 1)</b> .....	26
<b>3.4 NETCONF</b> .....	33
<b>3.4.1 NETCONF Data Modeling and Operations</b> .....	34
<b>3.4.2 Advantages of NETCONF and Use Cases</b> .....	38
<b>3.5 OpenFlow</b> .....	41
<b>3.5.1 OpenFlow Architecture and Operations</b> .....	43
<b>3.5.2 Advantages and presence SDN ecosystem</b> .....	48
<b>3.5.3 Practical Implementations of OpenFlow and Challenges</b> .....	50
<b>4. The Interface Developed</b> .....	53
<b>4.1 Methodology</b> .....	53
<b>4.1.1 Interface Design and Implementation</b> .....	56
<b>4.1.2 Testing and Validation</b> .....	59
<b>5. Conclusions and Future work</b> .....	60
<b>6. List of figures</b> .....	62
<b>7. List of Tables</b> .....	63
<b>8. References</b> .....	64
<b>Appendix A: Interface script code</b> .....	66

## **Abstract**

Control of Network elements is an essential idea in networking, especially for the 5G and 6G communication systems. Through the decades the scientific community has researched protocols and ways to remote control and manage network devices such as routers, switches even instruments. By taking advantage of such protocols, the network engineer is able to decide how the network can be designed and controlled.

The objective of this MSc thesis is to design, develop, implement and eventually evaluate a network interface capable of integrating a 6G Thz Node into a communication system that has been designed according to a specific paradigm.

First of all, a general presentation of the challenge at hand is presented. Next, the FPGAs and network paradigms used in the present are being addressed. In addition, a literature review of the network protocols for managing and controlling network devices is presented in detail

In the latest chapters, a presentation of the solution this research has proposed in order to integrate a Thz Node into an existing communication system. The way of thinking, the problem breakdown analysis and the script proposed are addressed.

Lastly, conclusions and evaluation of the integration plan of this thesis are presented as well as future work proposed on top of the integration performed.

## Prologue

This Thesis is presented in the context of acquiring the diploma of MSc in Electronics and Radioelectrology. A master provided by National Kapodistrian University of Athens, school of science, departments of Physic and Informatics.

This Thesis could not be devised without the constant guidance of professors A. Tzanakaki and M. Anastosopoulos. I would like to express my sincerest gratitude for giving me the opportunity to participate in something so fascinating, but also thank them for entrusting me with this project. The motivation provided by the two in order to devise this thesis, but also believe in my self to achieve something like this. Special thanks to the PhD candidate I. Floudas, first of all as a professional and mentor, as well as a friend who stood beside me from the beginning of this Master.

Last but not least I couldn't forget to thank my dearest family, friends and fiancé that helped me, motivated me and supported me through all these years.

Athens, January 2024

Georgios Kaponis

# 1. Introduction

The digital revolution we are experiencing as well as the one foreseen is associated with multiple applications available to common people, industries and governments. These applications depending on the function they serve come up with a different set of requirements as far as bandwidth, connectivity, mobility and latency are concerned. The 5G revolution has already triggered the fantasy of scientific groups in search of the next best technology. 6G development as well as the increase of the requirements of 5G, in order to achieve functions as smart cities, augmented reality, self-driven smart cars and very low latency applications that will make surgeries happen on the other side of the globe, demand that the elements of the system will work in a way that it will not concern the end user. Traffic control, administration of fatal errors as well as other unexpected events must be handled. Except for the unexpected events, the whole flow of information and traffic must happen in a way that each system element doesn't become an obstacle.

It becomes rather obvious that the control of each element in a way that cooperates efficiently with the other elements is crucial for an effective and fast system such as 5G. In order for this to happen, different protocols and interfaces are used in different layers as well. Different devices understand completely different commands and protocols and in order to make these cooperate interfaces must be developed. It is also crucial to integrate all these protocols and element in a system that the frontend and backend user doesn't bother on how this information is guided and routed between all these different layers and devices.

The first chapter of this essay will present the main problem of controlling a specific device very important for the implementation of mm Waves in a communication system. Chapter two will present the architecture of controlling this type of devices and the principles and main ideas of this architecture. We will also discuss the different layers in which we dive into in order to control and integrate. The next chapter will review the set of protocols used in networks in a way that is understood how the different devices are controlled. Chapter four will be about presenting the solution found in the lab about the device described in the first chapter. This control device is needed to implement the mm Waves in our communication system and we will present how it is managed, controlled and integrated in our communication system. Lastly, we will summarize the knowledge earned from all the research process and propose some future work.

## 2. Definition of the Problem

Before elaborating on the problem itself it is mandatory to refer to some technologies that influenced 5G and will play a significant role in 6G as well. The key technologies that this essay will refer to are the Software Defined Networking and the FPGAs (Field Programmable Gate Array).

### 2.1 Software Defined Networking

The era of 5G and the breakthrough that came with it, apart from the physical layer innovation and techniques, is mostly described by the implementation of Software Defined Networking (SDN). According to this technology network management can be dynamic. The changes in the network management aim to program the network configuration in a way that is efficient and improves the performance and monitoring. The difference between SDN and traditional network management is the key that made SDN network configuration more akin to cloud computing[1],[2]. As mentioned in Wikipedia, “SDN is meant to address the static architecture of traditional networks and may be employed to centralize network intelligence in one network component by disassociating the forwarding process of network packets (data plane) from the routing process (control plane). The control plane consists of one or more controllers, which are considered the brains of the SDN network, where the whole intelligence is incorporated”. The dynamic environment of SDN enabled the use of cloud computing in a way that the data plane and the control plane are separated and the infrastructure becomes software – programmable. This means that we can redirect the traffic in a way that is beneficial to us without having to replace the hardware of the network[3]. Off course the network traffic is not the only parameter that can be managed in this dynamic programmable environment. The SDN controller is an application inside the architecture of SDN that is responsible for the functions of network orchestration, management, analytics and automation. This controller runs on a server and uses protocols to tell the switches where to send the packets. The standards and API’s of the SDN controllers are open so that different equipment from different vendors can be controlled. The innovation of the implementation of SDN in the network management brought many benefits in business and cost efficiency bringing up more attention to this technology.

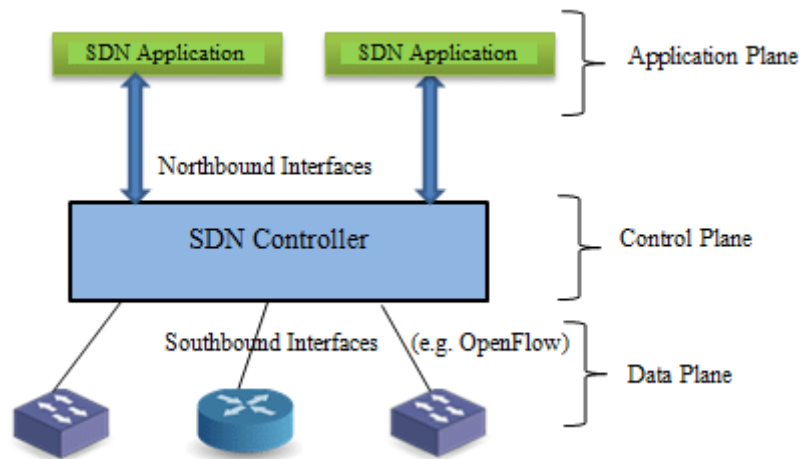


Figure 1 Simplified block diagram of SDN[4]

SDN is a paradigm that is very close to the term Network Functions Virtualization (NFV), which offers the ability to use virtually what was appliance dependent functions such as firewalls, load balancers and WAN accelerators. SDN becomes the orchestrator that uses efficiently all the virtual functions without having to dedicate these resources to one place. As far as the interfaces and protocols are concerned, literature review and elaboration on the matter will be presented in next chapters of this essay.

In conclusion, the SDN paradigm enables the remote programming of the network elements and resource allocation of functions. The paradigm as described above was taken into consideration in order to develop a way to control an FPGA remotely. The efficient control of the FPGA mentioned above will redirect the traffic in such way that the use of mm Waves can be achieved whenever they are available and needed.

## 2.2 Field Programmable Gate Array

The FPGA is an integrated circuit which allows the user to program it and reprogram it in the way that is most suitable for him after the manufacturing procedure[5]. This



*Figure 2 Xilinx FPGA[7]*

circuit consists of an array of programmable logic block and interconnects that can be reconfigured in order to perform various digital functions. The versatility, flexibility and speed of this kind of circuits as well as the ability of parallel processing make them ideal for applications in the field of telecommunications.

FPGAs are programmed using a hardware description language (HDL). In order to configure this kind of circuit a certain expertise on this kind of language is needed from the end user. In addition this kind of circuits employ designs with very fast I/O data rates and bidirectional buses. This kind of data rates require a certain amount of caution as far as timing is concerned. The valid data must be transmitted within the valid time windows between setup and hold time. It is safe to say that even though this kind of hardware offers great advantages, it comes with some disadvantages that can't be ignored.

The reason engineers and system designers choose FPGAs is the flexibility offered. However the tradeoff of this flexibility comes with the difficulty of implementing this kind of circuit into a greater system. FPGAs do not have the driver ecosystem and code/IP base that microprocessor architectures and OSs do. In addition, microprocessors coupled with OSs provide the foundation for file structures and communication to peripherals used for many, often essential, tasks such as logging data to disk. In order to implement this kind of hardware into a telecommunication system a hybrid architecture is advised. a microprocessor is paired with an FPGA that is then connected to I/O. Modern FPGAs often combine logic gates with processors



into a single chip called a System on Chip (SoC) for increased computing performance. This approach takes advantage of the benefits that both these targets offer.

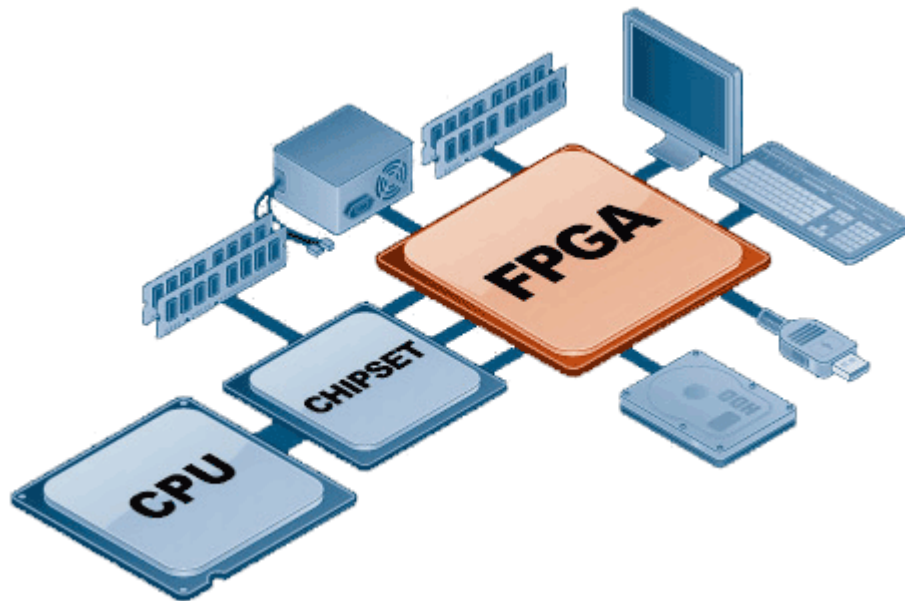


Figure 3 FPGA architecture

In order to design FPGAs, a set of tools are recruited. Two main categories are the traditional tools and the high-level synthesis design tools. In the context of this essay neither will be elaborated; however, both of them include specific skill set or “tool-box” such as labview.

As far as telecommunications are concerned, 5G and 6G especially, pure usage of FPGAs is not advised. On the contrary a heterogeneous architecture representing a System On Chip (SoC) that has the ability to be reconfigured and addressed by a processor is highly advised[6]. Following this paradigm and the SDN architecture the IHP institute, Radio Frequency & Broadband Communication System division developed a SoC that runs an OS making it able to communicate with serial ports. This was the starting point of the present essay’s research activity.

## 2.3 Communication and Interface with SoC

Researchers of IHP institute developed a SoC with the capability of wireless transmitting and receiving in the frequencies of millimeter waves. This specific frequency range is desired and adopted in the development of 6G. As far as the internal architecture and design of the SoC are concerned the research team in the lab doesn't have the ability to reprogram or intervene. However, the OS in the SoC has the ability to connect with ethernet or serial port on a server that will control the SoC. This SoC will provide the ability to implement a mm Wave transceiver, as long as we have the information of the status of the transceiver through the SoC. However, the lack of variety of communication commands with the SoC created the need to develop an interface capable of transferring the information. This part was the main theme of



*Figure 4 FPGA used for our research*

this research.

The main problem with this architecture is that the board was able to establish communication with the PC (server) with a very specific command and retrieve the parameters of the registers with a specific command with different parameters. This makes it unable to be directly controlled by an SDN controller. An interpreter or maybe better an interface had to be developed in order to establish connection with the

board and at the same time send the parameters to a broker that the SDN controller has direct communication.

Another problem that this research had to overcome was that the OS of the board had to maintain intact in order to continue to stay active. As a result, we had to be very careful in the lab not to send a wrong command or bit stream towards the board that might make it unserviceable. Specific instructions addressing the on and off as well as the tracing of this board were given in order not to create a mismatch or a software erase. In better words, trial and error was not an option.

Last but not least, the SDN controllers and APIs use a specific set of protocols that cannot establish communication or transmit bitstreams to the board. This board will be part of telecommunication system and has to conform with the 5G and 6G demands of SDN and VFN, as well as the protocols used for such paradigms.

All these problems and drawbacks had to be taken into account in order to integrate such technology to a 5G and 6G system. Having all these in mind, the research created a script that could solve the issues and also leave room for more development of more functions for future use. In the next chapters the solution that was created is presented. The code used is also demonstrated at the Appendix A of this essay.

## 2.4 Architecture

SDN is considered revolutionary because of the way it separated the control plane from the data plane. In order to perform so different architectures have been designed; however, each one has the same principle of separating the control from the data plane. In addition, as referenced above, the architecture must be open in order

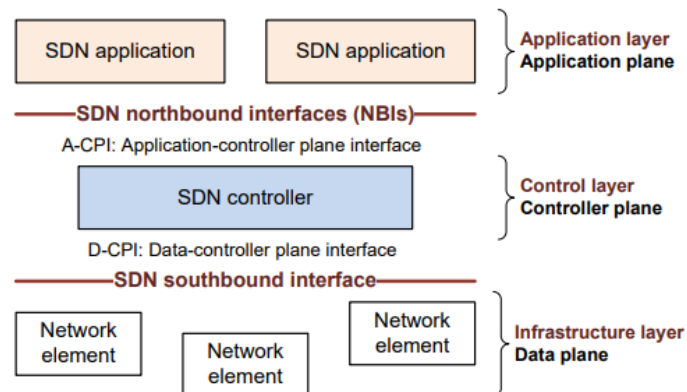


Figure 5 Basic SDN architecture[9]

for different equipment from different vendors can be used and integrated. In other words, the purpose of SDN is to provide open interfaces that enable the development of software specialized in controlling the network elements, the flow of the traffic, as well as redirecting the traffic after inspecting it[9].

In order to perform such a function SDN is separated in three layers as described in figure 5. The SDN controller, considered as the Control Layer, communicates with the

Application layer with Northbound interface and with the Data plane with the Southbound interface.

The infrastructure layer (data plane) includes all the network elements that present their capabilities towards the SDN controller (control layer) via the southbound interface. In “Software-Defined Networking: The New Norm for Networks” this interface is called control-data plane interface. The Southbound interfaces are responsible for delivering the status of the network infrastructure towards the controller giving it a complete picture of what resources are available. Important thing to notice is that this interface must also include the traffic forwarding and processing functions. Something as important as the fact that the data plane is responsible for the traffic monitoring and forwarding, is that the data plane can include the minimum subset of control and management functions.

The Northbound interface is responsible for the communication between the SDN applications and the SDN controller. The SDN applications reside in the application layer and communicate their network requirements towards the control plane via the Northbound interface. The SDN controller, as the man in the middle, receives the requirements from the application layer and the resources available from the data plane. Then proceeds to translates the higher-level requirements into low-level control towards the network elements, while giving feedback towards the SDN applications.

This simplified architecture depicts how the system should work. However, the need of management in each layer seemed to be mandatory. This is the reason why another architecture, based on the one referenced above, has been deployed and is pictured in figure 6.

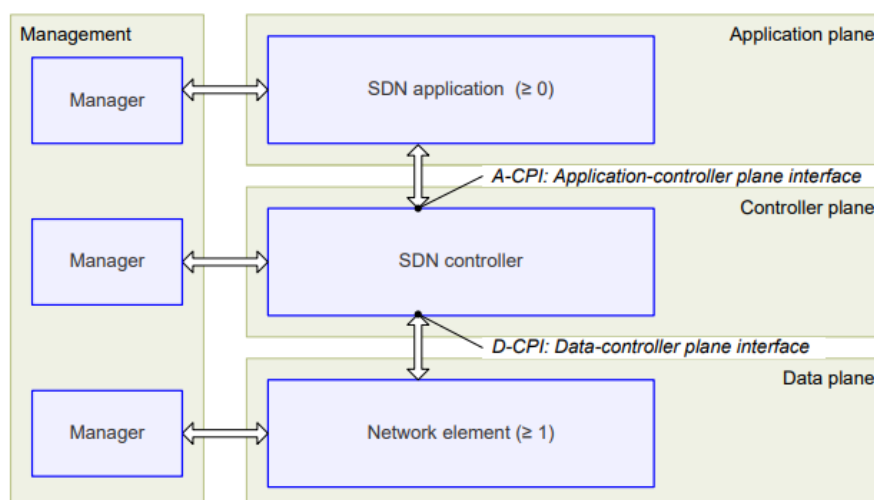


Figure 6 SDN overview[9]

According to the VNF that are needed to be established in order to set up a complete network the SDN has to adapt with managers and other interfaces, although the overview and the principals remain the same. For SDN controlled mmWave

transceivers interfaces in the southbound side must be developed in order to communicate with the open protocols such as openflow etc. The architecture this project has followed will be presented in chapter 4.

## 3. Control of Network elements through different Protocols

As already mentioned in previous chapters the SDN concept is adopted by 5G and 6G and quickly became a breakthrough because of the way it changed the static entity of network elements. As far as this essay is concerned more attention will be drawn towards the southbound interfaces.

The most common interface that dominated the modern telecommunication systems is openflow. Openflow has been a breakthrough and sometimes there is confusion as far as the meaning of SDN becomes equal to openflow. Truth is there are also other protocols for managing network elements that were used in the past but also still used up to this date due to the unique systems they control. After all 5G and 6G are standards that endorse IOT. Technology is very dynamic and the changes are rapid. Openflow is already considered “old” and new protocols and ways of communication between the Southbound interfaces are being studied and used in order to evolve the concept of SDN and VNF. Going through the literature and the information available in the world wide web we observed that many manufacturers move to VXLAN with a EVPN MP-BGP control plane. The revolution of Python also brought up changes in the way APIs are developed. However, knowledge of each technology and paradigm can help improve the way we handle the obstacles in developing and researching. In this chapter we will discuss about SNMP, SCPI, TL1, NetConf and off course extensively about openflow. This literature review will help the reader understand the take we had on tackling the difficulties of our research in integrating a 6G node for controlling a mmWave transceiver, in a deeper way and answer the question why we preferred the solution that was chosen.

### 3.1 SNMP

SNMP stands for “Simple Network Management Protocol” and is an Internet Standard protocol responsible for collecting and organizing information about managed devices on IP networks. This protocol is also responsible for modifying this information in order to change device behavior[10]. Most of the network devices such as routers and switches as well as cables support SNMP. This protocol uses a manager client architecture and is used widely in enterprise networks. Within SNMP networks, systems, components, and applications are described as entities. The number of entities that need to be managed is growing rapidly[11]. SNMP uses UDP port 161/162 in order to monitor and detect faults as well as reconfigure remotely some devices[12]. Requests and responses arrive at the 161 UDP port and notification on 162 [14].

SNMP manager or management system is a separate entity that is responsible to communicate with the SNMP agent implemented network devices. Usually, this is a computer that is used to run one or more network management systems. SNMP's manager key functions include sending queries towards the agents, getting responses from them, setting variables in the agents and acknowledging asynchronous events from them.

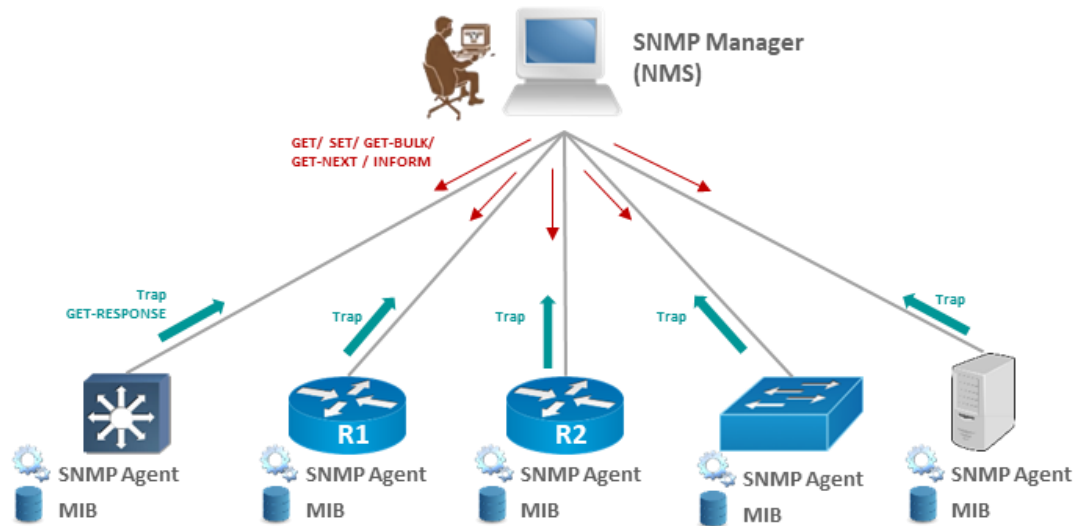


Figure 7 SNMP visualization [14]

On the other side we have the managed devices. A managed device or a network element, is a part of the network that requires some form of monitoring and management. For example this element might be routers, switches, servers, workstations printers etc. In order for the SNMP manager to communicate and/or control the network elements a program must be packaged within the network element itself. This program is called agent. Enabling the agent allows the manager to collect information about the device. The agent enables the management information database from the device locally and makes it available to the manager when is needed. The agent as a manager software in the device has local knowledge and is used as an interpreter between the device and the manager. Key functions of an agent include the collection of management information about its local environment, storage and retrieval of management information as defined in the MIB and signals of the events towards the manager. In some cases, when a network node is not SNMP manageable the agent acts as a proxy[13],[10].

In order to understand the way this protocol works we have to talk about OID and MIB. MIB stands for Management information base and is a collection of information organized hierarchically. SNMP accesses these databases in order to retrieve the OID. Scalar objects define a single object instance whereas tabular objects define multiple related object instances grouped in MIB tables. MIBs are collections of definitions which define the properties of the managed object within the device to be managed. The information stored in the MIB includes data such as system configuration details, performance statistics, and error messages. Devices that support SNMP have an agent software component that interacts with the MIB. The SNMP manager communicates

with the agent to query or set values in the MIB, enabling the monitoring and management of network devices.

There are various standard MIBs defined by organizations like the Internet Engineering Task Force (IETF) that cover general aspects of network management. Additionally, vendors often define their own private MIBs to provide information specific to their devices.

OIDs stands for Object identifiers. OIDs uniquely identify the managed objects in a MIB hierarchy. It can be depicted as a tree whose levels are assigned by different organizations.

OID stands for Object Identifier in the context of the Simple Network Management Protocol (SNMP). An Object Identifier is a unique identifier used to identify managed objects in the SNMP management information base (MIB). The MIB is a hierarchical tree-like structure that organizes information about devices on a network, making it accessible and manageable.

The OID is represented as a sequence of numbers separated by dots, similar to a hierarchical path. Each number in the sequence represents a node in the tree, and the complete OID uniquely identifies a specific variable or object within the MIB.

For example, the OID 1.3.6.1.2.1.1.1.0 might represent the system description in the SNMP MIB. Here's a breakdown of what each number in the OID might represent:

- 1: ISO (International Organization for Standardization)
- 3: Identified Organization
- 6: Internet
- 1: Private enterprises
- 2: IANA (Internet Assigned Numbers Authority)
- 1: SNMP MIBs
- 1: SNMP MIB-II
- 1: System
- 0: System description

In SNMP, the OID is crucial for uniquely identifying and retrieving information about specific variables or objects from network devices. It serves as a standardized way to reference and manage data in a network management system.

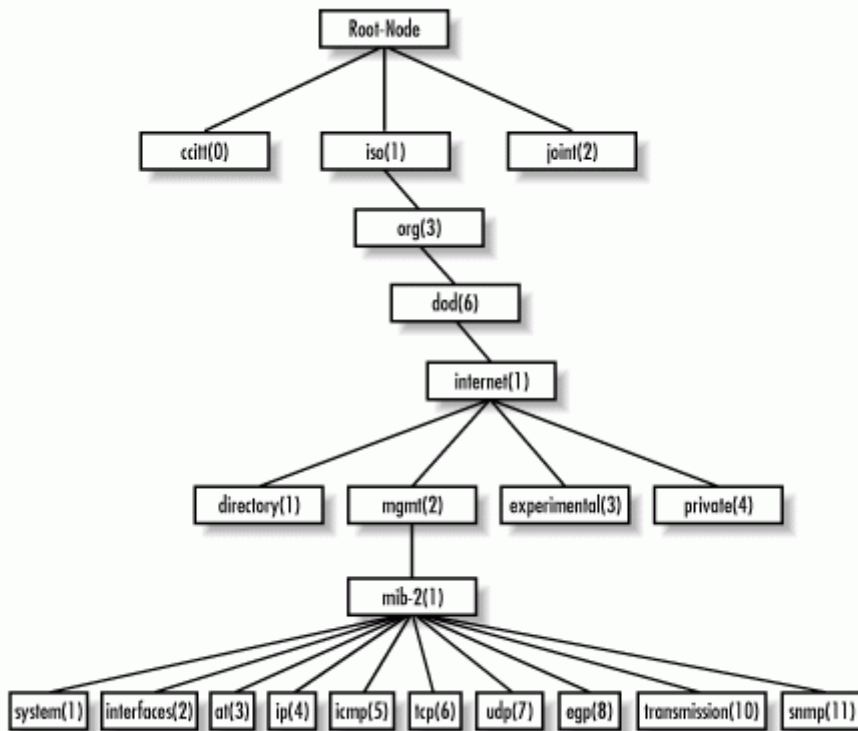


Figure 8 Standard MIB tree[15]

Scalar objects define a single object instance. Tabular objects define multiple related object instances that are grouped in MIB tables. SysDescr, for example, is a scalar object.

As far as the architecture is concerned, figure 9 is a simplified visualization of SNMP. SNMP is an application layer protocol that utilizes the UDP protocol on an IP network.

NMS is software that collects data from the devices, organizes it, and shows it to the end user. The ICONICS SNMP connector functions as a manager in an SNMP scenario.

## SNMP Architecture

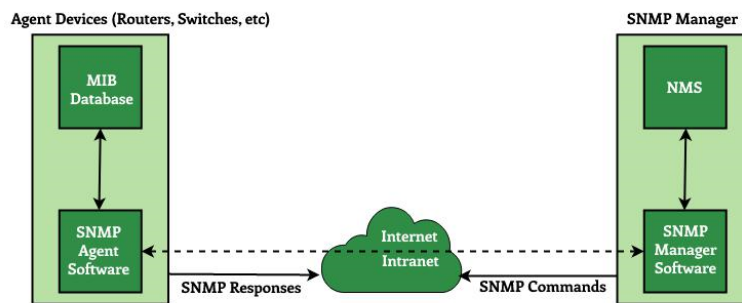


Figure 9 SNMP architecture[16]

An NMS would monitor or control managed devices that provide SNMP information



through the agents. An agent is software that runs on a device (such as a router, printer or PC) and answers to the messages from the NMS. These messages can either be read messages (NMS wants to retrieve data) or write messages (the manager wants to set data). The agent can also send a trap to the NMS. A trap is a notification similar to an alarm[17]. When a threshold is exceeded, the network element will not wait for the query of the SNMP manager. On the contrary it will send directly a trap notification giving that information.

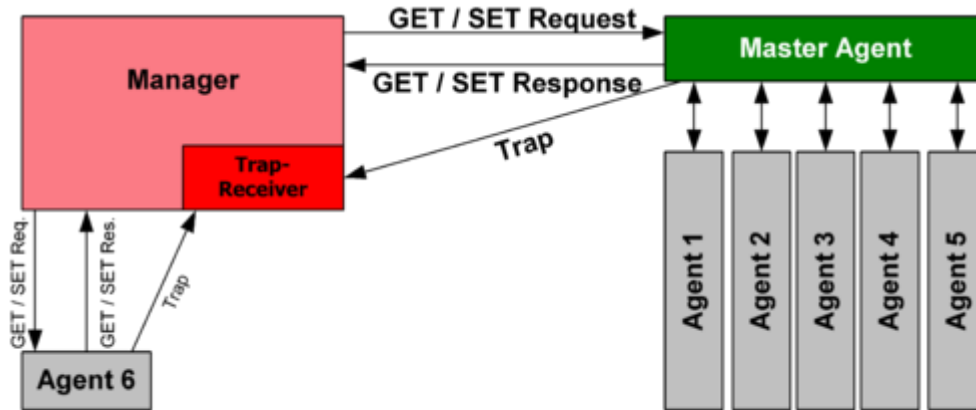


Figure 10 SNMP principle of communication [10]

The manager may send requests from any available source port to port 161 in the agent. The agent response is sent back to the source port on the manager. The manager receives notifications (Traps and InformRequests) on port 162. The agent may generate notifications from any available port. When used with Transport Layer Security or Datagram Transport Layer Security, requests are received on port 10161 and notifications are sent to port 10162[18].

Different versions of SNMP have been developed in order to efficiently control the elements of a network. The first versions did not emphasize much in the security of the protocol. Next versions implemented solutions on the security issues that arose. In order to conform to the versions different RFCs have been developed. In SNMPv1 specifies five core protocol data units (PDUs). In SNMPv2 two more PDUs were implemented, the GetBulkRequest and InformRequest and in SNMPv3 the Report PDU was added. Even though there are differences between each version all SNMP PDUs are synthesized as follows [10].

IP header	UDP header	version	community	PDU-type	request-id	error-status	error-index	variable bindings
-----------	------------	---------	-----------	----------	------------	--------------	-------------	-------------------

PDU type is described by the PDU-type field which is filled with one of the following:

**GetRequest**

A manager-to-agent request to retrieve the value of a variable or list of variables. Desired variables are specified in variable bindings (the value field is not used). Retrieval of the specified variable values is to be done as an atomic operation by the agent. A Response with current values is returned.

### **SetRequest**

A manager-to-agent request to change the value of a variable or list of variables. Variable bindings are specified in the body of the request. Changes to all specified variables are to be made as an atomic operation by the agent. A Response with (current) new values for the variables is returned.

### **GetNextRequest**

A manager-to-agent request to discover available variables and their values. Returns a Response with variable binding for the lexicographically next variable in the MIB. The entire MIB of an agent can be walked by iterative application of GetNextRequest starting at OID 0. Rows of a table can be read by specifying column OIDs in the variable bindings of the request.

### **GetBulkRequest**

A manager-to-agent request for multiple iterations of GetNextRequest. An optimized version of GetNextRequest. Returns a Response with multiple variable bindings walked from the variable binding or bindings in the request. PDU specific non-repeaters and max-repetitions fields are used to control response behavior. GetBulkRequest was introduced in SNMPv2.

### **Response**

Returns variable bindings and acknowledgement from agent to manager for GetRequest, SetRequest, GetNextRequest, GetBulkRequest and InformRequest. Error reporting is provided by error-status and error-index fields. Although it was used as a response to both gets and sets, this PDU was called GetResponse in SNMPv1.

### **Trap**

Asynchronous notification from agent to manager. While in other SNMP communication, the manager actively requests information from the agent, these are PDUs that are sent from the agent to the manager without being explicitly requested. SNMP traps enable an agent to notify the management station of significant events by way of an unsolicited SNMP message. Trap PDUs include current sysUpTime value, an OID identifying the type of trap and optional variable bindings. Destination addressing for traps is determined in an application-specific manner typically through trap configuration variables in the MIB. The format of the trap message was changed in SNMPv2 and the PDU was renamed SNMPv2-Trap.

### **InformRequest**

Acknowledged asynchronous notification. This PDU was introduced in SNMPv2 and was originally defined as manager to manager communication.[4] Later implementations have loosened the original definition to allow agent to manager communications.[19][20][21] Manager-to-manager notifications were already possible in SNMPv1 using a Trap, but as SNMP commonly runs over UDP where delivery is not assured and dropped packets are not reported, delivery of a Trap was not guaranteed. InformRequest fixes this as an acknowledgement is returned on receipt.[10]

According to RFC 1157, an SNMP implementation must be able to accept a message of at least 458 bytes length. In reality, the messages are longer. Malformed SNMP requests are discarded. Successfully decoded SNMP requests are authenticated by community string. If authentication fails, a trap is sent indicating an authentication failure message, and the message is dropped.



Figure 11 SNMP message types[14]

As mentioned above, three versions of SNMP have been deployed Version1, Version2 and Version3. The specifications of Version1 were described in requests for comments (RFCs) 1065, 1066 and 1067 in 1988. In 1990s the above documents were superseded by RFC 1155, 1156, 1157 and in 1991 the 1156 was replaced by 1213. Version 1 has been considered as unsafe due to its poor security, however the specification allow some custom authentication that were used in more trivial methods. As a result the security of each message depends on the security of the channel itself. For example, an organization may consider their internal network to be sufficiently secure that no encryption is necessary for its SNMP messages. In such cases, the community name, which is transmitted in cleartext, tends to be viewed as a de facto password, in spite of the original specification.[10]

SNMPv2, defined by RFC 1441 and RFC 1452, revises version 1 and includes improvements in the areas of performance, security and manager-to-manager communications. It introduced GetBulkRequest, an alternative to iterative GetNextRequests for retrieving large amounts of management data in a single request. The new party-based security system introduced in SNMPv2, viewed by many as overly complex, was not widely adopted. This version of SNMP reached the Proposed Standard level of maturity, but was deemed obsolete by later versions.[22] SNMPv2 also introduced the option for 64-bit data counters; Version 1 was designed only with 32-bit counters. Due to differences between v1 and v2 as far as PDU and header format, RFC 3584 defines two SNMPv1/v2c coexistence strategies: proxy agents and bilingual network-management systems in order to overcome incompatibility. This version can be described as an enhanced version of v1 using the same administration structure (“community based”) with some enhancements as far as packet types, transport mapping and MIB structure elements are concerned.

Even though Version 3 doesn’t make changes to the protocol, except for the enhanced security, it looks very different from the previous versions due to new textual conventions, concepts and terminology[20]. Version 3 implemented a secure version of SNMP by adding security and remote configuration enhancements to SNMP. The security aspect is addressed by offering both strong authentication and data encryption for privacy. For the administration aspect, SNMPv3 focuses on two parts, namely notification originators and proxy forwarders. The changes also facilitate remote configuration and administration of the SNMP entities, as well as addressing issues related to the large-scale deployment, accounting, and fault management[10].

The security features provided in SNMPv3 are as follows:

- Message integrity—Ensures that a packet has not been tampered with during transit.
- Authentication—Determines that the message is from a valid source.
- Encryption—Scrambles the content of a packet to prevent it from being learned by an unauthorized source.[24]

SNMPv3 is a security model in which an authentication strategy is set up for a user and the group in which the user resides. Security level is the permitted level of security within a security model. A combination of a security model and a security level determines which security mechanism is used when handling an SNMP packet.

The table below describes the combinations of SNMPv3 security models and levels

Table 1 SNMPv3 security levels[24]

Level	Authentication	Encryption	What Happens
noAuthNoPriv	Username	No	A username match is used for authentication
autNoPriv	MD5 or SHA	No	Provides authentication based on the Hashed Message Authentication Code (HMAC)-MD5 or HMAC-SHA algorithms.
authPriv	MD5 or SHA	Data Encryption Standard	Provides MD5 and SHA Authentication and also provides DES 56-bit encryption based on Cypher Block Chaining

As far as Architecture of SNMPv3 is concerned, the specifications of the Internet-Standard Management Framework are based on a modular architecture. This framework is more than just a protocol for moving data. The framework consists of

- A data definition language
- Definitions of management information (the Management Information Base, or MIB)
- A protocol definition
- Security and administration.

The framework was structured with a protocol-independent data definition language and Management Information Base, along with a MIB-independent protocol. The SNMPv3 Framework builds and extends these architectural principles by building on these four basic architectural components, in some cases incorporating them from the SNMPv2 Framework by reference, and by using these same layering principles in the definition of new capabilities in the security and administration portion of the architecture.

Those who are familiar with the architecture of the SNMPv1 Management Framework and the SNMPv2 Management Framework find many familiar concepts in the architecture of the SNMPv3 Management Framework. However, in some cases, the terminology may be somewhat different[25]. From 2004 the IETF considers SNMPv3 a full internet standard as defined in RFC 3411-3418. Previous versions are considered obsolete.

### 3.2 SCPI

Amongst other protocols, SCPI (pronounced “skipi”) stands for Standard Commands for Programmable Instruments and might not seem to intrigue the attention in telecommunication systems, but is important to refer to, after all 5G and 6G implement the usage of IOT. In set of sentences we could define SCPI a standardized communication protocol used in the field of test and measurement equipment, such as oscilloscopes, multimeters, and other electronic instruments. SCPI defines a set of commands and syntax that allows a computer or controller to communicate with and control these instruments over various communication interfaces, such as GPIB (General Purpose Interface Bus), LAN (Local Area Network), USB (Universal Serial Bus), and others.



Figure 12 Back of instrument providing connections for remote control [29]

The SCPI standard was developed to provide a common language for programming and controlling instruments from different manufacturers. This standardization helps ensure interoperability between instruments and simplifies the process of writing software or scripts to automate test and measurement tasks.

SCPI commands typically follow a hierarchical structure and consist of a series of keywords and parameters. Users can send SCPI commands to configure instrument settings, initiate measurements, and retrieve data. The standard covers a wide range of functionalities, making it versatile for various types of test and measurement applications.

In 1975 the IEEE standardized a bus developed by Hewlett-Packard originally called HPIB (Hewlett-Packard Interface Bus) which was later renamed to GPIB (General Purpose Interface Bus). The standard was called IEEE488 which defined mechanical aspects of the bus. With standard IEEE488.2 protocol properties were defined, however a set of rules between the manufacturers on commands to control the instruments was still missing. Even between different models from the same manufacturer the commands were different something that made controlling and implementing of these instruments quite difficult.[27]

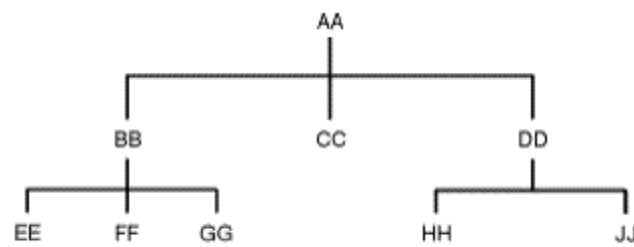
The physical hardware communications link is not defined by SCPI. While it was originally created for the IEEE-488.1 (GPIB) bus, SCPI can also be used with RS-232, RS-422, Ethernet, USB, VXIbus, HiSLIP, etc. SCPI commands are ASCII textual strings, which are sent to the instrument over the physical layer (e.g., IEEE-488.1). Commands are a series of one or more keywords, many of which take parameters.

In order to understand better the architecture of this protocol some definitions are provided:

- **Command:** A command is an instruction in SCPI consisting of mnemonics (keywords), parameters (arguments), and punctuation. You combine commands to form messages that control instruments.
- **Controller:** A controller is any device used to control the instrument, for example the instrument itself, a computer, or another instrument.
- **Event Command:** Some commands are events and cannot be queried. An event has no corresponding setting; it initiates an action at a particular time
- **Program Message:** A combination of one or more properly formatted commands.
- **Query:** A special type of command used to instruct the instrument to make response data available. A query ends with a question mark. Query any command value that is set is available.
- **Response message:** A collection of data in specific SCPI formats sent from the instrument to the controller. Response messages inform the controller about the internal state of the instrument

So SCPI commands either perform a set operation, for instance power on, or a query operation par example a reading of an input voltage to the instrument. As mentioned above queries require a question mark at the end of the command. A group of commands can be used for both set or queries. The distinction between the functionality is the question mark at the end of the command. For example, ACQUIRE:MODE and ACQUIRE:MODE? Commands. Another set of commands also set and query both at the same time. For example, the \*CAL? command runs a self-calibration routine on some equipment, and then returns the results of the calibration.

Except for the distinction between queries and set commands, the commands can be separated into two groups: common commands and subsystem commands. Common commands are used to manage status registers, synchronization, and data storage and are defined by IEEE 488.2. Common commands begin with an asterisk (\*). This type of commands are not part of a subsystem and are interpreted by the system in the same way, regardless the path setting. On the other hand, subsystem commands follow a tree structure and are easily distinguished from the colon “:” character. This special character is used at the beginning of the command statement and between keywords, for instance :CONTRol:IO1:OUTPut. The tree structure that the protocol follows is similar to a computer filesystem architecture. A simplified example is shown in the figure below.



ck710a

Figure 13 Simplified SCPI commad tree

This architecture is designed to provide a common language that facilitates communication between instruments and controllers, regardless of the manufacturer, however each manufacturer provides a user manual with the supported commands for the instrument used. As discussed above the commands are easy to comprehend and use due to the consistent syntax, standard prefixes and special characters and keywords. SCPI serves its purpose mainly because it has a set of commands that identify the equipment at hand, it ensures interoperability between the instruments, transfers data and specifically measurement data and supports different communication interfaces such as GPIB, LAN, USB and more. The identification of the instrument includes the manufacturer, the model and the serial number. This feature can be quite useful in remote controlling the devices in case of a failure. Another asset as far error handling is concerned is the definition of a standardized error reporting mechanism from the protocol. Instruments should provide detailed error information to help users diagnose and address issues. The interoperability and usefulness of this protocol is depicted in the way remote controlling can be achieved. Specifically, it allows users to write scripts or programs in various languages that can control the devices of different vendors without major modifications.

Such paradigm is followed mostly nowadays, mostly with the use of LabView and interfaces that allow the virtualization of the instrument. An approach by Balaji, Aravind & Sasikumar, Subramanian & K, Dr. Ramesh [28] shows the way such paradigm can be followed.



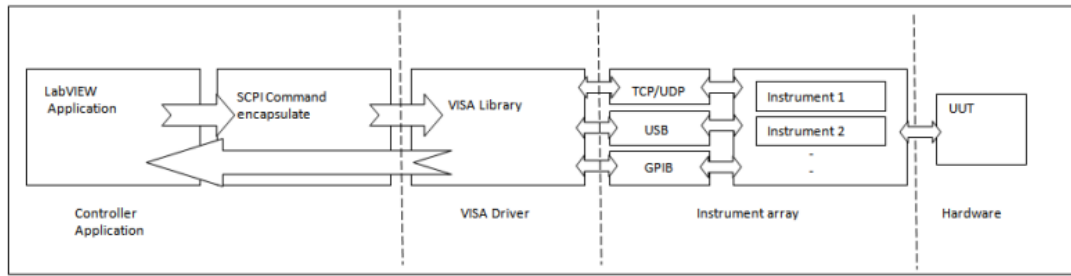


Figure 14 Design Flow of the integrated environment [28]

In conclusion this protocol is used to remotely control instruments through GPIB, LAN, USB, ethernet and more in a way that provides interoperability between devices. It also provides the user the ability to integrate other scripts or programs encouraging IOT paradigms and remote control and troubleshooting. It is important to mention the crucial role of VISA in such paradigm. VISA stands for Virtual Instrument Software Architecture and it is an industry-standard API that is used to communicate with instruments from a computer. There are many different versions of VISA – the most common is National Instruments NI-VISA, but there is also the Keysight IO Libraries, Tektronix TekVISA, Rohde & Schwarz VISA and more. While they are supposed to be interoperable to some extent, software which is built against one particular VISA may have difficulty operating with another which may necessitate having multiple VISAs installed with one designated as the primary VISA. Occasionally, this could result in a conflict which breaks instrument connectivity altogether. The job of the VISA is to interface your application program (which could be written in C or any other language where libraries are provided for your VISA) to the device through the operating system and its drivers. Although this protocol and the manufacturers provide flexibility and interconnection, some issues can be encountered due to slow instruments or long command sequences which might take instruments longer than expected to respond. This will result in a timeout error if it takes longer than the default 2000ms timeout. When integrating instruments in remote control the synchronization is a very crucial aspect that must be taken into severe consideration.

Most instrument manufacturers provide the documentation, the libraries even examples in order to integrate the devices into a system as shown in the figure below.

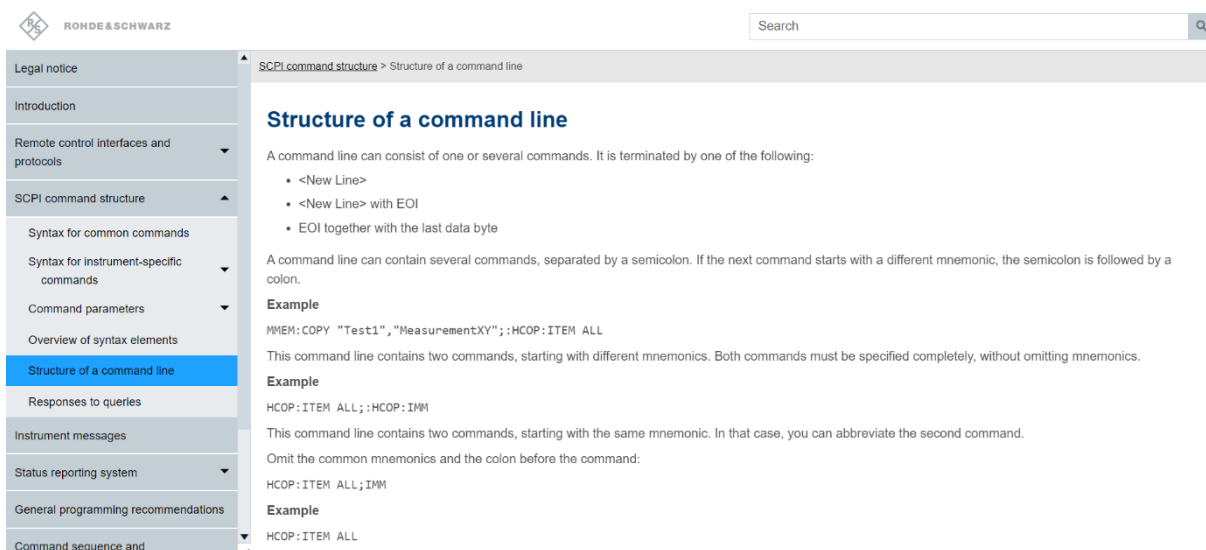


Figure 15 Complete documentation, libraries and examples from manufacturer [27]

In such way of integrating and remote controlling the controller can be a pc that is connected on the internet with a remote server that will control the controller. This kind of architecture and connection flexibility can provide a lot of potential for future work and even though this protocol has a history of around 30 years it is still reliable and useful with a lot of potential in system integration. As much as it is desired to remotely control instruments and integrate such devices, safety measures must be taken. When using this kind of equipment that are able to control current and voltages there is risk of overcharging or malfunctions. System engineering using this kind of automation should always take into account the worst-case scenario and implement ways of safely treating malfunctions and inconvenient situations.

### 3.3 TL1 (Transaction Language 1)

Transaction Language 1 (TL1) is a set of ASCII-based instructions, or "messages". These messages allow a human user or an Operations Support System (OSS) to manage a network element (NE) and its resources. It is a widely used protocol in telecommunication systems especially in SONET/SDH systems and considered a man machine language. TL1 is the primary method for managing conventional telecom equipment such as access devices as well as optical networking gears such as SONET/SDH boxes. TL1 is a messaging protocol that represents data as human-readable strings of characters rather than as objects. Study of this protocol is important in order to understand the control of network elements in Synchronous Optical Networking (SONET) and Synchronous Digital Hierarchy.

SONET and SDH are standardized protocols that transfer multiple digital bit streams synchronously over optical fiber or highly coherent light emitted from LEDs. A majority of the backbone transport for voice, video and data applications continues to be SONET and SDH optical transmission networks. SONET and SDH transmission network also continue to be used for conventional channelized traffic. TL1 is the language that organizes and controls the systems that implement a SONET/SDH protocol.

TL1 was developed by Bellcore in 1984 as a standard man-machine language to

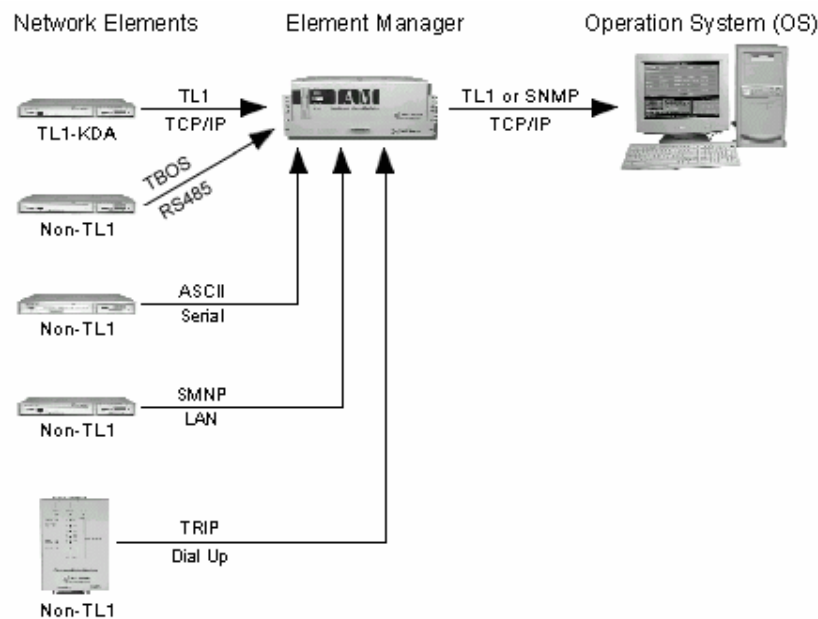


Figure 16 LAN transport of TL1 data [31]

manage network elements for the Regional Bell Operating Companies (RBOCs). It is based on Z.300 series man machine language standards. TL1 was designed as a standard protocol readable by machines as well as humans to replace the diverse ASCII based protocols used by different Network Element (NE) vendors. It is extensible to incorporate vendor specific commands. Telcordia OSSs such as NMA (Network Monitoring and Analysis) used TL1 as the element management (EMS) protocol. This drove network element vendors to implement TL1 in their devices. [32]

TL1 commands are issued to the system from a craft terminal (a terminal that performs local maintenance operations) or from an Operations Support System Interface (OSSI).

To issue commands, the user must be logged on to the shelf or node that is the target of the command. The shelf or node that is the target of the command is referred to as the targeted system.

A session is the period of time when a user is logged on to the system. The process of logging on is referred to as opening a session. Logging off is referred to as closing a session. To open a TL1 session, a user must have a user identifier (UID) and a private identifier (PID, or password). These codes are assigned by the system administrator.

The UID/PID database may be accessed on a per-user basis, so that if a session is inactive for 30 minutes, the system automatically closes the session.[33]

So in order to communicate with a network element the OSS user connects to the element and sends a message through a secure session. The messages transmitted have 4 types:

- Input message: Command sent by the user or the SSO to the NE (Network element)
- Output/Response message: Reply from the NE towards the SSO (or user) in response to the input message
- Acknowledgement message: This message is an acknowledgement that the NE received the input message and is sent if the response message needs more than 2 seconds to be sent
- Autonomous message: Asynchronous messages, usually alarms or events sent by the NE.

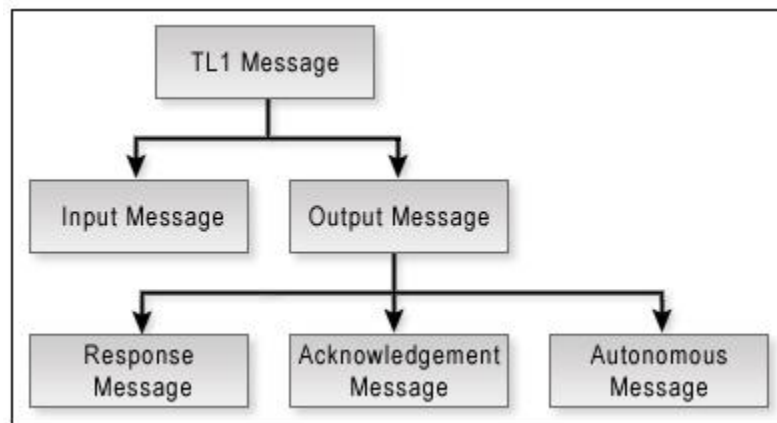


Figure 17 Simplified representation of message types

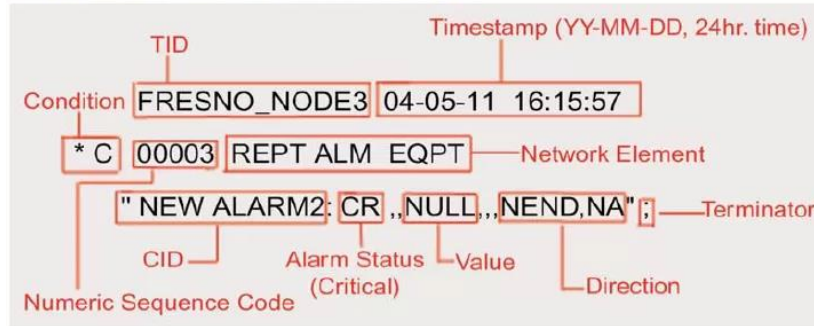
Each TL1 command is divided into a set of positional parameters that represent actions or objects. TL1 commands must be typed accurately, with no spaces and with parameters separated by colons. For example,

```
COMMAND:TID:AID:CTAG:KEYWORD=DOMAIN:STATE;
```

TID, AID, and CTAG are used for routing and controlling the command. The other parameters provide the information to complete the action that the command requests. TL1 commands are not case-sensitive. Commands, keywords, and domains can be strung together using commas. For example:

```
ED-SYS:TID::CTAG::TMG=EXT1544,AAT=0,ADT=0;
```

Sample Autonomous Message #1 (Critical Alarm):



Code	Condition
*C	Critical
**	Major
*^	Minor
A^	Status
M	Response

These codes are used to indicate an alarm severity or a response message.

Figure 18 Sample of autonomous message [36]

Some of the TL1 parameters used are described in the following table

Table 2 TL1 Command parameters[34]

Parameter	Description
<b>AID</b>	The AID is an access code used to identify the exact address of the equipment or facilities (channels) within the command.
<b>AID Type</b>	The access identifier (AID) type specifies whether the AID applies to equipment, communications, or one of the traffic or data channels.
<b>CTAG</b>	The correlation tag (CTAG) is a unique identifier that the operator gives to each input command. When the network element responds to that command, it includes the CTAG of the command in the reply. This prevents any confusion about which response applies to which command. CTAGs consist of up to six ASCII characters.
<b>DOMAIN</b>	<p>This parameter contains the list of possible settings for a given keyword. Only one value from the domain can be chosen for each use of the keyword.</p> <p>The proper domain setting must be chosen for the specified keyword when editing the system, equipment, or facilities.</p> <p>Keywords not used in a command are not affected by the command.</p>
<b>PID</b>	The private identifier (PID) is the password associated with the UIS.
<b>SID</b>	The source identifier (SID) is the same as a TID, except that it is used when the

	network element is identifying itself in a response message.
<b>TID</b>	<p>The target identifier (TID) is a unique name given to each system when it is installed in the network. This name identifies the particular network element to which each command is directed. Each TID comprises a maximum of 20 alphanumeric characters.</p> <p><b>Note:</b> If the data in the parameter reserved for the TID is not data that has been assigned to any NE as a SID, the command receives a timeout response. When the parameter position reserved for the TID is empty, or null, the parameter defaults to the value placed in that position in the last command entered.</p>
<b>UIS</b>	The user logon name.

After defining the command structure, we are able to describe the message structure in TL1. TL1 messages consist of the following parts: header, identifier, text block and terminator. Each part has a set of tokens that convey details about the network element and the event being reported.

**Header:** The header format is the same for autonomous and output response messages. It consists of TID (terminal identifier) and the date and time the event occurred.

**Identifier:** Contains information relevant to the nature of the message. For autonomous messages it contains an autoid that identifies the importance of the message. In output messages it contains a correlation tag (CTAG) that identifies to the input message that triggered this response and completion code (CompCode).

**Text Block:** Optional component that contains information specific to a message.

**Terminator:** Indicates the end of the message.

For instance this is a complete message in TL1:

```
<cr> <lf> <lf>
```

```
^^^BOCALFMA010^03-06-02^01:10:20 <cr> <lf>
```

```
*^^456.123^REPT^ALM^T1 <cr> <lf>
```

```
^^^"T1AID-16:MN,CGA,NSA" <cr> <lf>
```

```
^^^"T1AID-32:<cr> <lf> MN,CGA,NSA" <cr> <lf>
```

;

**Where:**

^^^ are spaces

BOCALFMA010 is the AID

03-06-02 is the date

01:10:20 is the time

456.123 is the Atag (Autonomous Correlation Tag)

REPT is the verb

ALM is an additional modifier to the verb

T1 is an additional modifier to the verb

"T1AID-16:MN,CGA,NSA" <cr> <lf> is a quoted line

Another example with a visualization of this format is:

```
MyNE 04-08-14 09:12:04
M 101 COMPLD
"UID=sridev:CID=CRAFT,UAP=1:"
;
[32]
```

TL1 output message						
Response Header			Response Id		Response block	Terminators
SID	Date	Time	M	CTAG	Completion code	
MyNE	04-08-14	09:12:04	M	101	COMPLD	"UID=sridev:CID=CRAFT,UAP=1:";

While TL1 (Transaction Language 1) is a widely used language in the telecommunications industry for managing and monitoring network elements, it does have some drawbacks. A few common drawbacks associated with TL1 are:

1. Human-Readability: TL1 commands are often considered less human-readable compared to some other network management languages. The syntax and structure may be complex, making it challenging for operators and administrators to understand and work with directly.
2. Limited Standardization: Although TL1 is a standardized language, the degree of standardization can vary across different vendors. This lack of strict adherence to a universal standard may lead to compatibility issues and interoperability challenges when working with equipment from different manufacturers.

3. Security Concerns: TL1 does not inherently provide robust security features. The lack of built-in encryption and authentication mechanisms can pose security risks, especially when communicating over unsecured networks. Security concerns are particularly important in today's environment where network security is a top priority.

4. Scalability Challenges: As network complexity and size increase, managing devices using TL1 commands might become less scalable. The manual execution of commands for large-scale network operations may become cumbersome and inefficient.

5. Dependency on Specific Equipment: TL1 is often associated with telecommunications equipment, and its usage may be limited to specific devices and vendors. This dependency can create challenges if an organization wants to diversify its network infrastructure or adopt equipment from different suppliers.

6. Limited Extensibility: Extending or customizing TL1 commands for specific needs can be challenging. The language may not be as extensible as some other network management protocols, limiting the ability to adapt to unique requirements without significant effort.

7. Proprietary Implementations: Some implementations of TL1 by different vendors may include proprietary extensions or variations. This can result in vendor lock-in and complicate interoperability when attempting to integrate equipment from different manufacturers.

8. Learning Curve: Learning TL1 commands and syntax may require additional training for network operators who are more familiar with other network management languages. This learning curve can be a barrier, especially when transitioning to TL1 from other more user-friendly protocols.

Despite these drawbacks, it's essential to note that TL1 continues to be widely used in the telecommunications industry. Many organizations have successfully integrated TL1



into their network management systems, leveraging its strengths while addressing its limitations through proper planning and implementation strategies



Figure 19 Integration of TL1 with Java

There are many reasons why TL1 is still used especially in the backbone network, some of them being the integration with legacy systems, multi-protocol support where TL1 and SNMP coexist along with NETCONF (which will be described in later chapter), the support from the equipment vendors and the transition strategies that were followed while evolving the core network. In cases where different devices in the network use different protocols modern network management systems employ protocol mediation mechanisms. An example of this integration paradigm is the SNMP proxy agent as described in [35]. Last but not least, as mentioned many times in this essay, 5G and 6G networks endorse cloud-native solutions. By integrating TL1 in such environments the management of both traditional and cloud-based networks are achieved.

In conclusion, TL1 is a very useful protocol for managing network elements especially those of SONET/SDH systems. Despite the fact that it has been introduced in 1984 it is still a widely used language. There are similarities between the already mentioned protocols such as SNMP with some differences off course in the structure and syntax. Developers didn't cancel this protocol but integrated TL1 in the future architectures by implementing ways like protocol mediation mechanisms or proxy agents [35]. Nevertheless, the criteria of selection in choosing the appropriate protocol or language lies in the system specifications and limitations, as well as the deployment that will be followed. Different languages may excel in different aspects, and the choice often depends on the goals and characteristics of the network management system.

### 3.4 NETCONF

Network Configuration Protocol (NETCONF) is an XML-based network management protocol that provides a programmable method to configure and manage network

devices. NETCONF was defined in RFC 4741 by the Internet Engineering Task Force (IETF) and revised in RFC 6241. NETCONF provides a standard framework and follows the Procedure Call (RPC) paradigm, through which network administrators and application developers can manage configurations of network devices and obtain network device status promptly. The NETCONF packets are in XML format and the NETCONF protocol has a powerful filtering capability. Each data field has a fixed element name and position. Therefore, the devices of the same vendor can use the same access mode and result display mode. The devices of different vendors can achieve the same effect by XML mapping. This feature facilitates third-party software development and NMS software customization in the multi-vendor, multi-device environment. With the help of such NMS software, NETCONF simplifies device configuration and improves device configuration efficiency.

NETCONF is a key protocol in the field of network management that plays a crucial role in modern networking environments. The importance of this protocol can be depicted through various aspects such as the standardized configuration, the YANG Modeling, the structured Data exchange and the remote device configuration. In addition, the transaction support and the security features that are provided as well as the adaptability to network changes through multiprotocol support, make NETCONF a widely used protocol.

Going through the specifications and the importance of this protocol, this chapter will allow the reader to comprehend the use of NETCONF and why it can be chosen for different networking systems. As mentioned in previous chapter there are similarities and differences between all the protocols as far as use and format are concerned. However, the choice of the protocol depends on the system engineered and the specifications provided.

#### 3.4.1 NETCONF Data Modeling and Operations

The NETCONF packets are in XML format and the NETCONF protocol has a powerful filtering capability. Each data field has a fixed element name and position. Therefore, the devices of the same vendor can use the same access mode and result display mode. The devices of different vendors can achieve the same effect by XML mapping. This feature facilitates third-party software development and NMS software customization in the multi-vendor, multi-device environment. With the help of such NMS software, NETCONF simplifies device configuration and improves device configuration efficiency. Due to its use on managing network elements the protocol subsequently added the support for encoding in JSON (JavaScript Object Notation).

This protocol is logically partitioned in four layers as represented in Figure 20

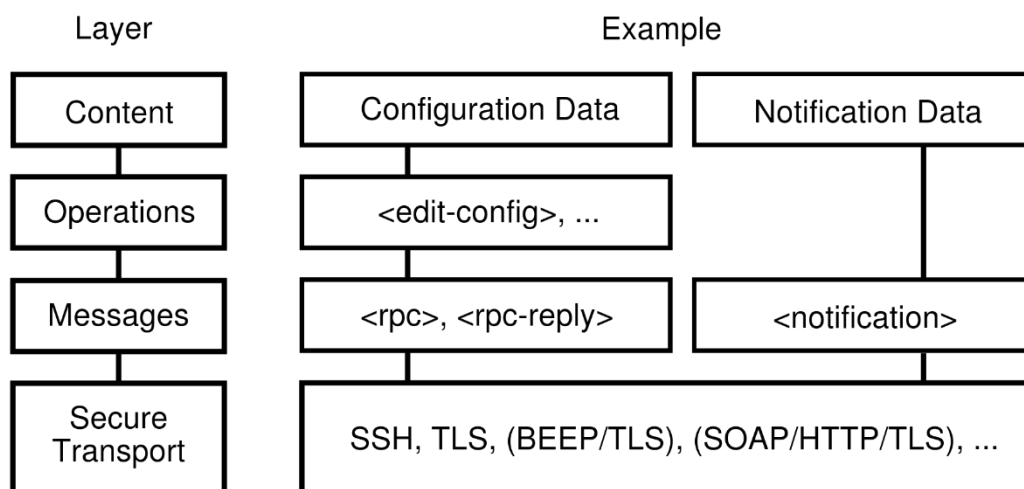


Figure 20 NETCONF protocol layers [38]

- Content layer: It consist of configuration data and status data. Describes configuration data involved in network management. Most configurations do not have standard NETCONF data models, so the devices from different vendors may use different configuration data.
- Operation layer: The operation layer defines a series of operations used in RPC. These operations compose the basic capabilities of NETCONF in order to retrieve and modify the configuration data. The content in this layer can be <get>, <get-config>, <edit-config>.
- Message layer (or RPC layer): This layer provides a simple RPC request and reply mechanism independent of transport protocols. The client uses the <rpc> element to encapsulate RPC request information and sends the RPC request information to the server using a secure, connection-oriented session. The server uses the <rpc-reply> element to encapsulate RPC response information (content at the operation and content layers) and sends the RPC response information to the client. Huawei switch functions as the NETCONF server to receive NETCONF requests from the NETCONF client. Normally, the <rpc-reply> element encapsulates data required by the client or a configuration success message. If the client sends an incorrect request or the server fails to process a request from the client, the server encapsulates the <rpc-error> element containing detailed error information in the <rpc-reply> element and sends the <rpc-reply> element to the client. The content of this layer can be <rpc> , <rpc-reply>.[37]
- Transport layer: This layer provides a secure and reliable transport of messages between a client and a server. NETCONF can be layered over any transport protocol that meets the following basic requirements:

- The transport protocol is connection-oriented. The NETCONF manager and NETCONF agent must establish a persistent connection. This connection must provide reliable, sequenced data transmission.
- The transport layer provides user authentication, data integrity, and confidentiality for NETCONF.
- The transport protocol provides a mechanism to distinguish the session type (client or server) for NETCONF.

The transport protocols that meet the requirements and that are used for NETCONF are BEEP, SSH, TLS, SOAP, HTTP.

In order to understand how the NETCONF server communicates with the NETCONF client a schematic representation is shown below.

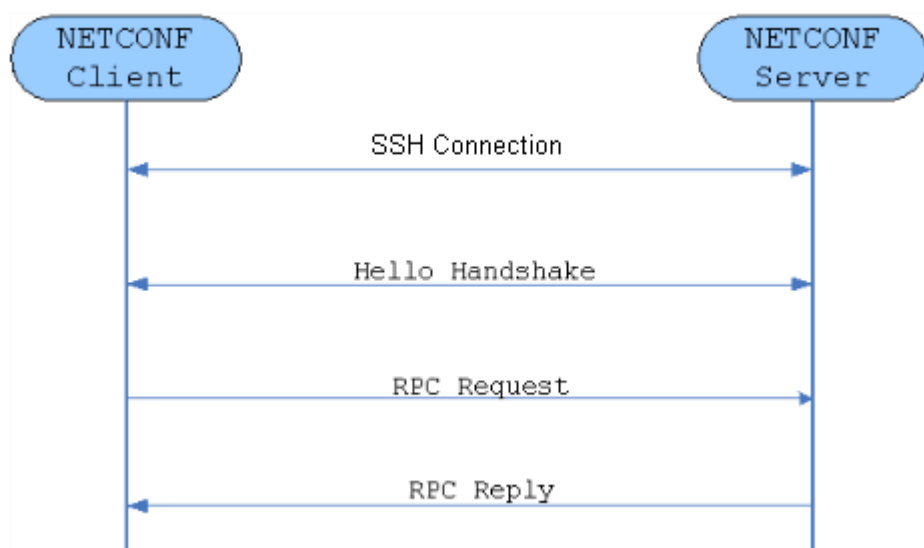


Figure 21 NETCONF Interaction process [37]

The NETCONF client initiates the SSH connection with the server. After the connection is established the server sends a “hello” packet to the client notifying him of his capability sets and in return after the reception the client sends back to the server a “hello” packet with its own capability sets. After the “hello” packet procedure is finished the client sends an RPC request to the server for configuration and management. After analyzing the RPC request from the client, the server returns an RPC reply. In that way the communication between server and client is established and the system is manageable and controllable.

In order for the system to be controlled and manageable this protocol defines the following operations.

Operation	Description
-----------	-------------

<get>	Retrieve running configuration and device state information
<get-config>	Retrieve all or part of a specified configuration datastore
<edit-config>	Edit a configuration datastore by creating, deleting, merging or replacing content
<copy-config>	Copy an entire configuration datastore to another configuration datastore
<delete-config>	Delete a configuration datastore
<lock>	Lock an entire configuration datastore of a device
<unlock>	Release a configuration datastore lock previously obtained with the <lock> operation
<close-session>	Request graceful termination of a NETCONF session
<kill-session>	Force the termination of a NETCONF session

NETCONF data modeling is subject to YANG data modeling language; after the description of data modeling and operations in NETCONF it is important to present YANG and how NETCONF followed this paradigm, as well as why it is important for NETCONF to apply to the rules defined by YANG.

YANG stands for “Yet Another Next Generation” and is a data modeling language for the definition of data sent over network management protocols. As mentioned above NETCONF is a protocol that follows YANG paradigm. YANG is maintained by the NETMOD working group in the IETF and originally published in RFC 6020 with an update in RFC 7950. The data modeling language can be used to model both configuration data as well as state data of network elements. In addition, YANG can also define the format of event notifications sent by network elements and it allows data modelers to define the signature of remote procedure calls that can be invoked on network elements via the NETCONF protocol. The language, being protocol independent, can then be converted into any encoding format, e.g. XML or JSON, that the network configuration protocol supports. In simple words, NETCONF was standardized protocol, but the data content was not standardized; as a result, the data content had to be modeled and YANG was created. The importance of YANG lies in the

fact that is protocol independent and simple to understand. In addition, compared with SNMP and MIBs YANG is more hierarchical, can distinguish between configurations and status and provides high flexibility. So independently of the encoding used xml or json or even the protocol used (NETCONF, RESTCONF) the way the data is organized in the encoding is the same. That is the most important aspect of YANG.

### 3.4.2 Advantages of NETCONF and Use Cases

After analyzing the processes and data structure of NETCONF protocol, as well as the importance of it, we are able to evaluate the usefulness of adopting NETCONF protocol. The evaluation performed takes into consideration the protocols mentioned in previous chapters as well as the operability in a state-of-the-art environment following the 2024 trends in the scientific community as well as the potential of technologies researched. Cloud computing and virtualization are core ideas in developing future networks. Recognizing the importance of these two leads us to the realization that virtualization and cloud computing involve huge networks or even better network of networks. That leads to the conclusion that in all these different systems and networks there are a lot of different devices and structures. In order to adapt in the era of virtualization and cloud computing there are some key indicators in order to chose which protocol. In addition, security is a very important part of telecommunication systems and networks. Taking all the above into consideration, a brief comparison between SNMP and NETCONF was performed in order to show the advantages. However, it is important to understand that each protocol has its own advantages and disadvantages and the systems specifications must be taken into consideration.

As far as configuration protection is concerned SNMP doesn't provide something like this. NETCONF provides a lock mechanism to prevent multiple user configuration conflicts while also ensuring the backup od these configurations. Multiple configuration databases are provided in NETCONF which are backups of each other. SNMP provides configuration queries but requires multiple times of interaction when in NETCONF you can query all configuration data of one object based on filtering conditions with a batch data collection speed 10 times faster than SNMP. Important aspect in developing a network management system is the scalability of the protocol used. SNMP provides poor scalability, when NETCONF uses a multi-layer model where each layer is independent of the other and the extension of one layer doesn't have impact on the other layer. In addition, the XML format of NETCONF expands the capability of management capability and system compatibility. Last but not least, security is a very crucial aspect in network protocols. As mentioned in chapter 3.1 SNMP added security levels in version 3 through MD5 and SHA. This provides authentication and encryption that is limited though through functions that cannot be expanded. NETCONF uses existent security protocols, such as SSH and SOAP in order to ensure network security and is not specific to any security protocols; this provides flexibility in security protection.

Having in mind the above comparison and the advantages of NETCONF a use case of this protocol is presented.

### **Use Case: Automated Configuration Deployment**

#### **Scenario:**

Imagine an enterprise network with a diverse set of networking devices from different vendors. The network administrator needs to deploy a new service that involves configuring multiple devices across the network. This service requires changes to be made in the configurations of routers, switches, and firewalls.

#### **Challenges:**

- **Diverse Vendor Devices:** The network comprises devices from different vendors, each with its own configuration syntax and command structure.
- **Large-scale Deployment:** The deployment involves a significant number of devices spread across different locations, making manual configuration impractical and error-prone.
- **Consistency and Accuracy:** Ensuring that the configurations are consistent across devices is crucial to avoid operational issues and potential security vulnerabilities.

#### **Solution using NETCONF:**

##### *YANG Data Models:*

Network administrators use YANG data models to define the configurations required for the new service. YANG provides a standardized way to model the configuration data, ensuring consistency and clarity.

##### *NETCONF Transactions:*

The network administrator creates a NETCONF transaction that includes all the necessary configuration changes for the new service. The transaction groups multiple configuration changes into a single atomic operation.

##### *Device Independence:*

NETCONF allows the network administrator to interact with devices from different vendors using a uniform set of operations. This ensures that the same NETCONF operations can be used regardless of the device type or vendor.

##### *Secure Communication:*

The configuration changes are sent securely using NETCONF over SSH, ensuring the confidentiality and integrity of the data in transit.

### *Error Handling and Rollback:*

NETCONF provides mechanisms for error handling. If an error occurs during the transaction, NETCONF allows for a rollback to the previous configuration state, preventing partial or inconsistent changes.

### *Automated Scripting:*

The network administrator can use scripts or automation tools to generate and execute NETCONF operations, streamlining the deployment process. This automation reduces the likelihood of human errors associated with manual configuration.

### *Real-time Monitoring:*

NETCONF allows for real-time monitoring of the operational state of devices. Network administrators can use NETCONF operations to retrieve information about the status and performance of devices post-configuration.

### **Benefits:**

#### Efficiency and Speed:

Automated configuration deployment using NETCONF significantly speeds up the deployment process, allowing the network to adapt quickly to changing service requirements.

#### Consistency and Accuracy:

The use of standardized YANG data models and atomic transactions ensures that configurations are consistent and accurate across all devices, reducing the risk of misconfigurations.

#### Scalability:

NETCONF's scalability allows administrators to deploy configurations simultaneously across a large number of devices, making it suitable for large-scale network environments.

#### Vendor Independence:

The use of NETCONF allows the enterprise to deploy configurations across devices from different vendors without having to deal with vendor-specific command syntax.

This use case illustrates how NETCONF, with its standardized approach to configuration management, automation capabilities, and support for diverse devices, can be a valuable tool for efficiently deploying and managing network configurations in complex and dynamic environments. However, it is of most importance to take into consideration the costs involved into creating new networks or performing modifications in the existing ones. As stated in [41] "NETCONF is a stable standard for writing network configurations, with outstanding features for automating sequences



of configurations and driving out the costs associated with manual manipulation of devices.”

In conclusion, NETCONF is a protocol associated with YANG data modeling language suitable for different network equipment from different vendors. It provides scalability, flexibility and versatility. The overall advantages of NETCONF make it a powerful tool for network configuration and management, in the context of modern dynamic network environments. It is natural that as every other protocol NETCONF also has drawbacks and disadvantages. Disadvantages like the complexity compared to SNMP, the XML overhead and that is resource intensive cannot be ignored when designing a network system. In addition, this protocol doesn't provide support for Legacy Devices, it is not well suited for real-time applications due to inherent delays introduced by the xml-based communication and transactional nature of the operations. Protocols with lower overhead are usually suitable for real-time requirements. Lastly, the ecosystem around NETCONF, including community support, tools, and libraries, may not be as extensive as for some other protocols. This can affect the availability of resources and support for network administrators. Considering all the above, we conclude to the non-original thought that depending on the system designed and the needs of each network we have to prefer the most suitable protocol.

### 3.5 OpenFlow

OpenFlow is a communications protocol that gives access to the forwarding plane of a network switch or router over the network. It is used between controllers and forwarders in an SDN architecture. As mentioned in chapter two SDN main idea is to separate the forwarding plane from the control plane. In order to achieve this, the communication standard that was build between the controllers and the forwarders allowed the controllers to directly access and control the forwarders. This separation allows a more sophisticated way of traffic management using ACLs (Access Control Lists) and routing protocols.

OpenFlow originated from the Clean Slate Program of Stanford University. This program considered how the Internet could be redesigned with a "clean slate", and aimed to change the network infrastructure that was slightly out of date and difficult to evolve. In 2006, Martin Casado, a student from Stanford University, led a project on network security and management. The project attempted to use a centralized controller to allow network administrators to easily define security control policies based on network flows and to apply these security policies to various network devices, thereby implementing security control over the entire network communication. Inspired by this project, professor Nick McKeown — the director of the Clean Slate Program — and his team found that if the data forwarding and routing control modules of traditional network devices were separated, a centralized controller could be used to manage and configure various network devices through standard interfaces. This would result in more possibilities for the design, management, and use of network resources, thereby facilitating network innovation and development. Therefore, they put forward the concept of OpenFlow and

published a paper entitled "OpenFlow: Enabling Innovation in Campus Networks" in 2008, introducing the principles and application scenarios of OpenFlow in detail for the initial time. The first version of OpenFlow, OpenFlow 1.0 was introduced in December 2009. Due to the huge interest OpenFlow attracted an organization was created in 2011 Open Networking Foundation. ONF is a user-led organization dedicated to promotion and adoption of software-defined networking (SDN) that manages the standardization of OpenFlow. Founders of this organization were huge companies like Microsoft, Google and Facebook as well as various research institutions that implemented and researched the protocol.

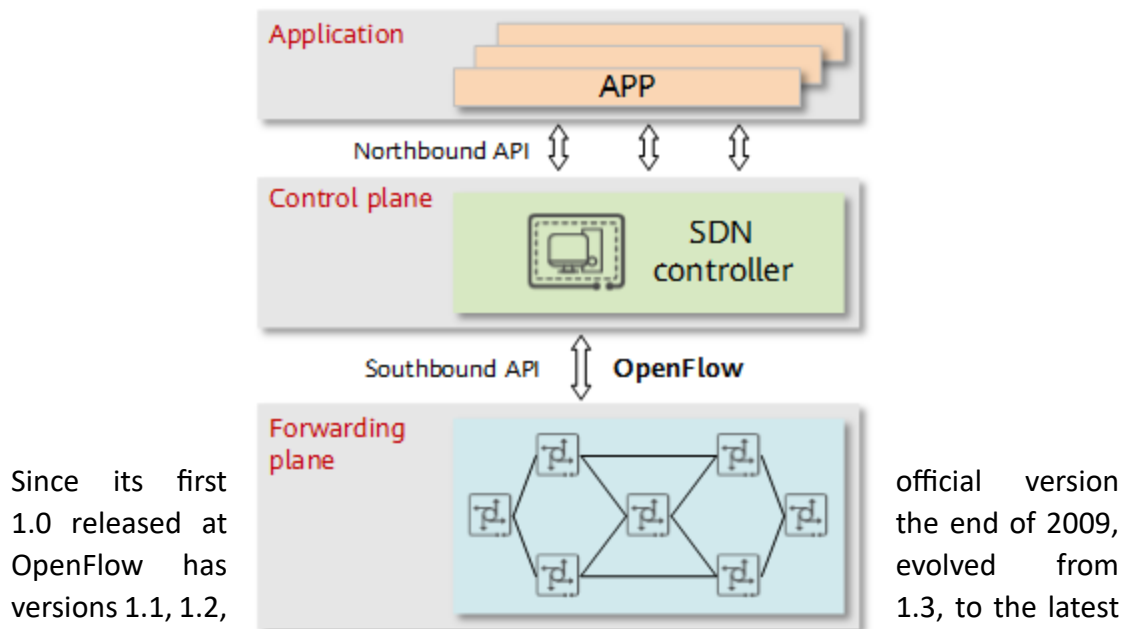


Figure 22 OpenFlow in the SDN architecture [42]

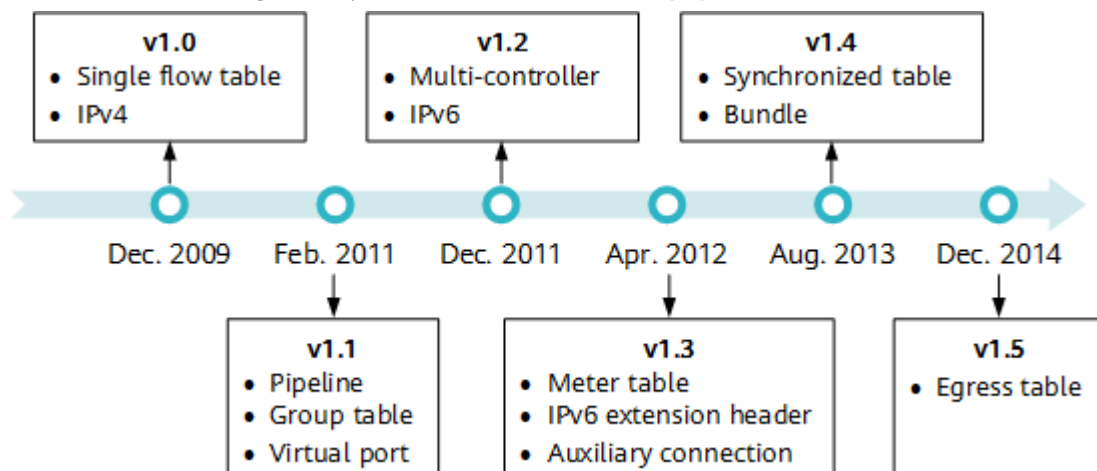


Figure 23 Evolution of OpenFlow [42]

version 1.5. Currently, versions 1.0 and 1.3 of OpenFlow are most widely supported and applied.

The importance of OpenFlow can already be understood from the attention it attracted from the early years of development and from the companies involved. This protocol played a crucial role in developing the SDN architecture, according to which almost all nowadays networks comply with. It revolutionized the Internet as we know it and led to the development of communication systems according to the SDN technology.

### 3.5.1 OpenFlow Architecture and Operations

The OpenFlow architecture consists of a controller, an OpenFlow switch and a secure channel. The OpenFlow protocol defines the interface between an OpenFlow Controller and an OpenFlow switch. The OpenFlow protocol allows the OpenFlow Controller to instruct the OpenFlow switch on how to handle incoming data packets.

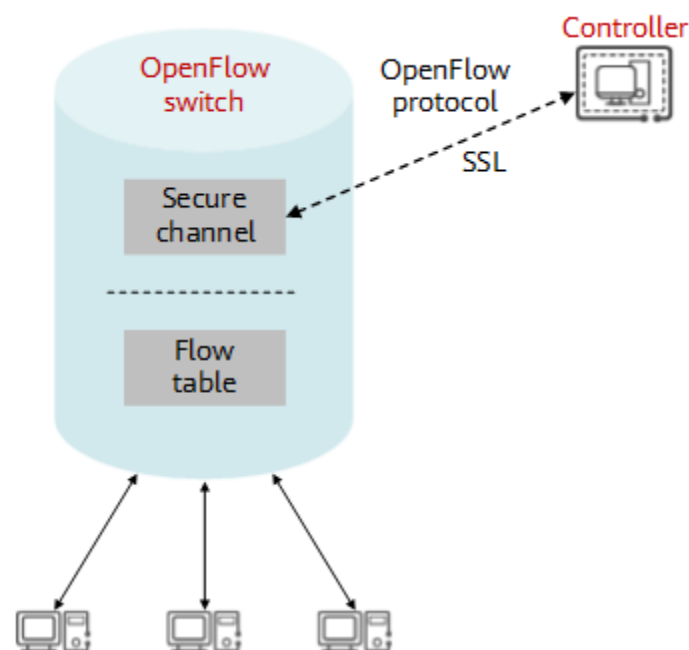


Figure 24 OpenFlow architecture [42]

After defining the architecture, it is essential to define each element of this architecture.

#### **OpenFlow Controller:**

It is the thinking mind of the SDN architecture and is located at the control layer. The controller's job is to instruct data forwarding through the OpenFlow protocol. Mainstream OpenFlow controllers are classified into two categories: open-source controllers and vendor-developed commercial ones. The widely used open-source controllers include NOX, POX, and OpenDaylight. Huawei's iMaster NCE controllers are commercial ones.

### OpenFlow Secure Channel:

A secure channel is established between the controller and an OpenFlow switch. Through this channel, the controller controls and manages the switch, and receives feedback from the switch. The messages exchanged over the OpenFlow secure channel must comply with the format specified by the OpenFlow protocol. The OpenFlow secure channel is usually encrypted using Transport Layer Security (TLS), but may be run directly over TCP in plain text in OpenFlow 1.1 and later versions. The following OpenFlow messages are transmitted over the channel: [42]

- Controller-to-Switch message: is sent by the controller to the OpenFlow switch to manage or obtain the OpenFlow switch status.
- Asynchronous message: is sent by the OpenFlow switch to the controller to update network events or status changes to the controller.
- Symmetric message: is sent without solicitation by either the OpenFlow switch or the controller. It is mainly used to set up a connection and detect whether the peer is online.

### OpenFlow switch:

Core component of the OpenFlow network architecture, mainly responsible for forwarding at the data layer. This switch can either be a physical or a virtual one (OpenvSwitch). The two types of switches based on their support of OpenFlow are:

- Dedicated OpenFlow switch: Standard OpenFlow device that supports only OpenFlow forwarding. The switch processes all traffic that passes through it in OpenFlow mode, and cannot perform Layer 2 or Layer 3 forwarding on the traffic.
- OpenFlow-compatible switch: Supports both OpenFlow and layer 2/3 forwarding. This type is usually a commercial switch that has the feature of supporting flow tables and secure channel.

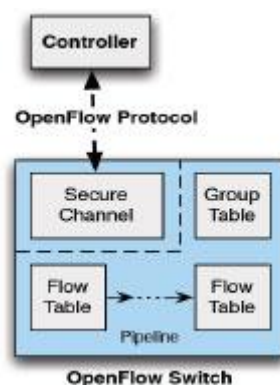


Figure 25 OpenFlow Protocol [45]

An OpenFlow switch forwards packets entering the switch based on the flow table, which contains a set of policy entries instructing the switch on how to process traffic. Flow entries are generated, maintained, and delivered by a controller. The OpenFlow switch may be programmed to identify and categorize packets from an ingress port based on a various packet header field; Process the packets in various ways, including modifying the header; Drop or push the packets to a particular egress port or to the OpenFlow Controller.

**Flow Entry:**

Network devices such as switches and routers that don't support OpenFlow, forward the data based on the locally saved Layer 2 MAC address forwarding table, Layer 3 IP address routing table and transport layer port numbers. OpenFlow switches forward the data according to the flow tables that contain information for all layers of the network. The entries in a flow table are flexible combinations of keywords and actions. Each flow entry in an OpenFlow flow table consists of match fields and a set of instructions applying to matching packets. When receiving a data packet, an OpenFlow switch parses and matches the packet header against match fields in the flow entries, and executes the corresponding instruction if a match is found. The flow entry structure varies according to the OpenFlow versions.

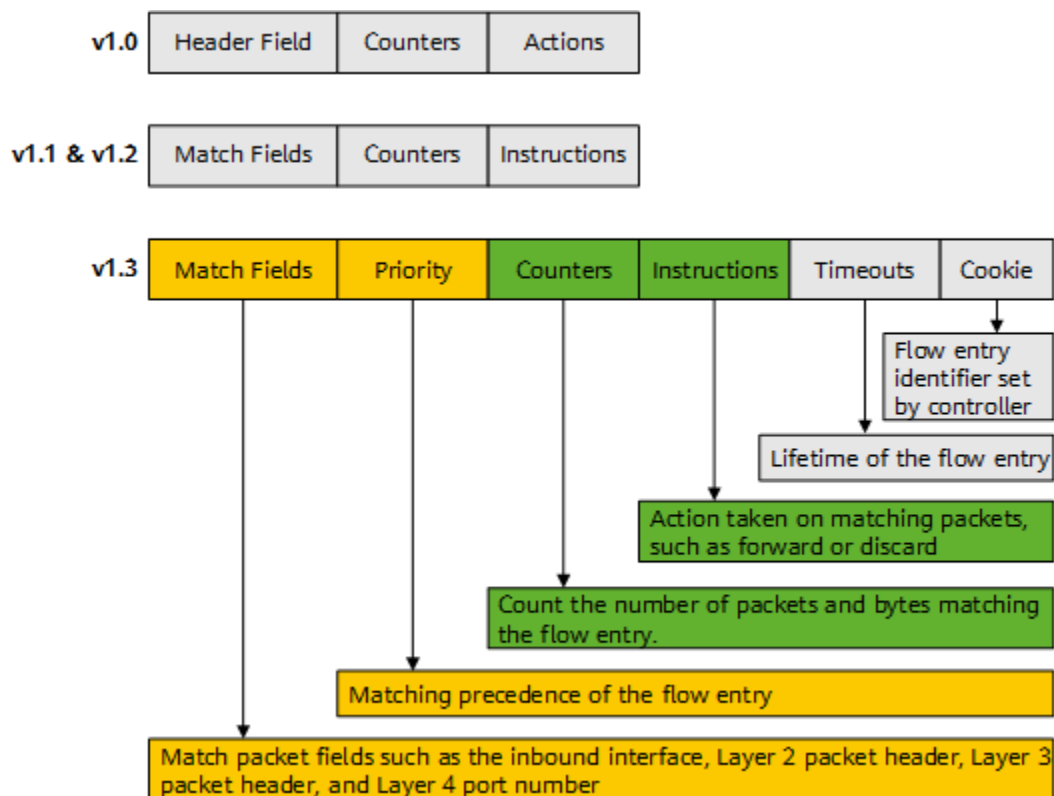


Figure 26 Flow entries in different OpenFlow versions [42]

According to OpenFlow specifications document for version 1.0 [47], each flow table entry contains:

- **Header fields** to match against the packet

- **Counters** to update for matching packet
- **Actions** to apply to the matching packet

The header value of a packet is either specific (which matches specific header in the flow table) or “ANY” (which matches every header). So in fact headers are important for matching the packet, counters for statistics which is something that the controller takes into serious consideration and off course actions that have to happen with the packet.

Switch designers are free to implement the internals in any way convenient provided that correct functionality is preserved. For example, while a flow may have multiple forward actions, each specifying a different port, a switch designer may choose to implement this as a single bitmask within the hardware forwarding table.

The matching procedure is explained in the flowcharts below:

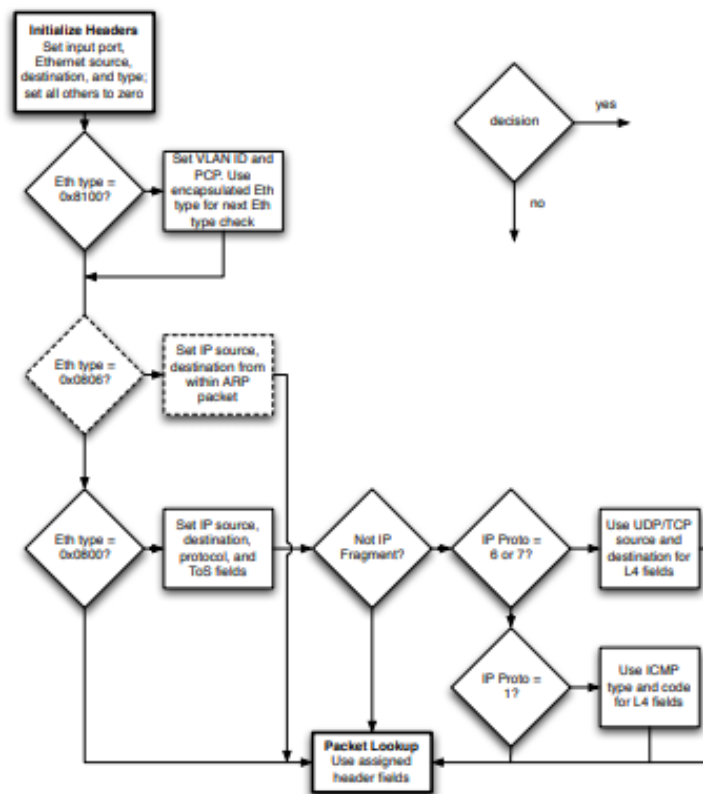
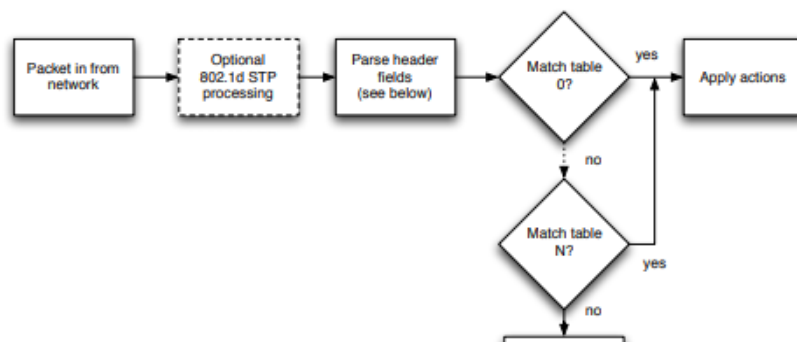


Figure 27 Parsing procedure for headers [47]

On receipt of a packet, an OpenFlow Switch performs the functions shown in Figure 28. Header fields used for the table lookup depend on the packet type as described below (and shown in Figure 28).

- Rules specifying an ingress port are matched against the physical port that received the packet.
- The Ethernet headers as specified in OpenFlow version 1.0 specifications [47] are used for all packets.
- If the packet is a VLAN (Ethernet type 0x8100), the VLAN ID and PCP fields are used in the lookup.
- (Optional) For ARP packets (Ethernet type equal to 0x0806), the lookup fields may also include the contained IP source and destination fields.
- For IP packets (Ethernet type equal to 0x0800), the lookup fields also include those in the IP header.
- For IP packets that are TCP or UDP (IP protocol is equal to 6 or 17), the lookup includes the transport ports.
- For IP packets that are ICMP (IP protocol is equal to 1), the lookup includes the Type and Code fields.
- For IP packets with nonzero fragment offset or More Fragments bit set, the transport ports are set to zero for the lookup.

**Multi-level flow table pipeline processing:**

OpenFlow v1.0 uses a single flow table to match packets. This implementation is simple, but the flow table becomes quite large as various policies are configured to implement increasingly complex network requirements. This makes management of the control plane more difficult, and poses higher requirements on hardware. OpenFlow v1.1 and later versions support multi-level flow tables and pipeline processing. When a packet arrives at a switch, matching starts from the flow table with the smallest sequence number, and may continue to additional flow tables in the pipeline. Multi-level flow tables can not only implement complex processing on data packets, but also reduce the length of a single flow table and therefore improve the entry lookup efficiency.

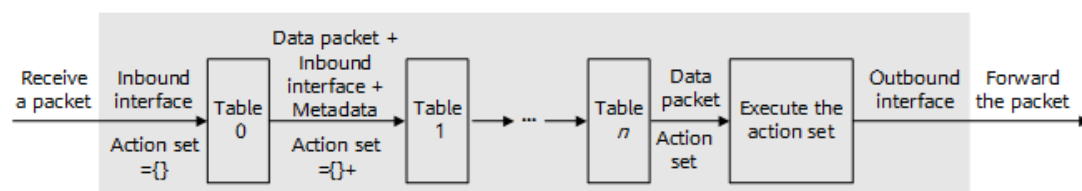


Figure 29 multi-level flow table processing [42]

### Flow table delivery:

Flow tables in OpenFlow can be delivered via two modes:

- In proactive mode, the controller delivers the flow table information to the switches
- In reactive mode, when a switch does not find a flow entry match any packet received sends a message to the controller informing him of the situation. The controller determines how the packet will be forwarded and sends the flow table entry to the switch. In addition, the switch will not maintain all entries. It obtains entries from the controller to serve the generated traffic and deletes the ones that are aged out. The procedure is described in OpenFlow specification paper [47] as:

“Each flow entry has an idle timeout and a hard timeout associated with it. If no packet has matched the rule in the last idle timeout seconds, or it has been hard timeout seconds since the flow was inserted, the switch removes the entry and sends a flow removed message. In addition, the controller is able to actively remove entries by sending a flow message with the DELETE or DELETE\_STRICT command. Like the message used to add the entry, a removal message contains a description, which may include wild cards”

#### 3.5.2 Advantages and presence SDN ecosystem

As addressed in previous paragraphs, OpenFlow is considered a very important part of SDN paradigm. The way this protocol revolutionized the network control and administration as well as the cost of network infrastructure development and maintenance are some of the benefits of this technology. In order to determine the role of OpenFlow in the SDN ecosystem it is important to highlight the key advantages and evaluate the open-source implementations.

The key advantages and optimizations of OpenFlow in network performance are demonstrated in the following key points:

**Centralized Control:** Centralized control of network traffic is enabled through OpenFlow, providing administrators with a complete supervision of the entire network. This enables efficient traffic management, resource allocation, and troubleshooting.

**Real-time Traffic Optimization:** With OpenFlow, administrators can dynamically route traffic based on real-time conditions. Analysis of the network's performance metrics, can lead to the redirection of traffic to less congested paths, optimizing bandwidth utilization and minimizing latency.

**Flexibility and Scalability:** OpenFlow's flexible architecture provides easy adaptation to changing network requirements. Businesses grow and evolve and OpenFlow enables scalability by providing a programmable environment, ensuring that networks can meet future demands without having to replace their equipment.

**Improved Security:** Network administrators gain granular control over traffic flows. This allows for the implementation of security policies and the identification and mitigation of potential threats in real-time. By providing rules over and over through



feedback from the switches the controllers are able to detect some threats and minimize the risk of attacks. In addition, OpenFlow uses an SSL channel in the controller-switch communication.

**Integration with Existing Infrastructure:** This protocol is designed to integrate with existing network infrastructures, enabling organizations to leverage their current investments while enhancing network performance. This makes the adoption of OpenFlow a cost-effective solution for network optimization. In fact according to “The wall street journal” many telecommunication enterprises aimed to use SDN and OpenFlow in order to save costs and advance their network.



Figure 30 The Wall Street Journal about AT&T in 2014 [49]

In year 2019, the global SDN market was valued \$9.86 billion.

The implementation of OpenFlow in network systems came hand to hand with SDN. The role of OpenFlow in SDN is described in some key points. After all, OpenFlow provided standardized interface between control and data plane. Network automation through OpenFlow became a reality through the enabling of automation in management tasks, traffic engineering, security policies and resource allocation. Complex networks operations are now simplified minimizing human error. In addition, organizations can meet SLA (Service-Level Agreement) commitments by allowing for granular policy enforcement dependent on service requirements. In simple words, with OpenFlow resources like bandwidth and latency can be allocated towards certain critical applications, minimizing performance issues. Another aspect of SDN very crucial that has been influenced by the implementation of OpenFlow is network virtualization. According to Wikipedia [50], “in computing, network virtualization is the process of combining hardware and software network resources and network functionality into a single, software-based administrative entity”. In other words, the

physical resources are used in a way where the SDN controller takes advantage of them so it provides certain QOS depending on the demands. OpenFlow's aid was crucial in this aspect of SDN.

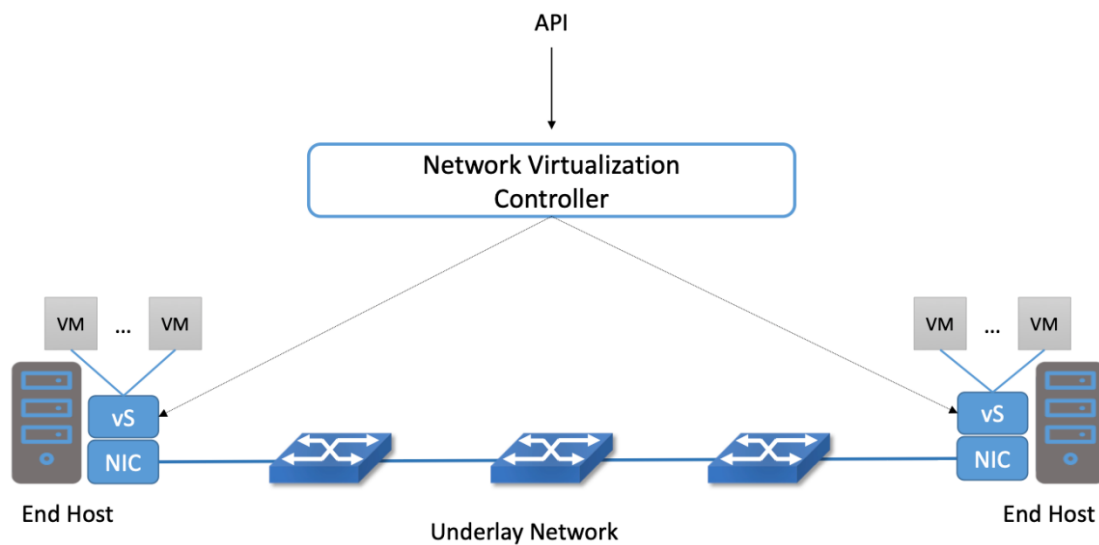


Figure 31 Basic Network Virtualization [51]

Another important aspect of OpenFlow in the implementation of SDN is the space it provided for experimentation and innovation. In an open and programmable platform many researchers could develop their own solutions and experiments without having to replace the network equipment. Last but not least, not only researchers didn't need to buy new equipment to experiment but also organizations. The fact that OpenFlow is a standard that ensures interoperability between different vendors' networking devices is called Vendor Neutrality. In that way Organizations can choose the equipment they want selecting from various suppliers, which promotes competition and flexibility.

### 3.5.3 Practical Implementations of OpenFlow and Challenges

As described in previous paragraph OpenFlow is the protocol that allows the Controller and the OpenFlow switch to communicate in a standardized way in order to separate the control from the data plane. It is logical to deduce that in order for that to happen, there have to be some deployments on the side of the SDN controller, some others on the side of the vSwitch and some that are more focused on the part of packet forwarding. The nature of OpenFlow is open-source, which allows a lot of implementations and deployments. In this paragraph we will refer to a set of implementations of OpenFlow from the three aspects mentioned above: SDN controller, vSwitch, packet forwarding. Indigo, Floodlight, and P4 Agent are software components related to OpenFlow and software-defined networking (SDN). Each plays a specific role in the OpenFlow ecosystem:

- **Indigo:** Indigo is an open-source software-defined networking (SDN) stack developed by the Open Networking Foundation (ONF). It provides a framework for building OpenFlow-based switches. It supports OpenFlow specifications and enables the deployment of OpenFlow-compliant switches on various

hardware platforms. Indigo includes both the OpenFlow agent, responsible for communication with an SDN controller, and the data plane forwarding elements. This project has been used as the basis for various OpenFlow switch implementations. Indigo is open-source supporting OpenFlow on a range of physical switches. By leveraging hardware features of Ethernet switch ASICs, Indigo supports high rates for high port counts, up to 48 10-gigabit ports. Multiple gigabit platforms with 10-gigabit uplinks are also supported. These exceed the limits of NetFPGA or pure-software OpenFlow implementations. Indigo firmware is actively used in many campus deployments (at Stanford and several other schools), in at least one OpenFlow startup, a 20-switch conference network deployment and a 32-switch data center deployment. The current release is based on the OpenFlow Reference Implementation from Stanford and currently implements all OpenFlow 1.0 features.

The Indigo software distribution is available in two ways:

- As pre-built firmware images (recommended for starting users)
- As a source distribution in a VM (recommended for advanced users)

Indigo supports multiple user interfaces including both a web-based configuration UI and a CLI accessible via telnet or ssh. These interfaces allow the configuration of the control interface (including behavior when the controller connection is lost) and monitoring of the ports and flow table.

More advanced users can download the Indigo Open Development System (IODS), a source release of the Indigo software. Users can modify the web server, command-line interface, some OpenFlow processing logic, and even add additional programs. IODS enables extensions in both C and Lua (a Python-like scripting language). The CLI and backend of the on-board web server are written in Lua and can be modified by the user.

- **Floodlight:** Floodlight is an open-source SDN controller platform developed in Java. It serves as the control plane in an SDN architecture, managing communication between the SDN controller and OpenFlow-enabled switches. It provides a modular and extensible platform for building SDN applications. Developers can create applications to customize network behavior through the Floodlight API. It also supports various network applications, including load balancing, network virtualization, and traffic engineering. Floodlight is widely used in research, educational, and commercial environments for SDN experimentation and deployment.
- **P4 Agent:** P4 (Programming Protocol-Independent Packet Processors) is a language designed for describing the forwarding behavior of network devices. The P4 Agent is a component that enables the deployment and management of P4-programmable devices within an SDN environment. It allows network operators to define and customize packet processing behavior in the data plane of network devices. The P4 Agent works in conjunction with P4-programmable switches, enabling them to interpret and execute P4 programs. It also facilitates the integration of P4-programmable devices into an SDN

architecture, providing a way to control and program these devices dynamically.

These components work together to implement and manage SDN environments based on the OpenFlow protocol. Indigo, Floodlight, and P4 Agent represent different aspects of the SDN ecosystem. Indigo provides an OpenFlow stack for building switches, Floodlight serves as an SDN controller for managing network devices, and P4 Agent facilitates the deployment and control of P4-programmable switches. Together, they contribute to the flexibility, programmability, and centralized control that are key characteristics of SDN architectures. Their open-source nature encourages collaboration and innovation in the development and deployment of SDN solutions.

Even though OpenFlow opened up a new window towards SDN and networking in general there are still challenges that have to be faced. Most of these challenges involve security, scalability and the limited support of complex policies [58],[59]. Another issue that has to be dealt is the scalability as the number of switches and flow entries increase. The centralized control model can become a bottleneck when managing a large-scale network with numerous devices and flow rules. Lastly, the complexity in Programming and Debugging is also an important challenge that has to be tackled. The complexity of defining and managing flow rules requires a certain level of skill and can lead to operational challenges.

In conclusion, OpenFlow is a protocol that revolutionized the network architecture and the implementation of SDN. The fundamentals of SDN and virtualization lie with the development of OpenFlow and the projects that implemented this protocol. The open-source nature of this standardization led to the heavy research and the adaptation of most organizations, that saw their profits go up due to the benefits it provided. The scientific and economic benefits of this technology became very crucial to the development of networking as it is today and will probably have an impact on the networking definition in the future generations. However, technology is a very dynamic field that evolves. OpenFlow is considered old news now even though a lot of organizations still use this protocol. P4 seems to be the future of SDN while this essay is being written, however the ever changing technological and economic environment will determine the next best thing. As in every protocol studied and addressed in this essay, OpenFlow has to be used according to the needs of the system that is being engineered and developed.

## 4. The Interface Developed

Having reviewed some of the most important network protocols used in control of network elements in Chapter 3 and defining the cause that led into the need of developing a solution for our SoC in chapter 2, a detailed roadmap of the interface build will be presented in this chapter. The methodology and the results of the outcome of this research will be presented in detail, as well as some limitations that came along with this solution.

### 4.1 Methodology

The chip provided by IHP institute is a Xilinx ZCU111 chip that has been programmed by the institute to have a specific behavior according to the content of some registers. Specifically, the developers in the institute managed to load the FPGA with a software that can be identified by a Linux operated pc with a specific set of commands with a specific set of parameters. The SoC is connected to a separate power supply and a USB port to the Linux operated PC. The SoC can also be detected with a Windows system, however the commands and the drivers in order to access the registers needed did not meet our demands and proceeded with the Linux OS. The developers in the institute provided the team with a “map” of the registers and a set of instructions on how to read and write on these registers, as well as a set of illustrations that gave us the capability to recognize the status of the equipment. The status of the FPGA is presented in the Figures below.

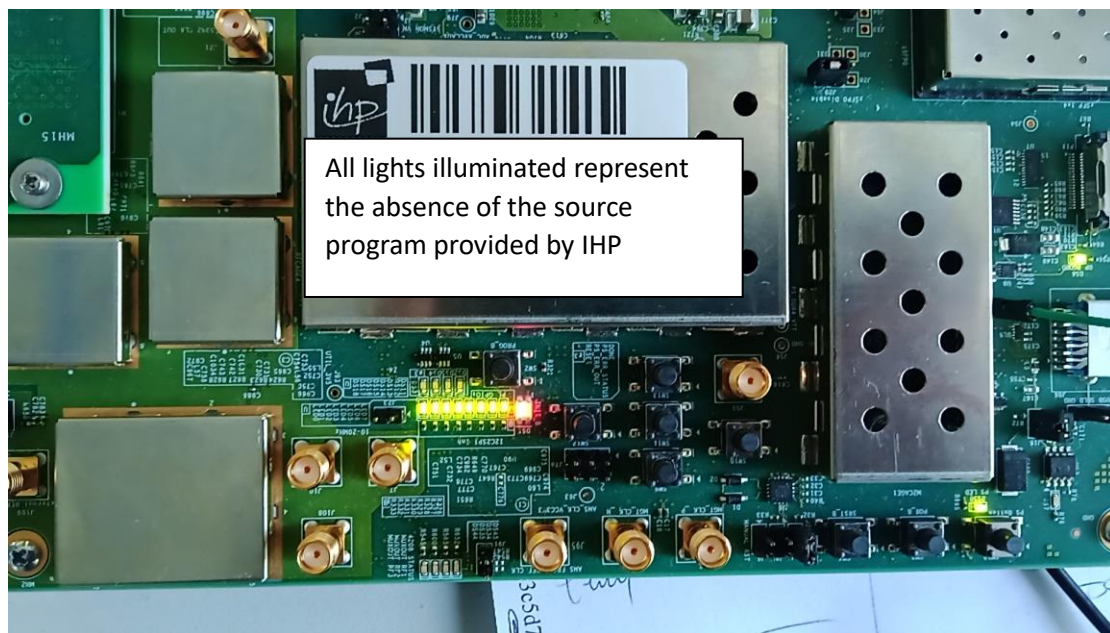


Figure 32 No program installed on the FPGA



Figure 33 FPGA in good operation

Also, the roadmap provided by institution described all the element of the registers and their meaning. The association between the bit sequence input of the registers and their meaning is provided in the table below.

Table 3 Registers Roadmap

Register Addr.	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Reset Value
0x00	Not used	Not used	Not used	Not used	Ch.3 enable	Ch.2 enable	Ch.1 enable	Ch.0 enable	0x0F
0x01	Not used	Not used	Not used	Not used	Not used	Not used	Modulation: "00": BPSK "01": QPSK "10": QAM16 "11": reserved		0x02
0x02	Frame format and settings: TBD								0x00
0x03	Not used	Not used	Not used	Not used	Not used	Not used	Not used	Not used	0x00
0x04	Not used	Not used	Not used	Not used	Not used	Not used	Not used	Not used	0x00
0x05	Not used	Not used	Not used	Not used	Not used	Not used	Not used	Not used	0x00
0x06	Not used	Not used	Not used	Not used	Not used	Not used	Not used	Not used	0x00
0x07	Not used	Not used	Not used	Not used	Not used	Not used	Not used	Not used	0x00

In order to access the FPGA and make changes to the functionality of it, without damaging the source code included is through two specific commands.



First of all, administrative rights had to be given to the user that will engage with the registers. Second, the port responsible for communicating with the relevant entities are USB serial port 2 9600. So in order to establish communication, at first the user had to initialize the port by providing the command

*“su sudo*

*screen /dev/ttyUSB2 9600”*

Apart from the initialization of the port this command is responsible for monitoring the good operation of the FPGA. If the connection is stable and in good condition (not overloaded from many commands running in the background) the screen command will initiate a window in which whatever character typed will be mirrored to the open terminal after being processed through the serial port. Upon initialization and check of status the operations needed to be performed are read and write in the registers mentioned above. In order to achieve this hexdump command is used. Hexdump gives the user the ability to access the registers. However, the command itself is not enough and has to have a set of parameters that complement the action at hand. The command that initiates the read right procedure is

*“hexdump -ve '1/1 "%3.2x"' /dev/ttyUSB2”*

This command enables the read/write procedure and has to stay active in the background. In another terminal a set of commands and parameters are sent towards the enabled for read and write FPGA. These commands and parameters are exhibited as a set of examples as shown in the table below.

*Table 4 Examples of commands and the register output*

<i>printf '\x01\x00' &gt; /dev/ttyUSB2</i>	Read register address 0x00
<i>printf '\x01\x01' &gt; /dev/ttyUSB2</i>	Read register address 0x01
<i>printf '\x01\x02' &gt; /dev/ttyUSB2</i>	Read register address 0x02
<i>printf '\x01\x07' &gt; /dev/ttyUSB2</i>	Read register address 0x07
<i>printf '\x02\x00\x55' &gt; /dev/ttyUSB2</i>	Write to register address 0x00, value equal to 0x55
<i>printf '\x02\x01\x66' &gt; /dev/ttyUSB2</i>	Write to register address 0x01, value equal to 0x66
<i>printf '\x02\x03\xff' &gt; /dev/ttyUSB2</i>	Write to register address 0x03, value equal to 0xFF

All these consisted the starting point of the research at hand. Having in mind the way of communicating with the FPGA and the tools as well as the desired architecture we were able to proceed to the engineering of the interface. In the next two sub chapters we will point out the interface design and implementation as well as the testing and validation.

#### 4.1.1 Interface Design and Implementation

The interface under research is part of a bigger movement in telecommunication systems. The paradigm at hand is “5G Complete” and the architecture involves this interface as a southbound interface. Having this in mind, as well as the tools provided the development of such interface wouldn’t be a hardware device that would translate signals towards the Thz Node. As part of a bigger picture, the interface had to be able to adapt and get feedback from higher level controllers. A software that is able to interact with the node, on a platform that is able to connect to another network from which it will receive orders and to which it will send information. That is the main principle of the interface developed.

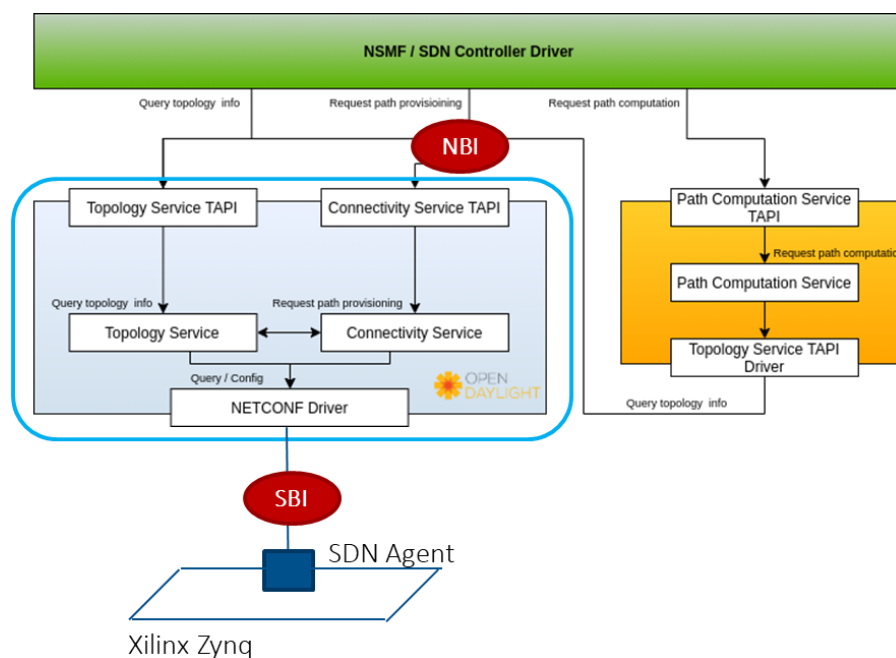


Figure 34 Integration of Thz Node in 5G complete architecture [60]

As shown in Figure 34 the researched interface will provide the coupling between the Thz Node and the OpenDaylight [61]. The interface specifications demand that it will be able to communicate with the Thz Node, read and write information on the registers, communicate with the Southbound interface in a way that it can provide the information of the registers and wait for orders as to what to write and to which one. In addition, since the tools we have apply on Linux OS, the interface must be compatible and run optimally on Linux. Also, the received coupling with the southbound interface must be applicable in Linux. Combining all the above the decision was to write a bash script on Ubuntu OS. That way we effectively take advantage of the standard tools already provided and at the same time, due to the nature of Open Source software of Linux, leave room for more implementations that will communicate with the Upper part of the southbound interface.

In order to make this work we decided to break the expected outcome in smaller procedures. The outcome of this thinking is presented in the Figure below.



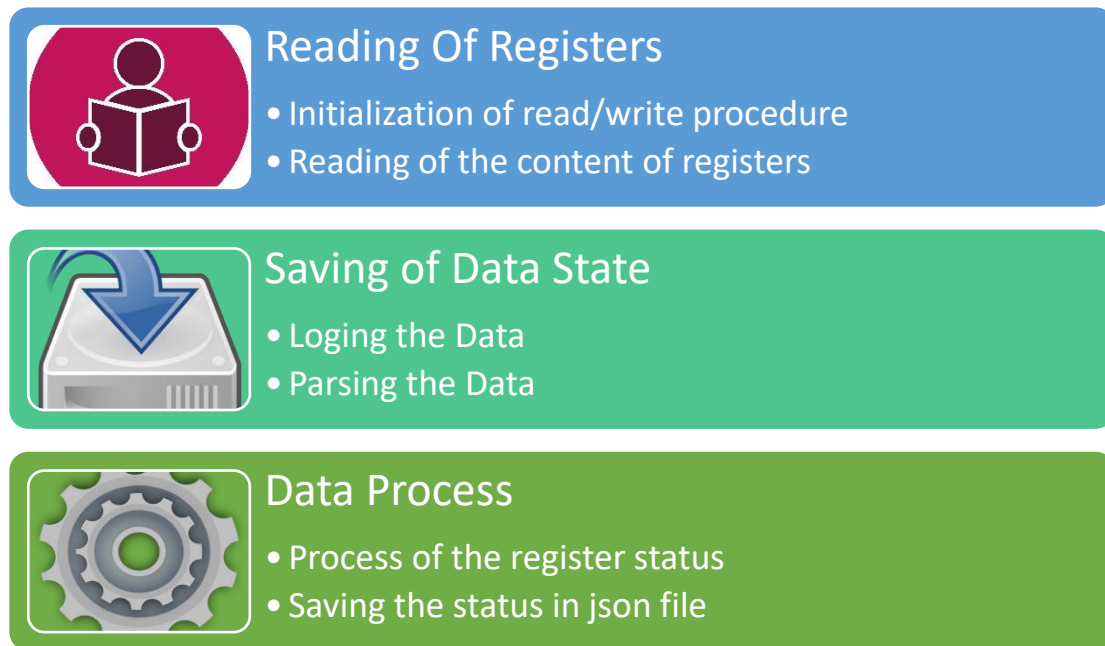


Figure 35 Problem Breakdown Analysis

Since the analysis of the problem was performed, a decision was taken: each one represents a specific function that can be called upon. However simple it might look each step had a set of challenges to be tackled in order to deliver the final product.

- **Reading of the Registers:**

In order to initiate the read/write procedure with the serial port 2 of the FPGA, a command of initialization had to be given. This tool has already been delivered from the institute. However, in order to read and write this procedure had to be in the background and proceed towards the read state. In order for the agent to establish a connection with the Node and then start the reading procedure, a specific tactical latency of 2 seconds was intentionally added so the system could use the time window for parameter retrieval. In addition, through a lot of tries a big latency in the response was detected. Due to the hexdump initialization commands given without terminating them, the FPGA started lagging. This meant that cleanup is mandatory in order to keep the good operation of the SoC. As a result, the final function called “total” was able to initiate the connection, read the register content (the writing procedure is something not provided in this version but will be implemented in future work) and cleanup the system memory.

- **Saving of Data State:**

This is the most challenging part of the interface. The SoC is unable to understand anything else than 2 specific commands with the exact parameters. Any other command didn’t apply and returned nothing. In addition, the Linux operators that play a crucial role in input-output and saving into file operations were not working on the SoC, as it was unable to “understand” them. With only 2 commands and the risk of damaging the core software of the SoC, making trial and error unavailable, the way of

saving the data seemed impossible. The solution found was an out of the box thought: don't expect the data from the FPGA. Since communication was very limited and the results only showed on the terminal of the operating system, we stopped monitoring the FPGA and started monitoring the terminal that was communicating. A "script" command with input the "save data function" was used in order to monitor the outcome of the terminal on a log text file. In this way, for the first time, we had the data of the registers in a text file along with a lot of the terminal events as clutter. So now the data parsing had to begin. The data we needed was a set of numbers. A series of "grep" commands along with specific parameters were used, as well as some temporary files that contained part of the information. In order to understand the procedure, a flow diagram below is presented.

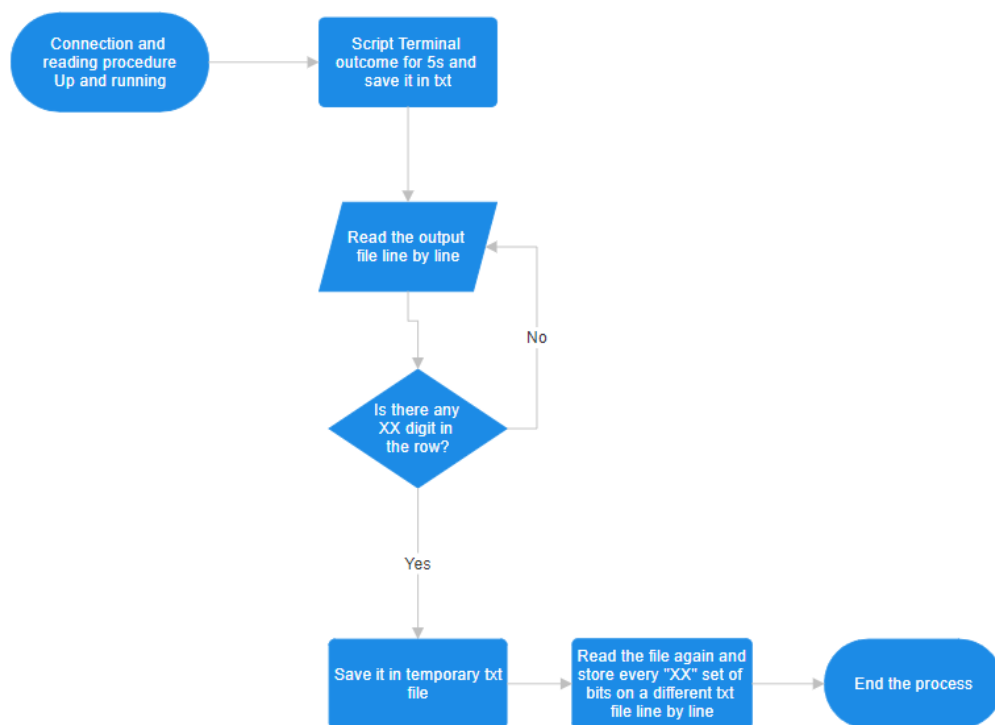


Figure 36 Flow Diagram of data saving function

The outcome of the Saving Data Function is a simple text file. Each line in this file is a set of bits. Each set represents a register content.

- **Data Processing:**

After successfully saving the status of each register line by line in a simple text file, a data process function was created. The concept was to read this file line by line and store each line in an array. Then the data of the array was compared in order to show whether the SoC is online or offline. If the registers didn't contain anything meant that we were unable to retrieve anything, which actually meant the Node is offline. Any other set of digits was presented and then formatted in a json file. A flow chart of the function is presented below.

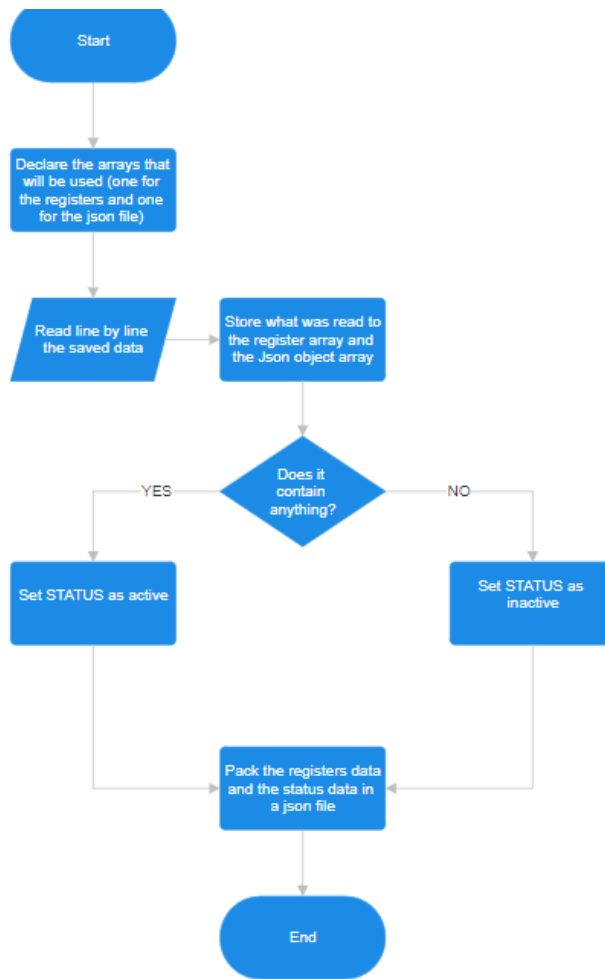


Figure 37 Process Function Flow chart

The format of the json file will allow the agent to send it to the other side of the interface directly to a controller or indirectly to a broker that will be accessed by the controller. In such way the system will now know that the Thz node is online and will be able to redirect traffic. As a result, a part of the SDN architecture has been implemented. The SDN controller has now access to the status of the Thz Node. No specific or exclusive technology has been used according to the SDN paradigm. The Thz Node is available for status retrieval and with small implementations in the read function, can be controlled. This implementation can help promote the SDN paradigm including the Thz Node and open a new window in the telecommunication technology.

#### 4.1.2 Testing and Validation

Testing and Validation of the interface happened step by step in order to be sure that the result of each function is valid and can be used. Each function is dependent on the outcome of the previous one as defined in the sub chapter above. Through testing and validation came the realization of clean up procedures as well as the intended delays

between commands and smaller functions. The equipment used for testing was a commercial laptop with the characteristics presented below.

**Processor:** AMD Ryzen 3 3250U with Radeon Graphics 2.60 GHz

**Installed RAM:** 8,00 GB (5,88 GB used)

**System version:** Operating system 64 bit, processor x64

The time needed for all the procedures to finish and have the final json file is 5 seconds due to the intended delays for synchronization between the commands. The interface is successful and has capabilities that will be implemented in next versions. The output json file is destined towards the other side of the interface enabling the coupling between the two sides.

The code of all the bash scripts as well as the main program that implements the functions are included in Appendix A of this essay.

## 5. Conclusions and Future work

The outcome of this research achieved the development of an interface between a Thz Node and an SDN controller, following the SDN paradigm that was studied throughout the extent of this essay. The interface developed gave the administrator of a network system the capability to communicate with an FPGA that was unable to understand anything more than two specific commands. In that way a resource addition to the 5G complete paradigm was implemented signaling the use of mmWaves and Thz transmission bringing 6G one step closer.

Developing a new technology is thrilling and gives the researchers pride and a feeling of fulfilment. However, this is only the tip of the iceberg. The literature that was studied gave the insight needed to develop technology and imagine the next step, and the step after the next one. Throughout the extent of this essay the writer became familiar with FPGAs and the upcoming innovations in this domain. SoCs seem to play a crucial role in cutting edge technology and especially in the field of network engineering. In addition, the programmability and the flexibility that they offer seem to be in line with the SDN paradigm which is ruling the telecommunication systems as we know them, and most probably as we will meet them in the nearest future. SDN paradigm that has been introduced in 2009 has impacted the way we develop and use networks. Characteristics such as virtualization and remote control of network elements became a key into developing new technology and ensuring specific QoS according to the needs of the application. Flexibility, scalability and vendor independency are just a few of the advantages that this paradigm has provided. Even though SDN is fascinating as an idea and as a usable paradigm the scientific community was aware of the importance of remote controlling elements and started introducing protocols to establish that since the 1980s. Network element control protocols such as SNMP, SCPI, TL1 and NETCONFIG have been developed and researched way before

the adaptation of SDN. While studying all these protocols the idea of sending messages between the agent and the controller became clear. Each one with its own advantages and disadvantages which the network engineer would have to take into consideration when he performs a design of the network. These protocols are by far not obsolete and are used up to this date. For example, the usage of SCPI is widely spread through the design and engineering of test benches in most industries. In addition, NETCONF is part of the 5G complete implementation as shown in Figure 34. Of course, the usage of OpenFlow is still very important and relevant as a protocol that separated the control from the data plane. The various enrolments of this protocol according to the usage that 'll have, either on the controller side or the vSwitch side, helped develop more the idea of SDN. While writing this essay the definitions and necessities of south bound and north bound interfaces became clear. With a clear view of all the protocols mentioned above, as well as the idea of SDN the decision on how to move towards the integration of this interface for the Thz Node in order to implement it to the paradigm of 5G complete became all the more feasible. As mentioned in Chapter 4, the methodology that was adopted always had in mind the way of SDN and the use of remote calls, as well as the fact that this interface must both be useful to both sides of this "coupling". The idea of interacting only with the Node without saving the status makes the system incapable of being controlled, therefore unable of being used in an SDN ecosystem. Lessons were learned through the literature review as well as through the procedure of implementing the interface. The Thz Node is now able to be controlled, since we can obtain the parameters of the registers and determine whether its is online or offline, but also information about the modulation adopted from this Node. Although it is important to know the status of this device, it is also important to have the ability of controlling it. In order to do so, the next version of this interface will implement the parameters for writing inside the Node by actuating remotely. More future work includes the usage of this script periodically without being invoked by a remote controller, but having the information of the Node available through a broker somewhere in the cloud. Security optimizations are also an aspect that has to be addressed in future work. It is also important to understand the limitations. This script interface is available every 5s. This delay is intentional and has a huge impact on the synchronization of the commands in order to draw crucial and original data from the registers. By all means, the concept of remotely controlling an innovative device like this Thz Node, at least at the south bound interface part has been achieved and even though future work can be performed, the pleasure of integrating technologies and making a step ahead towards progress is a feeling that few can indulge. Tackling the obstacles and delivering is important, but so is inspiration to achieve more of what has already been delivered. The future is closer than we thought.

## 6. List of figures

Figure 1 Simplified block diagram of SDN[4] .....	7
Figure 2Xilinx FPGA[7] .....	8
Figure 3 FPGA architecture.....	9
Figure 4 FPGA used for our research.....	10
Figure 5 Basic SDN architecture[9] .....	11
Figure 6 SDN overview[9] .....	12
Figure 7 SNMP visualization [14] .....	14
Figure 8 Standard MIB tree[15] .....	16
Figure 9 SNMP architecture[16] .....	16
Figure 10 SNMP principle of communication [10].....	17
Figure 11 SNMP message types[14] .....	19
Figure 12 Back of instrument providing connections for remote control [29].....	22
Figure 13 Simplified SCPI commad tree.....	24
Figure 14 Design Flow of the integrated environment [28] .....	25
Figure 15 Complete documentation, libraries and examples from manufacturer [27].....	26
Figure 16 LAN transport of TL1 data [31] .....	27
Figure 17 Simplified representation of message types .....	28
Figure 18 Sample of autonomous message [36] .....	29
Figure 19 Integration of TL1 with Java.....	33
Figure 20 NETCONF protocol layers [38] .....	35
Figure 21 NETCONF Interaction process [37] .....	36
Figure 22 OpenFlow in the SDN architecture [42].....	42
Figure 23 Evolution of OpenFlow [42].....	42
Figure 24 OpenFlow architecture [42].....	43
Figure 25 OpenFlow Protocol [45].....	44
Figure 26 Flow entries in different OpenFlow versions [42].....	45
Figure 27 Parsing procedure for headers [47] .....	46
Figure 28 Packet Flow in OpenFlow switch [47] .....	46
Figure 29 multi-level flow table processing [42] .....	47
Figure 30 The Wall Street Journal about AT&T in 2014 [49] .....	49
Figure 31 Basic Network Virtualization [51] .....	50
Figure 32 No program installed on the FPGA .....	53
Figure 33 FPGA in good operation.....	54
Figure 34 Itegration of Thz Node in 5G complete architecture [60].....	56
Figure 35 Problem Breakdown Analysis .....	57
Figure 36 Flow Diagram of data saving function .....	58
Figure 37 Process Function Flow chart.....	59

## 7. List of Tables

Table 1 SNMPv3 security levels[24].....	21
Table 2 TL1 Command parameters[34] .....	29
Table 3 Registers Roadmap.....	54
Table 4 Examples of commands and the register output .....	55

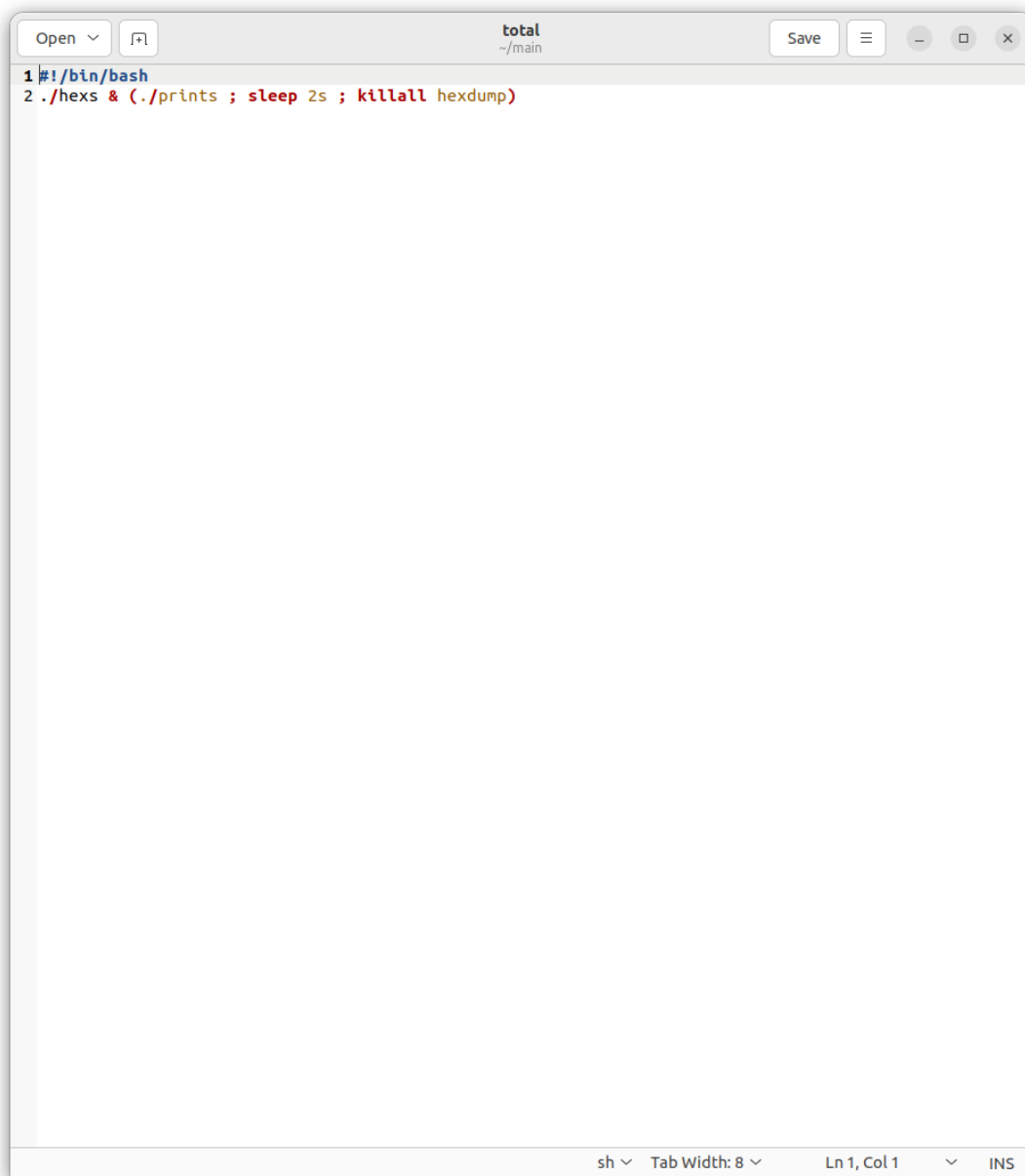
## 8. References

1. [https://en.wikipedia.org/wiki/Software-defined\\_networking](https://en.wikipedia.org/wiki/Software-defined_networking)
2. <https://www.techtarget.com/searchnetworking/definition/SDN-controller-software-defined-networking-controller>
3. Mittal, Sangeeta. (2018). Performance Evaluation of Openflow SDN Controllers. 10.1007/978-3-319-76348-4\_87.
4. [https://en.wikipedia.org/wiki/Field-programmable\\_gate\\_array](https://en.wikipedia.org/wiki/Field-programmable_gate_array)
5. <https://www.ni.com/en/shop/electronic-test-instrumentation/add-ons-for-electronic-test-and-instrumentation/what-is-labview-fpga-module/fpga-fundamentals.html>
6. <https://gr.mouser.com/new/xilinx/xilinx-sp701-eval-kit/>
7. <https://www.ihp-microelectronics.com/fields-of-activity/radio-frequency-broadband-communication-systems>
8. [https://opennetworking.org/wp-content/uploads/2013/02/TR\\_SDN\\_ARCH\\_1.0\\_06062014.pdf](https://opennetworking.org/wp-content/uploads/2013/02/TR_SDN_ARCH_1.0_06062014.pdf)
9. [https://en.wikipedia.org/wiki/Simple\\_Network\\_Management\\_Protocol](https://en.wikipedia.org/wiki/Simple_Network_Management_Protocol)
10. <https://docs.oracle.com/cd/E19253-01/817-3000/introduction-3/index.html>
11. <https://www.geeksforgeeks.org/simple-network-management-protocol-snmp/>
12. <https://www.manageengine.com/network-monitoring/what-is-snmp.html>
13. <https://networkwalks.com/snmp-simple-network-management-protocol/>
14. [https://docs.iconics.com/V10.96/GENESIS64/Help/Apps/WBDT/SNMP/SNMP\\_Management\\_Information\\_Base\\_MIB\\_.htm](https://docs.iconics.com/V10.96/GENESIS64/Help/Apps/WBDT/SNMP/SNMP_Management_Information_Base_MIB_.htm)
15. <https://www.geeksforgeeks.org/snmp-enumeration/>
16. [https://docs.iconics.com/V10.96/GENESIS64/Help/Apps/WBDT/SNMP/SNMP\\_Introduction.htm](https://docs.iconics.com/V10.96/GENESIS64/Help/Apps/WBDT/SNMP/SNMP_Introduction.htm)
17. [RFC 6353](#) Section 10
18. D. Levi; P. Meyer; B. Stewart (April 1999). "RFC 2573 – SNMP Applications". Internet Engineering Task Force. doi:10.17487/RFC2573. {{cite journal}}: Cite journal requires |journal= (help)
19. ^ Jump up to: ^ "SNMP Inform Requests". Cisco. Retrieved 2011-12-09. {{cite journal}}: Cite journal requires |journal= (help)
20. ^ "Understanding the SNMP Implementation in JUNOS Software". Juniper Networks. Retrieved 2013-02-11. {{cite journal}}: Cite journal requires |journal= (help)
21. "RFC Search Detail: Standards Track snmpv2 RFCs". The RFC Editor. Retrieved 2014-02-24.
22. Douglas R. Mauro & Kevin J. Schmidt. (2001). *Essential SNMP* (1st ed.). Sebastopol, CA: O'Reilly & Associates.
23. <https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/snmp/configuration/xs-3se/3850/snmp-xe-3se-3850-book/nm-snmp-snmpv3.pdf>
24. [https://snmp.com/snmpv3/snmpv3\\_intro.shtml](https://snmp.com/snmpv3/snmpv3_intro.shtml)
25. [https://en.wikipedia.org/wiki/Standard\\_Commands\\_for\\_Programmable\\_Instruments](https://en.wikipedia.org/wiki/Standard_Commands_for_Programmable_Instruments)
26. [https://www.rohde-schwarz.com/au/driver-pages/remote-control/remote-programming-environments\\_231250.html](https://www.rohde-schwarz.com/au/driver-pages/remote-control/remote-programming-environments_231250.html)
27. Balaji, Aravind & Sasikumar, Subramanian & K, Dr. Ramesh. (2021). SCPI based integrated test and measurement environment using LabVIEW. IOP Conference Series: Materials Science and Engineering. 1045. 012036. 10.1088/1757-899X/1045/1/012036.
28. <https://goughlui.com/2021/03/28/tutorial-introduction-to-scpi-automation-of-test-equipment-with-pyvisa/>
29. <https://www.dpstele.com/network-monitoring/alarm/tl1/what-is.php>
30. <https://www.dpstele.com/dps/protocol/2001/jan-feb/index.php>
31. [https://en.wikipedia.org/wiki/Transaction\\_Language\\_1](https://en.wikipedia.org/wiki/Transaction_Language_1)



32. <https://www.ibm.com/docs/en/netcoolomnibus/8?topic=acquisition-issuing-commands>
33. <https://www.ibm.com/docs/en/netcoolomnibus/8?topic=acquisition-tl1-command-structure>
34. S. S. Chavan and R. Madanagopal, "Generic SNMP proxy agent framework for management of heterogeneous network elements," 2009 First International Communication Systems and Networks and Workshops, Bangalore, India, 2009, pp. 1-6, doi: 10.1109/COMSNETS.2009.4808873.
35. <https://www.dpstele.com/blog/how-to-understand-tl1-protocol.php>
36. <https://support.huawei.com/enterprise/en/doc/EDOC1100112399/3707247/introduction-to-netconf>
37. <https://en.wikipedia.org/wiki/NETCONF>
38. RFC 6241
39. <https://support.huawei.com/enterprise/en/doc/EDOC1100126923/a8fda134/overview-of-netconf>
40. Caroline Chappell, "Creating the Programmable Network: The business Case for NETCONF/YANG in Network Devices", 2013 Heavy Reading
41. <https://info.support.huawei.com/info-finder/encyclopedia/en/OpenFlow.html>
42. <https://en.wikipedia.org/wiki/OpenFlow>
43. <https://www.sdxcentral.com/networking/sdn/definitions/what-the-definition-of-software-defined-networking-sdn/what-is-openflow/>
44. <https://noviflow.com/the-basics-of-sdn-and-the-openflow-network-architecture/>
45. <https://medium.com/@fiberoptics/openvswitch-and-openflow-what-are-they-whats-their-relationship-d0ccd39b9a5c>
46. <https://opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.0.0.pdf>
47. <https://utilitiesone.com/role-of-openflow-in-software-defined-networking-sdn-for-traffic-management>
48. Αθανασία Αλωνιστιώτη, Διαφάνειες μαθήματος «Δικτύωση Βασισμένη στο Λογισμικό» , 2021 ΕΚΠΑ.
49. [https://en.wikipedia.org/wiki/Network\\_virtualization](https://en.wikipedia.org/wiki/Network_virtualization)
50. <https://sdn.systemsapproach.org/netvirt.html>
51. <https://floodlight.atlassian.net/wiki/spaces/Indigo/overview>
52. <https://www.oreilly.com/library/view/software-defined-networking-with/9781783984282/b67ed638-71a5-46ac-a81a-9d7e69506a99.xhtml>
53. <https://github.com/floodlight/floodlight/blob/master/README.md>
54. <https://github.com/p4lang/p4ofagent>
55. da Costa Cordeiro, W.L., Marques, J.A. & Gaspary, L.P. Data Plane Programmability Beyond OpenFlow: Opportunities and Challenges for Network and Service Operations and Management. J Netw Syst Manage 25, 784–818 (2017). <https://doi.org/10.1007/s10922-017-9423-2>
56. Wenjuan Li, Weizhi Meng, Lam For Kwok,
57. A survey on OpenFlow-based Software Defined Networks: Security challenges and countermeasures, Journal of Network and Computer Applications, Volume 68, 2016, Pages 126-139, ISSN 1084-8045, <https://doi.org/10.1016/j.jnca.2016.04.011>.
58. <https://opennetworking.org/p4/>
59. 5G-COMPLETE deliverable D4.1, "Initial report on the development of advanced signal processing tools for fronthaul and midhaul services", 2022.
60. <https://opendaylight.org/about>

## Appendix A: Interface script code



The image shows a terminal window titled "total" with the current directory set to "~/main". The window contains two lines of code:

```
1 #!/bin/bash
2 ./hexs & (./prints ; sleep 2s ; killall hexdump)
```

The status bar at the bottom of the terminal window displays "sh", "Tab Width: 8", "Ln 1, Col 1", and "INS".

The image shows a window titled "hexs" with a path of "~ /main". The window contains a terminal-like interface with the following text:

```
1 #!/bin/bash
2 hexdump -ve '1/1 "%3.2x"' /dev/ttyUSB2
3
```

At the bottom of the window, there is a status bar with the following information: "Loading file "/home/kapo/main/hexs" ...", "sh", "Tab Width: 8", "Ln 1, Col 1", and "INS".

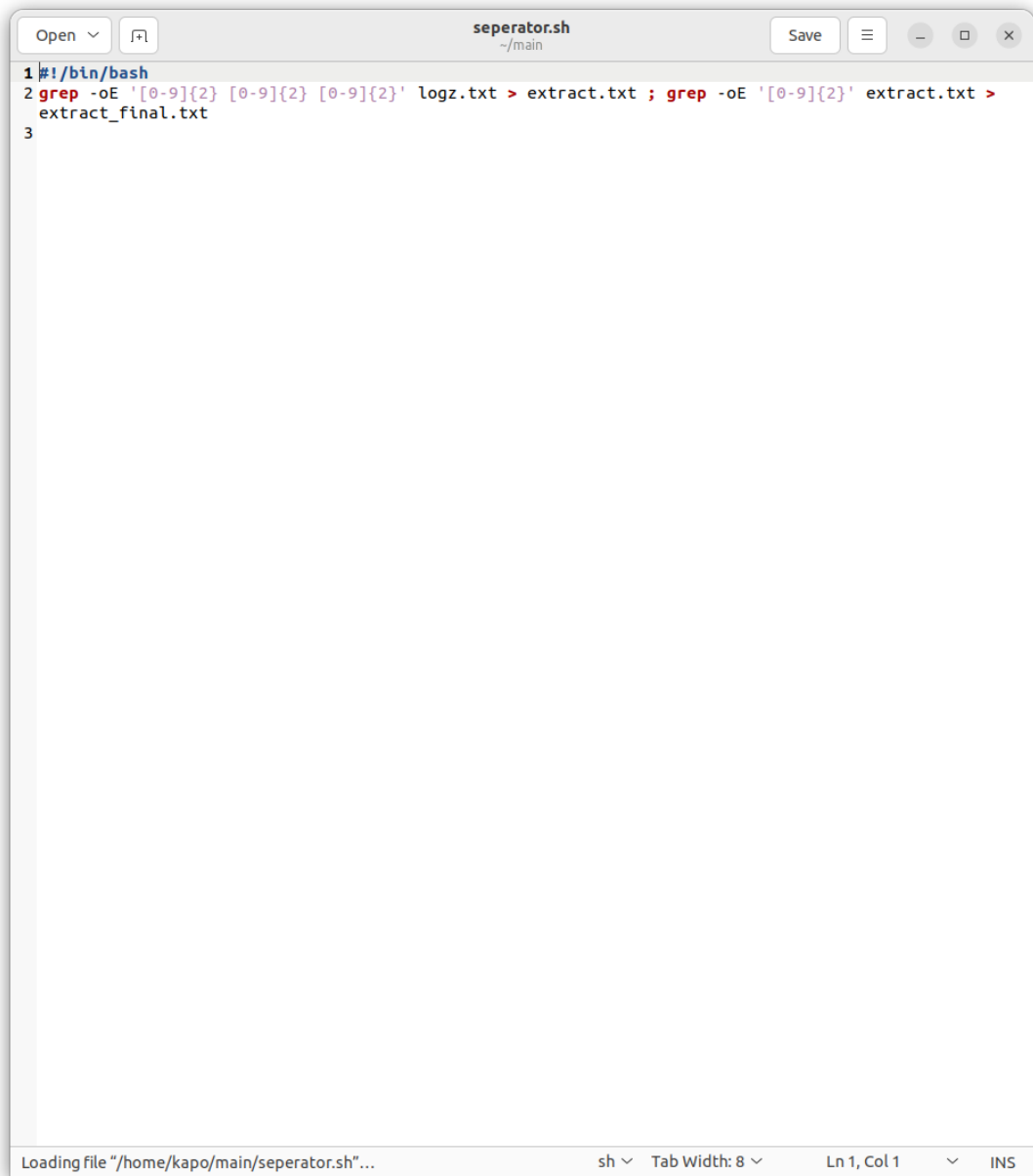
```
Open [icon] prints -/main Save [icon] [icon] [icon]
1 #!/bin/bash
2 sleep 1
3 printf '\x01\x00' > /dev/ttyUSB2
4 printf '\x01\x01' > /dev/ttyUSB2
5 printf '\x01\x02' > /dev/ttyUSB2
6

Loading file "/home/kapo/main/prints"... sh Tab Width: 8 Ln 1, Col 1 INS
```

The image shows a code editor window with the following content:

```
1 #!/bin/bash
2 timeout 5s script -f logz.txt < ./total
```

The window title is "test1.sh" and the current directory is "~/main". The status bar at the bottom indicates the shell is "sh", tab width is 8, and the cursor is at line 1, column 1.



```
seperator.sh
~/main
Open Save
1 #!/bin/bash
2 grep -oE '[0-9]{2} [0-9]{2} [0-9]{2}' logz.txt > extract.txt ; grep -oE '[0-9]{2}' extract.txt >
  extract_final.txt
3

Loading file "/home/kapo/main/seperator.sh"... sh Tab Width: 8 Ln 1, Col 1 INS
```

```
Open  [icon]  jason.sh  Save  [icon]  [icon]  [icon]
1 #!/bin/bash
2 #define lines table
3 declare -a regs
4 declare -A json_obj
5 #assign at each register the
6 readarray -t regs < extract_final.txt
7 json_obj[0]="reg1:" "${regs[0]},"
8 json_obj[1]="reg2:" "${regs[1]},"
9 json_obj[2]="reg3:" "${regs[2]},"
10 if [ -v "${regs[0]}" ]; then
11 STATUS="active"
12 else
13 STATUS="inactive"
14 fi
15 echo json_data="{${json_obj[0]} \"status\": \"${STATUS}\"}" > jsonj.json

Loading file "/home/kapo/main/jason.sh"...  sh  Tab Width: 8  Ln 9, Col 36  INS
```

```
main.sh
~/main
Open [icon] Save [icon] [icon] [icon] [icon]
1 ./test1.sh
2 ./seperator.sh
3 ./jason.sh
Loading file "/home/kapo/main/main.sh"... sh Tab Width: 8 Ln 1, Col 1 INS
```