THE FUTURE OF SNMP – REST BASED API MONITORING

CHIA JIAN WEI

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

BORANG PENGESAHAN STATUS LAPORAN

JUDUL: [**THE FUTURE OF SNMP – REST BASED API MONITORING**]

SESI PENGAJIAN: [2022 / 2023]

Saya: CHIA JIAN WEI

mengaku membenarkan tesis Projek Sarjana Muda ini disimpan di Perpustakaan Universiti Teknikal Malaysia Melaka dengan syarat-syarat kegunaan seperti berikut:

1. Tesis dan projek adalah hakmilik Universiti Teknikal Malaysia Melaka.
2. Perpustakaan Fakulti Teknologi Maklumat dan Komunikasi dibenarkan membuat salinan unituk tujuan pengajian sahaja.
3. Perpustakaan Fakulti Teknologi Maklumat dan Komunikasi dibenarkan membuat salinan tesis ini sebagai bahan pertukaran antara institusi pengajian tinggi.
4. * Sila tandakan (✓)

_____	SULIT	(Mengandungi maklumat yang berdarjah keselamatan atau kepentingan Malaysia seperti yang termaktub di dalam AKTA RAHSIA RASMI 1972)
_____	TERHAD	(Mengandungi maklumat TERHAD yang telah ditentukan oleh organisasi / badan di mana penyelidikan dijalankan)
✓		
_____	TIDAK TERHAD	

_____ (TANDATANGAN PELAJAR)

Alamat tetap: 33, JALAN JAYA PUTRA 3/2

TAMAN JP PERDANA, 81100 JOHOR BAHRU, JOHOR

Tarikh: 25 SEPTEMBER 2023

FAIZAL

_____ (TANDATANGAN PENYELIA)

PROFESOR MADYA DR MOHD FAIZAL ABDOLLAH

Nama Penyelia

Tarikh: 25/9/23

CATATAN: * Jika tesis ini SULIT atau TERHAD, sila lampirkan surat daripada pihak berkuasa.

THE FUTURE OF SNMP – REST BASED API MONITORING

CHIA JIAN WEI

This report is submitted in partial fulfillment of the requirements for the Bachelor of Computer Science (Computer Networks) with Honours.

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY
UNIVERSITI TEKNIKAL MALAYSIA MELAKA

2023

DECLARATION

I hereby declare that this project report entitled

[THE FUTURE OF SNMP – REST BASED API MONITORING]

is written by me and is my own effort and that no part has been plagiarized

without citations.

STUDENT : _____ Date : 25 SEPTEMBER 2023

(CHIA JIAN WEI)

I hereby declare that I have read this project report and found

this project report is sufficient in term of the scope and quality for the award of

Bachelor of Computer Science (Computer Networks) with Honours.

SUPERVISOR : _____ Date : 25/9/23

(PROFESSOR MADYA DR MOHD FAIZAL ABDOLLAH)

DEDICATION

This research paper is sincerely dedicated to my supportive parents who encouraged and inspired me in conducting this study. They have never left my side throughout the process and gave me strength and hope when I thought of giving this up. They provided me a great sense of enthusiasm and perseverance in continuing this. This research is made possible with their love and assistance.

Moreover, I dedicate this research paper to this subject lecturer, Professor Madya Dr Mohd Faizal Abdollah, who constantly guiding and teaching me to make this study even better, to my family for cheering up for me, and to my friends who have helped me in finishing this project. I really appreciate your words of advice and in continuously giving me moral, and emotional support.

ACKNOWLEDGEMENTS

This work would not have been possible without the support of Universiti Teknikal Malaysia Melaka. I am especially indebted to Professor Madya Dr Mohd Faizal Abdollah, who have been supportive of my career goals and who worked actively to provide me with the protected academic time to pursue those goals.

I am grateful to all of those with whom I have had the pleasure to work during this and other related projects. Nobody has been more important to me in the pursuit of this project than the members of my family.

Each of the members of the PSM Committee has also provided me extensive personal and professional guidance and taught me a great deal about both scientific research and life in general. I would especially like to thank Professor Datuk Ts. Dr. Shahrin Bin Sahib @ Sahibuddin, my evaluator for this project.

As my teacher and mentor, I would like to mention Ts Mohd Ropi bin Abdollah, he has taught me more than I could ever give him credit for here, and I could even say he was the one who made me who I am today.

Most importantly, I would like to thank my parents, whose love and guidance are with me in whatever I pursue. They are supportive and a support throughout my life, and ultimately my biggest and most personal role models.

ABSTRACT

This paper examines the role of Simple Network Management Protocol (SNMP) as a de-facto protocol in the field of network monitoring system. This paper discusses in detail the transmission of SNMP protocol, how it works in the network layer and also the three different revision that exists. Despite this popularity, there is a genuine concern for the fundamental limitation of SNMP as a protocol, and there are a growing number of network monitoring system shifting towards alternative method, and one such system is RESTful API. This paper will explore the argument towards the shift away from SNMP to API based system, and compare the differences between the two approaches. This paper also aims to analyze the actual difference when implementing the system in the network level, such as the difference delay of messages, the size of packets and the available security parameter on both systems.

ABSTRAK

Kertas kerja ini mengkaji peranan *Simple Network Management Protocol* (SNMP) sebagai protokol de-facto dalam bidang sistem pentadbiran rangkaian. Kertas kerja ini membincangkan secara terperinci protokol penghantaran SNMP, fungsi yang tersedia dalam lapisan rangkaian dan juga tiga versi berbeza yang wujud. Walaubagaimanapun, terdapat kebimbangan terhadap SNMP sebagai protokol asas, dan terdapat semakin banyak sistem pentadbiran rangkaian yang beralih ke kaedah alternatif, dan satu sistem sedemikian adalah RESTful API. Kajian akan meneruskan penerokaan dan peralihan daripada SNMP kepada sistem berasaskan API, dan membandingkan perbezaan antara kedua-dua pendekatan. Kertas kerja ini juga bertujuan untuk menganalisis perbezaan di dunia nyata apabila melaksanakan sistem dalam peringkat rangkaian, seperti kependaman rangkaian mesej, saiz paket dan parameter keselamatan maklumat yang tersedia pada kedua-dua sistem.

TABLE OF CONTENTS

PAGE

LIST OF TABLES

PAGE

LIST OF FIGURES

PAGE

LIST OF ABBREVIATIONS

FYP	**-**	**Final Year Project**
SNMP	**-**	**Simple Network Management Protocol**
REST	**-**	**Representational State Transfer**
API	**-**	**Application Programming Interface**
AES	**-**	**Advanced Encryption Standard**
JSON	**-**	**JavaScript Object Notation**
YAML	**-**	**Yet Another Markup Language**
ASCII	**-**	**American Standard Code for Information Interchange**
MIB	**-**	**Management Information Base**
OID	**-**	**Object Identifier**
HTTP	**-**	**Hypertext Transfer Protocol**
IETF	**-**	**Internet Engineering Task Force**
IEEE	**-**	**Institute of Electrical and Electronics Engineers**
RFC	**-**	**Requests for Comments**
PDU	**-**	**Protocol Data Units**
UDP	**-**	**User Datagram Protocol**
USM		**User-based Security Model**
VACM		**View-based Access Control Model**
MD5		**Message-Digest Algorithm**
SHA		**Secure Hash Algorithm**
DES		**Data Encryption Standard**
IAB	**-**	**Internet Activities Board**
XML	**-**	**Extensible Markup Language**
SMI	**-**	**Structure Of Management Information**

COAP	**-**	**Constrained Application Protocol**
CBOR	**-**	**Concise Binary Object Representation**
HTTP	**-**	**Hypertext Transfer Protocol**
WWW	**-**	**World Wide Web**
HTTPS	**-**	**Hypertext Transfer Protocol Secure**
URI	**-**	**Uniform Resource Identifier**
URL	**-**	**Uniform Resource Locator**
QUIC	**-**	**Quick UDP Internet Connections**
gRPC	**-**	**gRPC Remote Procedure Calls**
TCP	**-**	**Transmission Control Protocol**
VM	**-**	**Virtual Machines**
TLS	**-**	**Transport Layer Security**
MTU	**-**	**Maximum Transmission Unit**
USM		**User-based Security Model**

UNIVERSITI TEKNIKAL MALAYSIA MELAKA
UTeM
اونيۏرسيتي تيکنيکل مليسيا ملاك
UNIVERSITI TEKNIKAL MALAYSIA MELAKA

CHAPTER 1: INTRODUCTION

1.1 Introduction

Monitoring and resource management has been dominated by the protocol Simple Network Management Protocol (SNMP) since 1988 from RFC 1065. Despite numerous revision and changes to the protocol, launching version 2, version 2c and even version 3, the protocol is over-engineered, not being fully utilized, and the over-extend configuration of the protocol leads to massive issues with proprietary compatibility issues and more importantly security issues.

Hence there is a need to pivot to protocols other than SNMP, and RESTful service are one of the architectural solutions for solving the underlying problem. It utilizes the most common protocols, more specifically HTTP(s) service as a stateless client-server service.

1.2 Problem Statement

One of the issues that plagues SNMP is the concern of privacy and security when sending the SNMP traffic. SNMP has undergone several revisions, with each one addressing the flaws of the previous version. The SNMPv2c, second revision brings some feature improvement, but it does not even support encrypted message. Even the implementation of SNMPv3 is considered not secure as it uses stateless protocol to secure the traffic, opting out the more secure challenge-handshake authentication protocol. Moreover, most of the hardware that runs SNMPv2c either

require firmware update, or some compatibility issue makes supporting SNMPv3 impossible.

This brings to the issue of interoperability and compatibility issue. Some network engineers opt to still use SNMPv2c because of the mentality “don’t change it if it works”, and even that the SNMPv3 adoption rate is slow because the different network device vendor does not have a consensus on how to implement the cipher due to lack of RFC that defined AES-192 and AES-256. The issue is exacerbated by the major equipment vendor over-extend the proprietary configuration which it makes the convoluted system that much harder to navigate.

1.3 Project Question

a) Why the REST API and the SNMP protocol are relevant when discussing about network monitoring?

b) What are the transmission methods, network traffic and available security parameters between the two REST API and SNMP?

c) What is the performance difference that concerns the network engineer that intends to evaluate the current system of SNMP API

1.4 Project Objective

a) To configure a basic RESTful API system alongside SNMP system that makes the transmission of network monitoring information possible.

b) To explain, compare and contrast the fundamental differences between the protocols, and how the differences is reflected on how the data is transmitted between devices

c) To test and investigate the performance between different protocols and how each configuration differences impact the monitoring process.

1.5 Project Scope

1. The project able to convey the functionality and overview of each of the protocols, why the protocols are relevant and how both of the SNMP and REST API works in brief.

2. The project also includes the exploration the differences between SNMP protocol and REST API. It highlights the difference between the two protocols in terms of transmission, traffic generation and also security parameters.

3. Based on the exploration, gather the data for each of the highlighted parameters, and make comparison for the both protocols, and make summary of each of the analyzed data.

1.6 Project Contribution

a) The project benefits system admin that wants to explore the alternative of SNMP, where some researchers argue that there should be a pivot of monitoring system towards alternative method such as REST API

b) The project helps to have a more understanding of the two protocols, the initial idea of the pivot, the differences and constraint of each method of monitoring for a more comprehensive decision in choosing a monitoring method.

c) Explore the difference concern of both the protocols, and be a reference for engineers in deciding the protocols, based on the relevant performance parameters.

CHAPTER 2: LITERATURE REVIEW AND PROJECT METHODOLOGY

2.1 Introduction

SNMP is an extremely useful tool for a network engineer as it is a well-established protocol supported by most devices in an IP network. However, the protocol is also notorious for the many limitations, such as the unreliability of UDP transport, poor SNMP agent implementation, outdated 32-bit counters and also the persistent security concerns despite there is already 3 revisions of the protocol.

Every problem is an opportunity for you to create a solution, as the scale of network infrastructure is growing in an enormous rate especially with the introduction of cloud-based system, the unresolved issue poses challenge in keeping up the demand of increasing connectivity. Few alternative ideas are starting to float around, and one of the more prominent ways to complement the function of SNMP protocol seems to utilize another well-defined and established protocol, HTTP to create a RESTful based system.

2.2 Simple Network Management Protocol (SNMP)

When the topic of Network Management System is discussed among the network administrator, The Simple Network Management Protocol (SNMP) is the most widely used protocol, or even the preferred choice for the management of IP-based networks and internets. SNMP is an official standard protocol and it is defined and governed by an organization called Internet Engineering Task Force (IETF). The IETF publishes Requests for Comments (RFCs), which are the specification for the widely used protocol that exist for devices in the internet to communicate. As of

2023, there are three version of SNMP, which are SNMPv1, SNMPv2c and SNMPv3, with the respective RFC as shown in Table 2.2.1 below.

SNMP Version	**Year**	**RFC Document**
SNMP Version 1 (SNMPv1)	1990	RFC1157 (Official Standard) RFC1155, 1212 (Definition language)
SNMP Version 2 (SNMPv2c)	1996	RFC1902, 1903, 1904, 1905, 1906,1907 RFC 1908 (Coexistence and transition)
SNMP Version 3 (SNMPv3)	2002	RFC3410, 3411, 3412, 3413, 3414, 3415 RFC 3416 (Coexistence and transition)

Table 2.2.1 SNMP version with corresponding RFC documents

Based on publication by W. Stallings in the IEEE Communication Magazine in March 1998, the SNMP protocol consists of three specifications, which are the protocol for exchanging information between management system and agents (protocol), the framework for format and storage of management information (MIB), and also the general-purpose management information objects or variables (data definition). At the time of publication, there is only two version of SNMP protocol. They author made a survey and concluded that the first version of SNMP is flawed when it gained widespread use, and the second version, SNMPv2 did not received the acceptance as anticipated by the protocol designers.

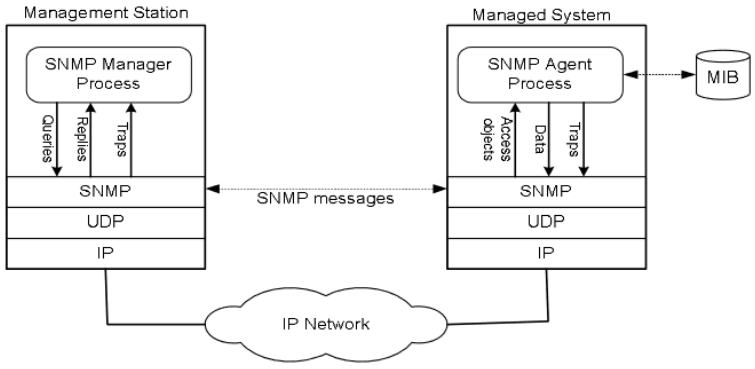

Figure 2.2.1 Overview of SNMP Interactions

Referring the Figure 2.2.1 above, the SNMPv1 framework describes the encapsulation of SNMP Protocol Data Unit (PDU) in the message between different entities and the distinction between application or protocol entities. (Case, Fedor, Schoffstall, & Davin, 1990) The document describes the protocol operation in PDU

on list of variable bindings. The basic operators are get, get-next, get-response, set-request and trap. The document also defines the layering of protocol on a connectionless transport service (or better known as UDP).

As the SNMP protocol becomes widespread, as mentioned above, the drawbacks of the first version become apparent especially the protocol is considered barebone and lacks functionality, and require changes for things such as transfer efficiency for it to continue being a viable protocol, hence SNMPv2c is proposed.

The SNMPv2c provides several changes and advantages over SNMPv1, one of the most important is the massive efficiency improvement with the introduction of GetBulk command and also changes to Get command. It is only possible for SNMPv1 to retrieve information from the table one row at a time and a tedious series of get/response transaction is required if the manager needs to retrieve the whole table. This command GetBulk in SNMPv2 could retrieve the whole table in one transaction, and even retrieve additional information from the same message. It is similar with the Get command, where the SNMPv1 agent will reject the command if even one value is missing. SNMPv2 introduces partial results to return, where it ignores the value that could not be retrieved. These two major changes improve the efficiency by reducing the exchange across network.

The SNMPv1 and SNMPv2c implements security in the form of community string, which are the cleartext password that the devices need to be allowed to exchange information when SNMP requests occur. However, the implementation is massively flawed and with the popularization of internet services, the security is inadequate moving forward, hence the next version is proposed, with the main focus being the enhancement of security.

The SNMPv3 architecture introduces the User-based Security Model (USM) and View-based Access Control Model (VACM) for message security and access control respectively. It also supports SNMP Engine ID identifier, is a unique identifier for SNMP entities, and is used to generate key for authentication. The model of SNMPv3 security is comprised of two parts, which are Authentication and

Encryption or Privacy. In RFC 2574, the security levels for the USM MIB are defined in three levels, as shown in Table 2.2.2 below. (Blumenthal & Wijnen, 1999)

Security Level	**Definition**
noAuthnoPriv	Communication without authentication and privacy
authNoPriv	Communication with authentication only. The protocol supported are MD5 and SHA
authPriv	Communication with authentication and privacy. Protocol for Auth: MD5 and SHA Protocol for Priv: AES and DES

Table 2.2.2 SNMPv3 Authentication and Encryption levels

In a survey of SNMP by Pallavi et al. (2017), the paper states SNMP has its demerits, it builds very complex software agents, and sometimes it reduces the bandwidth of the network. The paper describes in depth about the security issues that arise from the SNMP protocol itself, in their words, *"By enabling SNMP services it is easy to administrate any network adequately and productively yet enabling it will make a network defenseless to security attacks"*

The paper continues to advocate for the usage of SNMPv3, the third version of the SNMP. It highlights the few security mechanisms that exists in the particular revision such as strong privacy, view-based access control, authentication and integrity. It also briefly touches the introduction of 64-bit counter instead of the previous version of 32-bit counter.

2.3 The fall of SNMP and the transition towards alternative protocol

The problem of deploying SNMP as the internet standard is long in discussion and in 2002, an organization called IAB discussed the concerns about the protocol. (Schoenwaelder, 2002) One of the conclusions that they made from two papers was to investigate alternative network management technologies that take advantage of protocols such as XML or web service. The biggest benefit is that the protocol is a generic technology that was supported by many vendors on multiple different protocol, and was more well-established than even SNMP itself, and still dominating the web traffic even today.

XML as the gateway for SNMP has been researched by Choi & Hong (2002), As part of their research, they investigated the performance difference between the two protocols. The authors concluded that for their test set-up, the XML performed marginally better than the SNMP itself.

The research that was done by P. Aiko et al. (2004) also proves the point above where for individual retrieval SNMP is much efficient but it reverses when more object is required. The choice of encoding that they used in the testing, BER and XML have negligible effect and is not the determining factor in performances. The choice of encoding that are the most popular with the web technology are XML and JSON. There is different use case for the two different encodings but based on the information K. Alnafjan (2017) has gathered, XML is a great in type definition, schema similar to SNMP and it is longer in the market. JSON on the other hand is much programmer friendly, ease of serial or de-serialization, and most modern devices able to interpret JSON better. Each of them has their strengths and weaknesses and should consider the workload of the particular task before deciding either one.

The latest revision of the SNMPv3 protocol is still flawed even if the protocol was updated with security in mind, (Taha et al. 2021), as they created a lightweight script, scoured the whole internet network for SNMP traffic, and analyzed the detailed information that were gathered. Their proof-of-concept campaign fingerprinted more than 12 million devices and around 350k network routers, directly highlight the more fundamental issue of the SNMP protocol that still persists even with multiple revisions.

2.4 HTTP as a protocol

One of the arguably the most widely used application layer protocol that have ever existed in the Internet protocol suite is most probably the Hypertext Transfer Protocol (HTTP). This first version of HTTP/1 was finalized and ratified in 1996 under the RFC 1945 by the infamous Tim Berners-Lee from CERN. HTTP is the foundation of data communication for the World Wide Web (WWW). In essence the HTTP protocol is a stateless request-response protocol that exchange information between client and server using a reliable network transport protocol (TCP). HTTP uses the port 80 to communicate or port 443 for a secure variant of HTTP protocol called HTTPS, with the resources identified and located on the network by the Uniform Resource Identifiers (URI) scheme 'http' and 'https' using Uniform Resource Locator (URLs) as defined in RFC 3986.

As the popularity of this protocol exploded, there are multiple revisions of HTTP since the inception in 1996, with the introduction of HTTP/2 in March of 2012 that aims to improve upon the performance, latency and data compression while maintain a high-level compatibility with the older HTTP/1.1 protocol. The HTTP/2 specification was published under the RFC 7540 on May 14 2015 and became the de-factor standard for the data transfer in WWW.

The HTTP traffic by default is transmitted in plain text form, and as more service integrate with the HTTP protocol, especially more sensitive information such as banking industries are exploring the technology, the security aspect becomes one of the fundamental aspects for wider adoption of this protocol. Hence, a cryptographic protocol called Transport Layer Security (TLS) was introduced. This TLS protocol aims to provide confidentiality, integrity and authenticity (CIA trinity) through the use of RSA certificates in asymmetric encryption. The protocol that implements the TLS is identified as Hypertext Transfer Protocol Secure (HTTPs) with the URI starting with HTTPS, an extra 'S' compared to the plaintext variant of HTTP and

The internet traffic is increasing in an exponential manner, with low latency with high throughput being the focus in order to provide a seamless experience for everyday users, and some Internet Service Provider, more specifically based on one

blog post by Gigaspaces company even claims that a 100ms extra delay on the network potentially costs millions of dollars of losses.

2.5 Representational State Transfer (RESTful API) and RESTCONF

Representational state transfer (REST) is one of the predominant application integration mechanism or software architectural style over the Internet. (Bergmann, Bormann & Gerdes, 2020). REST architectural style is becoming popular in recent years due to its ease of implementation as API compared to Simple Object Access Protocol (SOAP) and XML-RPC (Wenhui, Yu, Xueyang & Chen, 2017). With the advent and growth of agile software development paired with the popularization of DevOps methodology, there is a genuine requirement for a simpler and faster iteration that could keep up with the change of demands, hence, API exists to bridge the gap and fill the role as the first choice, especially the capability of automating tasks such as testing and integration.

There is no formal definition of what REST is, but as introduced and defined in a 2000 doctoral dissertation by Fielding, R. T., the concept is that the server will respond with representation of a resource, (commonly in the form of HTML, XML, JSON or YANG), and the state of the system change will be based on the resources that contains the hypermedia link that can be followed. Fielding further clarifies in the 11th Joint Meeting on Foundations of Software Engineering (2017), acknowledging the lack of formal definition, and emphasized that the RESTful concept is a set of architectural style, rather than an architecture itself.

Based on the Fielding and multiple different peer-review, REST architectural style consists of six design principles or constraints:

1. *resources are identified by one resource identifier mechanism – URI schema is the most commonly used one at present;*
2. *resources have representations and representation metadata– a representation is considered a series of bytes that could be described by metadata;*

3. *only a few primitive operations/methods are available to operate on resources– these primitives have the same meaning for all the resources (i.e., are designed to operate exactly the same, no matter the target resource);*
4. *all interactions are stateless– all allowed primitives must receive complete requests and requests must be processed independently;*
5. *idempotent behavior– usage of caching techniques (through resource's metadata) and idempotent behavior are encouraged;*
6. *intermediate entities are encouraged– such entities could provide for proxy/caching techniques or could alter the requests and the responses.*

The key concept in REST is the resource itself, and this protocol is often confused with the HTTP itself due to the similarity of operations of functionality in CRUD (Create, Read, Update, Delete) such as PUT, GET, DELETE, POST. It does not help with the confusion where most developer migrate from SOAP and WS-* based RPC approach to REST in web services with the minimal change of perspective in application approach.

2.6 Critical review of current problem and justification

Choi M. and Hong J. has published several papers related to the design of XML-SNMP gateway. The duo investigated the performances differences of XML and SNMP by measuring the XML traffic as well as SNMP traffic in their set-up. Usage of resources such as CPU and network resources are also gathered and comparison of the factors shows that the XML system is comparable in small scale with negligible difference, and a clear benefit for XML compared to SNMP for larger data as the overhead of SNMP makes scaling more resource intensive. The system is similar in concept but this project uses the REST based system instead of XML.

In a paper published in June 2014 by Bergmann O. et. al, there is a study titled REST-based access to SMIv2-structured information on constrained devices. This study focuses on the IoT, specifically the constraint of IoT devices such as Low-Power and Constrained Application Protocol (CoAP) in implementing services especially SNMP. The predominant protocol for network management is SNMP,

however in a constraint environment the extra complexity of adding layers of protocol is not feasible. This paper explores the multiple existing technology in an alternative way for SNMP, which uses Concise Binary Object Representation (CBOR), utilizing Network Configuration Protocol or REST-based web technology to transfer the information through network. This paper put the emphasis on efficiency of data transfer such as payload size, with the goal of implementing it in a hyper-efficient way. The method of implementation in this particular research is similar with the current project, however with the wide range of devices and the exponential growth of processing power efficiency, this project will less focused on the efficiency of data encoding/decoding.

There is also another research about the SNMP and Web service by Ricardo Neisse and Lisandro Granville. As a direct comparison of security features, they enabled Secure HTTP for the security aspect. They also implement zlib compression before transmitting the data. The conclusion is divided into two parts, where SNMP gains edge in protocol level as it requires less header and bandwidth per message, however at object level, web services perform much better if larger amount of object are retrieved.

CHAPTER 3: METHODOLOGY

3.1 Introduction

In the previous section, there is sufficient research regarding both the SNMP and also the RESTful system. Despite the many challenges the SNMP protocol faces in this ever-changing industry, SNMP is still used extensively in networking devices especially older-generation devices. However, as the computational power follows the Moore law by increasing exponentially, more and more embedded devices are more than capable of delivering more functionality with the extra CPU cycles.

This shift and push in the industry to support more modern streaming telemetry sources with extra customizability has led to few alternatives to SNMP such as NETCONF, gRPC and the focus of this research, REST based systems. This shift arises few questions, particularly the performance difference between the two system when performing basic tasks in terms of network transport. Hence, this paper aims to answer the question of performance difference between SNMP and also REST in terms of the network layer communication.

3.2 Methodology and research approach

The approach of this research will be based on a modified version of the Spiral Model in the software development life cycle. There are four main phases in the Spiral Model, which are review, then determine the feasibility of idea, continued with implementation analysis and development, and then the data gathering and analysis. Finally, the cycle will complete and there will be the plan for next iteration. The focus of spiral model will be determining the risk of a project, this particular project instead modifies it to instead focus on the feasibility of idea.

3.3 Process of reviewing the project

This project starts with reviewing the idea of project itself. After obtaining an idea of the project to be done, the first thing to be determine is the problem, or in another word, what are the underlying problems that exists based on the idea itself. In the case of this project, the idea is about the differences between SNMP and HTTP Based REST API. The underlying problem is about the differences between the two protocols, how each protocols work, which one are more beneficial compared to others, what parameters or metrics will be changed based on which protocols chosen.

From the numerous underlying problems that is determined, there is a need for selecting few of the more specific problems to be assessed, as not every single problem is able to be discussed within a project. This project narrowed down the problem into three points, with the difference in the performance metric in terms of transmission of data and security concerns as the focal point.

Based on the narrowed down problem statements, there is a need to highlight the scope of the project. It helps in determining the specific goals, constraints, strategies, task and deliverables that should accomplish. The scope helps in preventing project being too broad or out of topic and potential delay or overwork may hinder the progress itself. The project will be focused on the transmission of the data. This project, despite the generation of data, and the format of data is also an integral part of the discussion from the problem statements, it is not included within the scope of the project.

3.4 Determining the feasibility of the idea

Before the actual process of gathering data, and going into the building of the system, there must be an in-depth justification of all of the potential knowledge required to actually analyze the project. The whole process of literature review is where a more comprehensive understanding of the subject matter is laid out, and all relevant information are being addressed before analysis could be understood.

For this particular project there are two major components, SNMP and the REST API process. The review starts with explaining the SNMP as a protocol, the mechanism of the protocol, what are the details in terms of transport based on the listed scope and the additional information that could help in making a more comprehensive overview.

The review continues with an idea floating around in the network management community where SNMP is inadequate despite it being a widely used protocol. There are quite some researches also done in regards to the weaknesses of that protocol despite there exists three iteration that tried to address the issues. The researchers also hinted on the possibility of some form of alternative protocol complement what SNMP has to offer.

The review then shifts to the alternative that was proposed, which is the transmission of monitoring data through RESTful API using the existing HTTP technology. The involved protocols, namely HTTP, the secure version of it, HTTPS, and also the concept of REST is also laid out to create a comprehensive image on how and what the alternate ideas consists, and how it could complement or in some sense even replace the SNMP itself.

3.5 Implementation analysis and development of project

After all of the theory have laid out and the requirements are highlighted, the preliminary research suggests that there is a demand for more research in this topic. There are already few researches done before, more specifically the research done by Choi M. and Hong J. about XML-SNMP gateway more than 20 years ago, which actually was one of the papers that inspired this project.

Based on the project scope, the approach will be purely assessing the network layer of the response between the server and the client. The test will involve two different operating system, Cisco iOS and also Linux system. The Windows-based system is not included in this testing because there will be concern of performance degradation due to too much external services being included in said system, making it hard to isolate the relevant traffic and potentially skewing the data. The test will be

conducted for the two major component, SNMP based system, and then the RESTful system.

The testing tools for SNMP is straightforward as there is already a standard implementation the SNMP system in both Linux system and also Cisco devices and also snmpwalk command to query all the SNMP system. The testing will involve two different protocol versions from SNMP, namely SNMPv2c and SNMPv3. The SNMPv3 will be tested with both the AuthPriv settings as well as the noAuthnoPriv settings.

For the RESTful HTTP API system, the testing of network routers will be utilizing the RESTCONF function that was supported in the Cisco operating system. The testing of the Debian-based Linux system does not have a standard for implementing RESTful system, hence the system will be built using a third party software called YumaWorks. It also involves two iterations, the plaintext HTTP and the secured HTTPS protocol.

3.6 Gathering and analysis of data.

The measurement of each of the testing will involve few key parameters. Firstly, there will be a measurement of the processing duration parameter. The time between the process of retrieving the data until the data is obtained is calculated.

Secondly will be the parameter of the size of packets. Each of the tasks will be given a requirement of set of data to be retrieved. The measurement of the packets will be based on all of the data sent between the network in one transaction including all of the overhead and related packets.

The third parameter will be measuring the security of the system. Basic analysis and security testing will be applied during the transport of messages between the devices in order to make an assessment of how the system performs in term of security.

The data parameters will be repeated multiple times to get an average data value, which in turn will be compared between all of the different iteration on both SNMP and HTTP API.

CHAPTER 4: DESIGN

4.1 Introduction

As the project will be an analysis of the performance of a protocol, a system should be designed and created in order for the protocol to function. In the previous section, it is presented that the systems used will be Cisco iOS and also Linux system.

This chapter will be the result of analysis of the preliminary design and also the tools at my disposal. It will also highlight the actual physical and logical implementation of this project, the software that was chosen to be operated and the environment that makes the project able to be implemented.

4.2 Network System Architecture

Based on the hardware and tools at hand, the architecture of this testing will be done in a virtualized environment. The architecture will be using a workstation, and the testing environment will be based on a virtualized system on top of a host system.

There will be argument about the performance degradation by conducting analysis in a virtualized environment, but as the trend of computing moving towards cloud computing and IaaS is becoming the first choice for companies, it is fair to argue this research also considers the performance parameter to be somewhat resemble the cloud environment where all of the machines deployed in the cloud are also actually virtualized.

As mentioned above, the Windows System will be absent in the research as the overhead of the whole system introduces too much noise, increasing the difficulty of isolating the intended parameters and skewing the data that could be used to compare and contrast the different protocols.

CPU	AMD Ryzen 7 5800H (3201 MHz), 8 Cores, 16 Logical Processors
Memory	16 GB X 2 (3200 MHz) SODIMM
OS	Windows 11 Pro 21H2 Build 22000.1936
Software	VMWare Workstation Pro Version: 17.0.0 build-20800274

Table 4.2.1 The specification of the WORKSTATION device

Table 4.2.1 below lists down the actual specification of the HOST PC that will be virtualizing the operating systems. The software environment chosen will be the latest VMWare Workstation and will be installed on Windows 11 operating system.

Device	Virtual Processors	Virtual Memory	Virtual HDD	Network Adapter	Operating System
CSR1000v (Cisco Router)	1 processor, 2 core per processor	8 GB	8 GB	VMnet vmxnet3	Cisco IOS XE Version 17.03.02
SRV-HOST (SNMP Agent)	2 processor, 2 core per processor	8 GB	15 GB	VMnet vmxnet3	Debian 11.6 Debian 5.10.158-2 (2022-12-13) x86_64
SRV-CLIENT (Managed device)	2 processor, 2 core per processor	8 GB	15 GB	VMnet vmxnet3	Debian 11.6 Debian 5.10.158-2 (2022-12-13) x86_64

Table 4.2.2 The specification of the virtualized devices for testing

Table 4.2.2 below summarizes all of the resources that are allocated for use in each of the virtualized operating system. The network will be using vmxnet instead of e1000 for better network performance as e1000 is an emulated interface while vmxnet is para-virtualized. The operating system will be based on IOS-XE version of Cisco routers that supported REST configuration, and also the Debian 11.6. As of this writing of this report, there yet to be a later version of the Debian software.

4.3 Logical and Physical Design

Referring to the Figure 4.3.1, the topology will consist of two Linux system and also one Cisco system. The network configuration is relatively simple with a virtual switch interface of VMnet1 connecting every single device and the host PC also able to access through the virtual switch interface. The hardware will just consist of a host PC, and all of the system will be virtualized in the host PC within VMWare Workstation software environment.

Figure 4.3.1 The physical layout of the topology

The IP addressing will be using the same subnet of class C (192.168.1.0/24) and the three system could communicate directly with each other using the assigned static IP addresses. The HOST PC is also assigned an IP for ease of management. The summary of the logical topology is shown on Figure 4.3.2

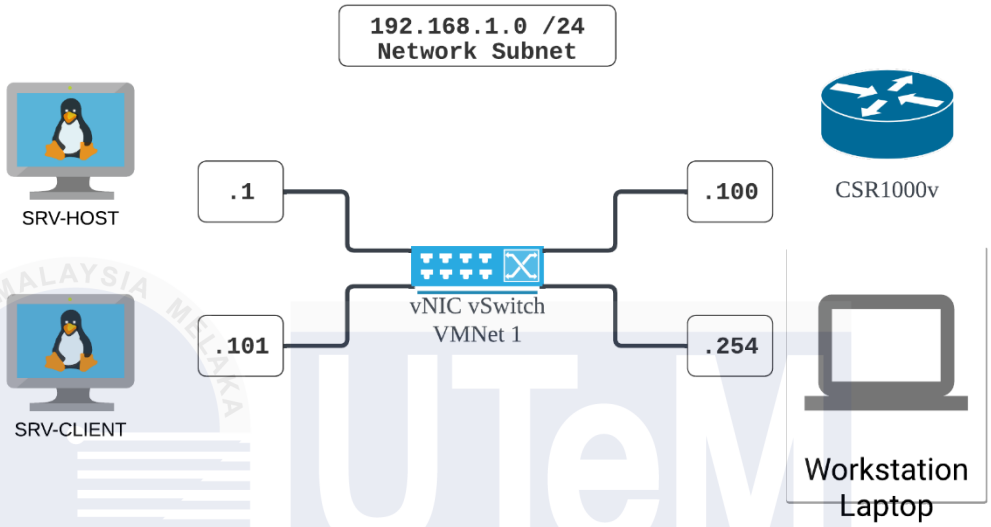

Figure 4.3.2 The logical layout of the topology with IP addressing

4.4 Testing parameters and justification of decision

There are three distinct data points that will be gathered for each process. The figure below shows the expected interaction between the server (SRV-HOST) and the clients (CSR100v and SRV-CLIENT).

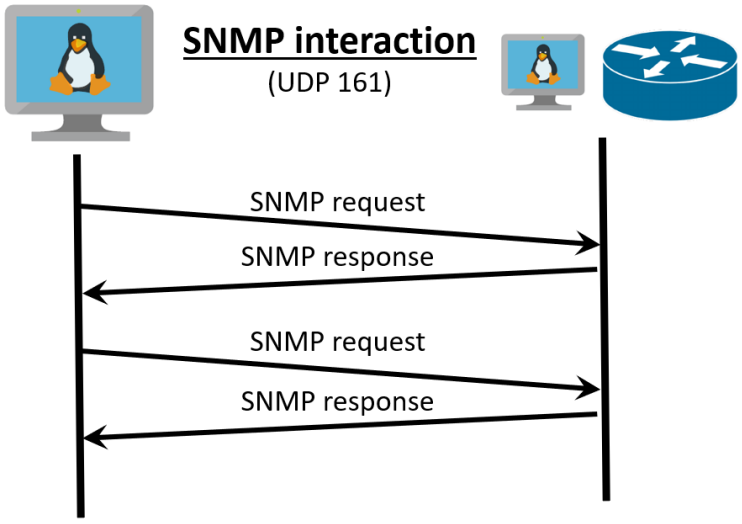

Figure 4.4.1 The interaction of SNMP between server and client

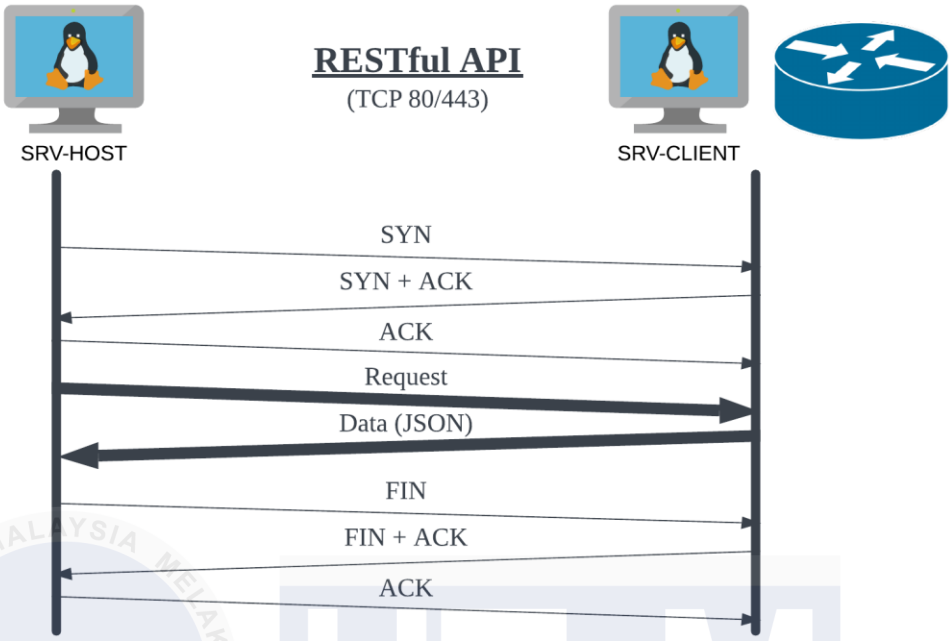

Figure 4.4.2 The interaction of RESTful API between server and client

The first data point that will be gathered is the size of all the transmission between two devices, including all of the traffic overhead. The protocol UDP is a connectionless communication, while TCP is a connection-oriented communication, and the data size calculation will include the headers and all of the security negotiation if applicable. Figure 4.4.1 and Figure 4.4.2 will be the expected interaction between system.

Based on the size of data, there will be multiple repetition of same iteration with increasing number of items requested between the two devices, with the total data transferred between the devices calculated. The data will be graphed and compared between all of the iterations.

The second data point will be the time between the execution of data and the displaying of the data on console. This will focus on the execution time and will be exclusively measured in SRV-HOST. The measurement of delay will be the round-trip instead of one-way communication.

The delay of communication will be done multiple time using the same dataset, and the average of the delay will be calculated to show the average time from the execution of command, the transfer delay, the processing of request and the

processing of response. The summary of the measurement is shown as the Figure 4.4.3 below.

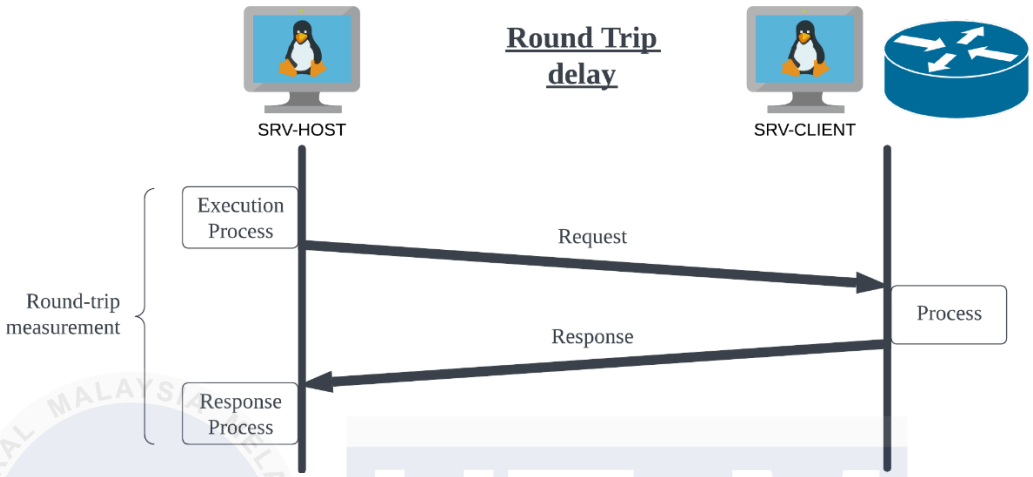

Figure 4.4.3 The summary of Round-Trip delay process

The third data point will be the security aspect, and the security part will be evaluated. There are few configuration parameters that will be changed, namely the SNMP version and the inclusion of TLS in REST. Few basic analyses on the security aspect during transfer will be done and a simple preliminary assessment will be done for each of the iteration regarding the security aspect, such as ease of deciphering data and the error handling of unsolicited messages

The parameters chosen on this project is based on previous scholar paper that investigates the approach for performance requirement verification by A. Waleed, X. Chen and Unterkalmsteiner, M. Based on their study from few primary studies chosen by the authors, they listed out 5 main performance aspect that affects a piece of software which are the following:

1. *Efficiency*
2. *Resource Utilization*
3. *Throughput/Speed*
4. *Capacity*
5. *Time Behavior*

. This paper expands on the idea and apply it on network services, because fundamentally every single network protocol used by network engineers in essence

comprises of multiple different software on top of networking hardware. The choices is further supported by the International Organization for Standardization, where they actually prepared a standard, ISO/IEC 25010:2011 that standardizes the Software Quality Requirements and Evaluation (SQuaRE). The performance efficiency is defined as the Time Behavior, Resource Utilization and also Capacity.

When mapping the performance aspects to suit this project, the throughput will be tested before the systems are actually implemented. The efficiency in software typically measures the output of the work compared to the expected outcome, hence the data size of transmission is the factor. The overhead and the supporting protocols transmitted, although is required for the protocol to function normally, still considered as unnecessary when talking about the actual data. This also relates to the capacity of the protocol, where the degree of limits of system parameter while still conforming to the network requirements.

The resource utilization refers to the performance and effort over an amount of time. CPU utilization are always referred when talking about the resources available, as these are the basic components where the software is executed. Time behavior on the other hand relates to the response of the software, and this could have different ways to measure the processing time of a certain software. This project chooses the Round Time Trip that includes the query and response of the software as the two exchanges is completed when both of the query and response are done and received by both ends.

4.5 Flow of testing process

The testing of the environment will be separated into two parts, where each of the configuration parameter are tested twice using two different tools, namely the software iPerf and also Wireshark.

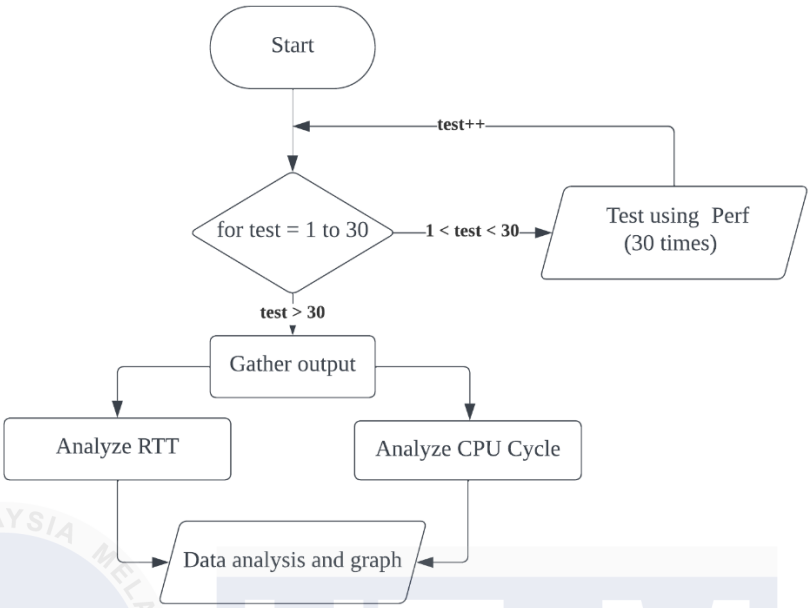

Figure 4.5.1 The flowchart of measuring the performance

The Perf is a powerful tool that instrument CPU performance counter on Linux systems. The tools provided are very detailed and technical, but the focus on this testing will be only be two main parameters under the "perf stat", which are the total cycle of CPU to complete the task, and also the time elapsed for the particular command. The command will be run for 30 times and the data will be averaged to decrease the random fluctuation of individual data points.

Testing for the SNMP will be conducted under 5 different scenarios for each of the SNMP version configurations. The testing will query different OID with different amount of data within the request. Table below shows the chosen OID that will be query and also the amount of data actually returned.

OID Value	Node name	Number of data
1.3.6.1.2.1.1.5	sysName	1
1.3.6.1.2.1.4.20	ipAddrTable	5
1.3.6.1.2.1.6	tcp	19
1.3.6.1.2.1.5	icmp	1058
1.3.6.1.2.1	mib	2124

Table 4.5.1 The OID used to test the SNMP values

The same thing will be used to test the REST API system, similar with SNMP system, but instead of using OID values to retrieve data, REST API actually uses URL with the HTTP GET method to obtain the data. The method requires authentication, hence there are also few parameters need to be added on the header of the HTTP Request. Table below shows the URL that will be queried and also the amount of data actually returned.

URL	**Number of data**
https://*{ip}*/restconf/data/Cisco-IOS-XE-native:native/hostname	1
https://*{ip}*/restconf/data/ietf-interfaces:interfaces	16
https://*{ip}*/restconf/data/ietf-routing:routing-state	94
https:// *{ip}*/restconf/data/Cisco-IOS-XE-native:native	184
https://*{ip}*/restconf/data/ietf-netconf-monitoring:netconf-state	2733

Table 4.5.2 The URL used to test the REST values

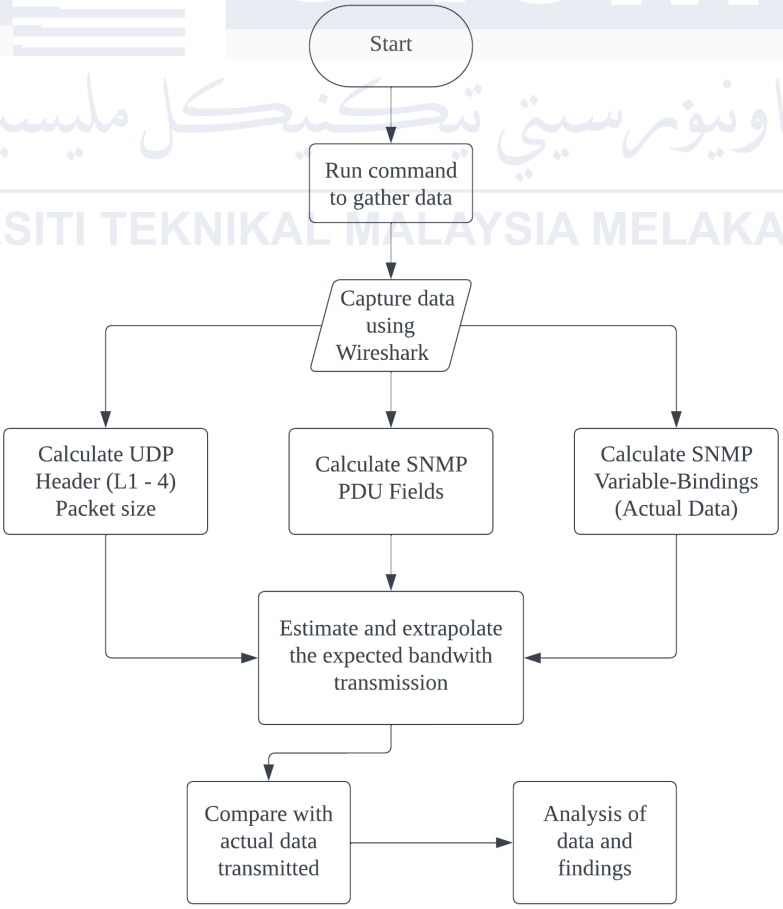

Figure 4.5.2 The flowchart of measuring the size of transmission for SNMP

The second part of the testing will be using the tool called Wireshark. It is an open-source packet analyzer that contains a suite of tools for analysis and troubleshooting of network environment. This project will only focus on one of the available functionalities, which is the capture of the packets, understanding the flow and also calculation of the byte size of each frame passing through the virtual switch.

Based on the data that was gathered, few assumptions will be made and a simple formula will be made, namely the expected average size of actual data in the PDU, as most of the fields should be fixed based on the configured environment. The data is then extrapolated, and is compared with the actual data for a rough trendline of how the data grows.

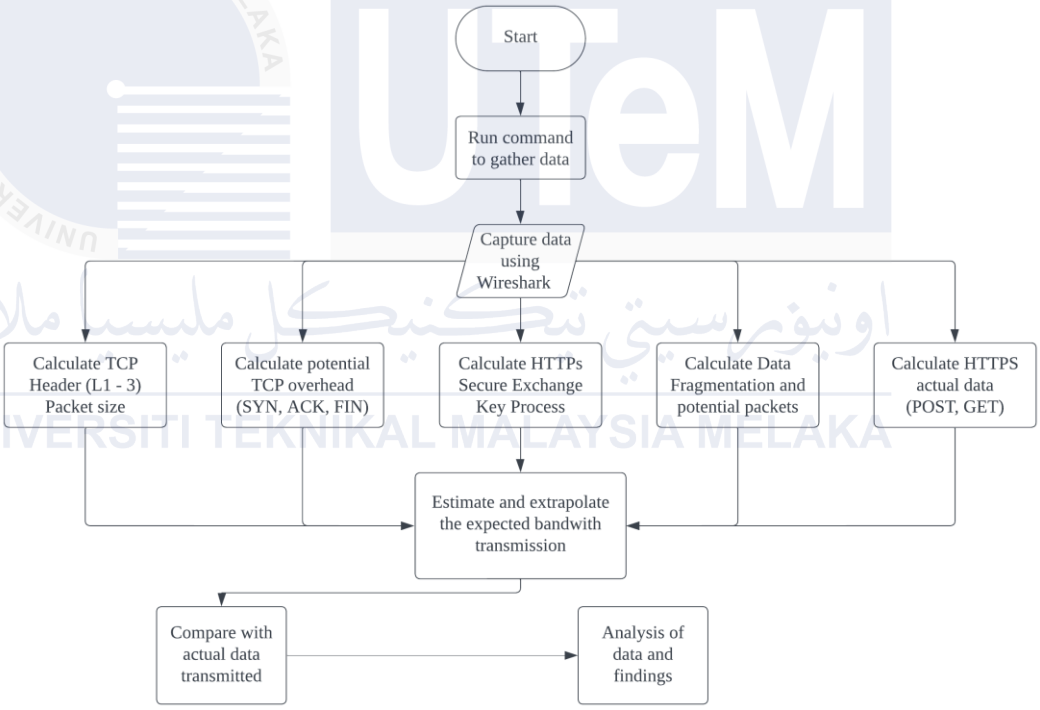

Figure 4.5.3 The flowchart of measuring the size of transmission for REST

The REST API will be following similar process as the process of the SNMP, which there is also calculation of the byte size of each frame passing through the virtual switch. However, there are some fundamental differences between the TCP protocol used by the REST API and the SNMP that uses UDP protocol.

TCP protocol is a type of connection-oriented protocol, means that the connection requires an established three-way handshake for communication between

devices before data can be exchanged. TCP will start the connection with SYN packets and ends with FIN packets, meanwhile the acknowledgement of packets received is represented by the ACK packets. These packets are also taken into account when calculating the total byte size.

CHAPTER 5: IMPLEMENTATION

5.1 Introduction

Based on the previous chapter, the implementation will be done in a virtualized environment. The implementation starts with the configuration of a clean copy of the required virtual machines, which are CSR1000v IOS-XE Cisco VM, and two Debian 11 bullseye CLI environment.

The next part is to configure a basic SNMP system on both of the testing VM, and also implement RESTful API at the same time. The different protocol is implemented simultaneously as there is no conflict for this particular configuration.

5.2 Basic Environment Setup

The setup starts with the installation of CSR1000v IOS-XE VM, that was provided from Cisco. The steps are very simple as the clean image is already pre-configured and just need to import the OVA into the VMWare.

The Debian 11.6 will be installed using the debian-11.6.0-amd64-DLBD-1.iso. The packages will only install the SSH and the essential tools without any GUI to reduce any variables and make it lightweight for the testing process.

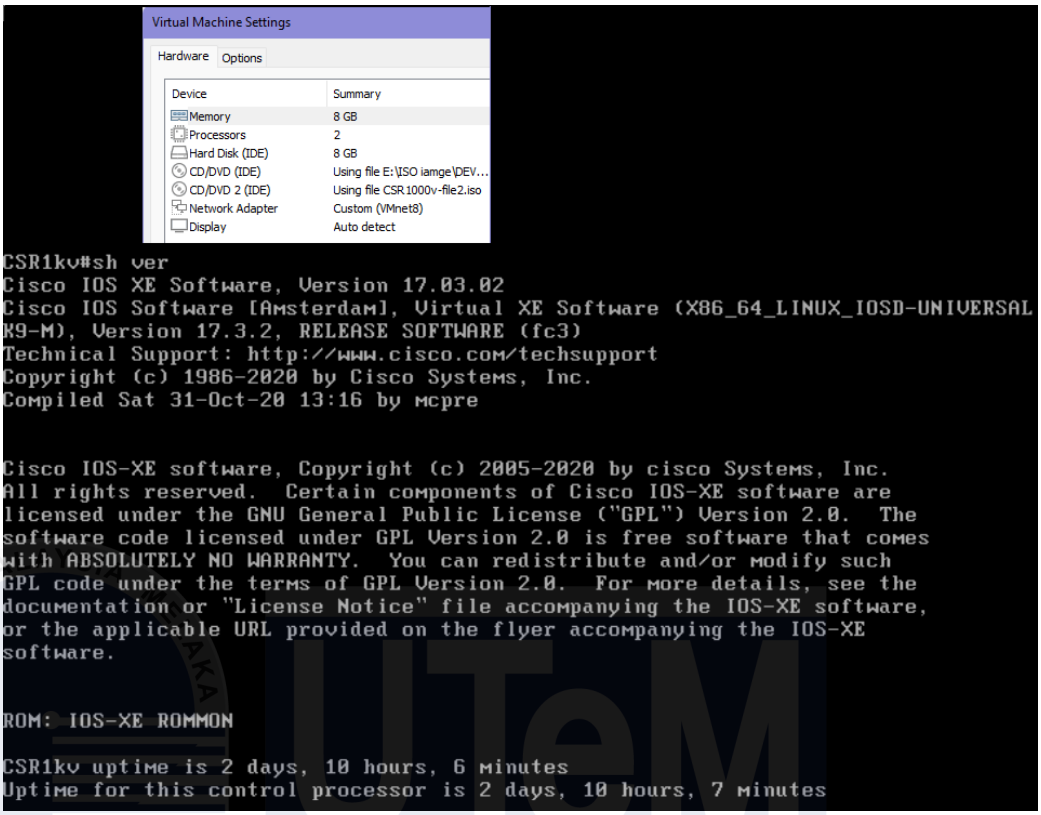

Figure 5.2.1 The setup of CSR1000v

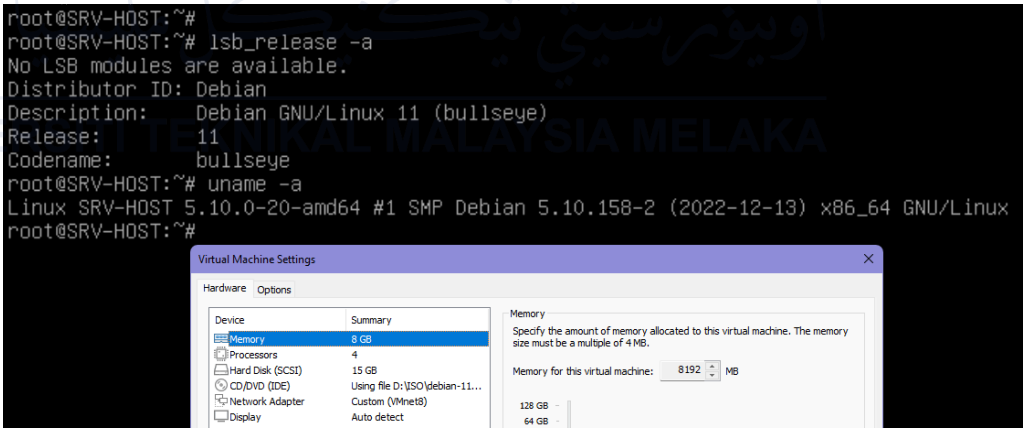

Figure 5.2.2 The setup of Linux Debian 11.6

After the installation is done and all VM are setup with the IP address assigned, a speed test is done between the devices. The tool that is used to test the bandwidth between the devices is called iPerf.

iPerf is a network performance measurement tool. that is used commonly to test the maximum achievable throughput between any two devices. A simple iPerf test normally requires one side of the device running iPerf in server mode, and another as client mode, connecting and testing the link with the server.

The implementation for iPerf in Linux is straightforward as the software is included in the official repository. The CSR is a bit unique as the iPerf is installed on top of the guestshell, sort of a virtualization layer running CentOS Linux on top of the Cisco IOS operating system. The iPerf is tested between the SRV-HOST VM with either CSR or SRV-CLIENT.

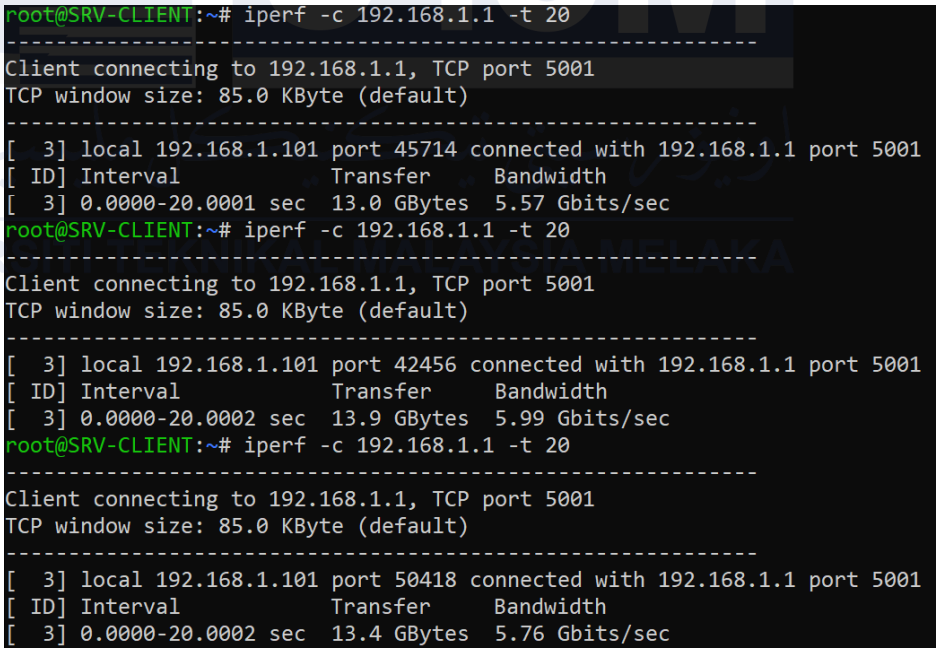

Figure 5.2.3 iPerf bandwidth between SRV-HOST and SRV-CLIENT

```
CSR1kv#guestshell
[guestshell@CSR1kv ~]$ iperf -c 192.168.1.1
------------------------------------------------------------
Client connecting to 192.168.1.1, TCP port 5001
TCP window size: 88.8 KByte (default)
------------------------------------------------------------
[  1] local 192.168.1.102 port 58614 connected with 192.168.1.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  1] 0.00-20.63 sec  2.63 MBytes  1.07 Mbits/sec
[guestshell@CSR1kv ~]$ iperf -c 192.168.1.1
------------------------------------------------------------
Client connecting to 192.168.1.1, TCP port 5001
TCP window size: 88.8 KByte (default)
------------------------------------------------------------
[  1] local 192.168.1.102 port 58616 connected with 192.168.1.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  1] 0.00-22.50 sec  3.38 MBytes  1.26 Mbits/sec
[guestshell@CSR1kv ~]$ iperf -c 192.168.1.1
------------------------------------------------------------
Client connecting to 192.168.1.1, TCP port 5001
TCP window size: 88.8 KByte (default)
------------------------------------------------------------
[  1] local 192.168.1.102 port 58630 connected with 192.168.1.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  1] 0.00-20.48 sec  2.50 MBytes  1.02 Mbits/sec
[guestshell@CSR1kv ~]$
```

Figure 5.2.4 iPerf bandwidth between SRV-HOST and CSR1000v

Due to the absence of license for the Cisco devices, the bandwidth for the CSR is capped at 1 Mbits/sec. This is a very different scenario as the bandwidth between Linux machines are 5.7 Gbits/sec, which is a huge difference with CSR.

```
^Croot@SRV-HOST:~# iperf -u -s
------------------------------------------------------------
Server listening on UDP port 5001
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 192.168.1.1 port 5001 connected with 192.168.1.101 port 50304
[ ID] Interval           Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3] 0.0000-10.0153 sec  1.25 MBytes  1.05 Mbits/sec   0.048 ms    0/  895 (0%)
[  4] local 192.168.1.1 port 5001 connected with 192.168.1.101 port 40765
[ ID] Interval           Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  4] 0.0000-10.0155 sec  1.25 MBytes  1.05 Mbits/sec   0.036 ms    0/  895 (0%)
[  3] local 192.168.1.1 port 5001 connected with 192.168.1.101 port 33810
[ ID] Interval           Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3] 0.0000-10.0152 sec  1.25 MBytes  1.05 Mbits/sec   0.037 ms    0/  895 (0%)
```

Figure 5.2.5 iPerf jitter between SRV-HOST and SRV-CLIENT

```
root@SRV-HOST:~# iperf -u -s
------------------------------------------------------------
Server listening on UDP port 5001
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 192.168.1.1 port 5001 connected with 192.168.1.102 port 46632
[ ID] Interval           Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3] 0.0000-10.0154 sec  1.25 MBytes  1.05 Mbits/sec   0.043 ms    0/  895 (0%)
[  4] local 192.168.1.1 port 5001 connected with 192.168.1.102 port 44290
[ ID] Interval           Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  4] 0.0000-10.0152 sec  1.25 MBytes  1.05 Mbits/sec   0.033 ms    0/  895 (0%)
[  3] local 192.168.1.1 port 5001 connected with 192.168.1.102 port 44771
[ ID] Interval           Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3] 0.0000-10.0152 sec  1.25 MBytes  1.05 Mbits/sec   0.053 ms    0/  895 (0%)
```

Figure 5.2.6 iPerf jitter between SRV-HOST and CSR1000v

When the bandwidth is being set to the same rate, which are 1.05 Mbits/sec, there is no significant difference between the two devices, with no packet loss for UDP connection, and a jitter less than 0.05 ms. The link between the two devices could be considered stable, and it should be considering the VM are directly connected to the same virtual switch within the same subnet.

5.3 SNMP Environment setup

After all of the system are installed and running, the implementation starts with the configuration of the SNMP system. The setup of the SNMP server in the CSR and Linux will involve the community psm2 for the SNMPv2, and also 3 users for SNMPv3, each with different security levels based on the table below

Username	**Version**	**Authentication**	**Encryption**
psm2-user1	v3 noAuthnoPriv	-	-
psm2-user2	v3 authNoPriv	sha: Skills39	-
psm2-user3	v3 authPriv	sha: Skills39	aes: Skills39

Table 5.3.1 List of user and parameters for SNMP

SRV-HOST will be tested using the snmpwalk command to verify that the SNMP server is functioning, with the command is listed below. The output should be printed on the console.

Version	**Commands**
v2c	snmpwalk -v2c -c psm2 *{ip address} {OID}*
v3 noAuthnoPriv	snmpwalk -v3 -l noAuthNoPriv -u psm2-user1 *{ip address} {OID}*
v3 authNoPriv	snmpwalk -v3 -l AuthNoPriv -u psm2-user2 -a sha -A Skills39 *{ip address} {OID}*
v3 authPriv	snmpwalk -v3 -l authPriv -u psm2-user3 -a SHA -A Skills39 -x AES -X Skills39 *{ip address} {OID}*

Table 5.3.2 Commands to verify the functionality of SNMP server

```
root@SRV-HOST:~# snmpwalk -v3 -l noAuthNoPriv -u psm2-user1 192.168.1.100 1.3.6.1.2.1.4.20
iso.3.6.1.2.1.4.20.1.1.192.168.1.100 = IpAddress: 192.168.1.100
iso.3.6.1.2.1.4.20.1.2.192.168.1.100 = INTEGER: 1
iso.3.6.1.2.1.4.20.1.3.192.168.1.100 = IpAddress: 255.255.255.0
iso.3.6.1.2.1.4.20.1.4.192.168.1.100 = INTEGER: 1
iso.3.6.1.2.1.4.20.1.5.192.168.1.100 = INTEGER: 18024
root@SRV-HOST:~# snmpwalk -v3 -l noAuthNoPriv -u psm2-user1 192.168.1.100 1.3.6.1.2.1.1.5
iso.3.6.1.2.1.1.5.0 = STRING: "CSR1kv"
```

Figure 5.3.1 SNMP commands example output

Based on the figure above, if configured correctly the terminal should output the corresponding OID value with the value of the data. This shows that the commands runs successfully and the SNMP system is running as expected.

5.4 RESTful Environment setup

The IOS-XE version of Cisco operating system by default supports the RESTful API functionality. Before the REST is actually enabled, there are few pre-requisites configuration need to be done. It starts with creating a user, in this environment, a user with the credential of psm2 and password Skills39 is created.

Next, the RESTful API system only runs on HTTPS protocol, hence a keypair is required to be generated as the self-signed certificate to secure the traffic. Generate a keypair that is exportable for testing as shown in the figure below. Then generate a certificate, and apply it in the ip http secure-server on the CSR.

```
CSR1kv#sh crypto key mypubkey rsa CSR-KEY
% Key pair was generated at: 15:59:26 UTC Sep 8 2023
Key name: CSR-KEY
Key type: RSA KEYS
 Storage Device: private-config
 Usage: General Purpose Key
 Key is exportable. Redundancy enabled.
 Key Data:
  30820122 300D0609 2A864886 F70D0101 01050003 82010F00 3082010A 02820101
  00A5E0E8 CB1EC08A D4F1093E 2121347E D2D36F48 21EFDC11 819EEAB7 8069CC2C
  3C25C92F 85BAB835 5ADD725E 17955F69 475BD093 4389E3CB 7689DBB6 E5315B27
  46BB10FE 472A5D26 5E849B93 729742EA 0A0DEFBF 5BCFB9CB 8D81F89A 679F1D77
  A9B4E839 029C1869 58F84991 41FCDFEB 4BE3C636 A94EAD82 36A49312 F6BC3930
  79CEFC4E 77623DD2 E53E41B3 164E7AD4 7F5D0306 3111E9B8 B038AE8A 18754E1E
  C5104731 5D074CD2 5B67D3B5 959547E8 B5B4EEDF AFE5C42F 6FCC334E 2C5F08B9
  DCE719BB 0B88A00D CF158111 91E974AA E3544055 D75FCC53 26D5C23B 2FE2EE99
  3FD2B821 53F9540A 604C77AD D2DFB7D3 ED0CE513 E57EC584 B17AB7F5 00DCC327
  C5020301 0001
```

Figure 5.4.1 RSA Key on CSR1000v

Linux does not contain any standardized RESTCONF system, hence for this testing purpose, a third-party software, YumaWorks will be implemented on the Linux system. If the system is configured correctly, the HTTP request for the RESTCONF should return code 200 OK.

```
root@SRV-HOST:~# curl -IL http://192.168.1.101/restconf/data
HTTP/1.1 200 OK
Date: Sun, 10 Sep 2023 15:59:21 GMT
Server: Apache/2.4.54 (Debian)
Cache-Control: no-cache
Pragma: no-cache
Last-Modified: Sat, 12 Aug 2023 17:07:07 GMT
ETag: 8
Content-Type: application/json

root@SRV-HOST:~# curl -IL -k --location 'https://192.168.1.100/restconf/data/Cisco-IOS-XE-native:native'
 --header 'Accept: application/yang-data+json' --header 'Authorization: Basic cHNtMjpTa2lsbHMzOQ=='
HTTP/1.1 200 OK
Server: openresty
Date: Sun, 10 Sep 2023 15:59:59 GMT
Content-Type: application/yang-data+json
Connection: keep-alive
Cache-Control: private, no-cache, must-revalidate, proxy-revalidate
Pragma: no-cache
```

Figure 5.4.2 Verification of RESTful API based on header and return code

```
root@SRV-HOST:~# curl -k --location 'https://192.168.1.100/restconf/
data/Cisco-IOS-XE-native:native/hostname' --header 'Accept: applicat
ion/yang-data+json' --header 'Authorization: Basic cHNtMjpTa2lsbHMzO
Q=='
{
  "Cisco-IOS-XE-native:hostname": "CSR1kv"
}
root@SRV-HOST:~# curl -k --location 'https://192.168.1.100/restconf/
data/Cisco-IOS-XE-native:native/version' --header 'Accept: applicati
on/yang-data+json' --header 'Authorization: Basic cHNtMjpTa2lsbHMzOQ
=='
{
  "Cisco-IOS-XE-native:version": "17.3"
}
```


Figure 5.4.3 Verification of RESTful API based on output

The return code is normally reflected on the header of the HTTP response, the actual response with the desired value will be actually returned within the data field of HTTP, in the form of JSON as shown in the figure above.

5.5 Capturing packets through Wireshark

The implementation is relatively simple, and the environment is designed to focus on two systems, namely the SNMP and the RESTful API. In this testing case, Wireshark is also used to discern the interaction and packets sent between the different VM. The HOST PC is also connected to the virtual switch of the VM, hence the Wireshark will be listening to every packet running through the virtual switch, similar to the port mirroring of a physical switch

	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.1.253	192.168.1.255	UDP	86	57621 → 57621 Len=44
2	0.913857	192.168.1.101	192.168.1.1	ICMP	98	Echo (ping) request id=0xe375, seq=1/256, ttl=64 (reply in 3)
3	0.913969	192.168.1.1	192.168.1.101	ICMP	98	Echo (ping) reply id=0xe375, seq=1/256, ttl=64 (request in 2)
4	4.801206	VMware_75:32:ae	Broadcast	ARP	42	Who has 192.168.1.100? Tell 192.168.1.101
5	4.801983	VMware_d3:1d:d3	VMware_75:32:ae	ARP	60	192.168.1.100 is at 00:0c:29:d3:1d:d3
6	4.802049	192.168.1.101	192.168.1.100	ICMP	98	Echo (ping) request id=0x8b74, seq=1/256, ttl=64 (no response found!)
7	5.812756	192.168.1.101	192.168.1.100	ICMP	98	Echo (ping) request id=0x8b74, seq=2/512, ttl=64 (reply in 8)
8	5.813261	192.168.1.100	192.168.1.101	ICMP	98	Echo (ping) reply id=0x8b74, seq=2/512, ttl=255 (request in 7)
9	5.946220	VMware_09:40:7c	VMware_75:32:ae	ARP	42	Who has 192.168.1.101? Tell 192.168.1.1
10	5.946362	VMware_75:32:ae	VMware_09:40:7c	ARP	42	192.168.1.101 is at 00:0c:29:75:32:ae
11	6.100750	VMware_75:32:ae	VMware_09:40:7c	ARP	42	Who has 192.168.1.1? Tell 192.168.1.101
12	6.100874	VMware_09:40:7c	VMware_75:32:ae	ARP	42	192.168.1.1 is at 00:0c:29:09:40:7c
13	6.814387	192.168.1.101	192.168.1.100	ICMP	98	Echo (ping) request id=0x8b74, seq=3/768, ttl=64 (reply in 14)
14	6.814899	192.168.1.100	192.168.1.101	ICMP	98	Echo (ping) reply id=0x8b74, seq=3/768, ttl=255 (request in 13)
15	15.508164	192.168.1.253	192.168.1.255	UDP	86	57621 → 57621 Len=44
16	18.538310	192.168.1.253	192.168.1.255	UDP	86	57621 → 57621 Len=44
17	33.077985	fe80::135f:e756:449…	ff02::1:ffb7:9656	ICMPv6	86	Neighbor Solicitation for fe80::a62a:95ff:feb7:9656 from 00:50:56:c0:00:08
18	35.622029	192.168.1.253	224.0.0.251	MDNS	87	Standard query 0x0000 PTR _spotify-connect._tcp.local, "QM" question
19	35.622281	fe80::135f:e756:449…	ff02::fb	MDNS	107	Standard query 0x0000 PTR _spotify-connect._tcp.local, "QM" question
20	37.730108	192.168.1.253	239.255.255.250	SSDP	167	M-SEARCH * HTTP/1.1
21	48.544201	192.168.1.253	192.168.1.255	UDP	86	57621 → 57621 Len=44
22	57.088929	192.168.1.1	192.168.1.100	SNMP	106	get-request
23	57.089711	192.168.1.100	192.168.1.1	SNMP	149	report 1.3.6.1.6.3.15.1.1.4.0
24	57.089868	192.168.1.1	192.168.1.100	SNMP	155	get-next-request 1.3.6.1.2.1.1.5
25	57.090251	192.168.1.100	192.168.1.1	SNMP	161	get-response 1.3.6.1.2.1.1.5.0
26	57.090685	192.168.1.1	192.168.1.100	SNMP	156	get-next-request 1.3.6.1.2.1.1.5.0
27	57.091238	192.168.1.100	192.168.1.1	SNMP	163	get-response 1.3.6.1.2.1.1.6.0

Figure 5.5.1 Example of packet capturing in Wireshark

The figure shows an example of the packet capture of the virtual switch, with many different types of traffic, such as ARP, ICMP and few more. The packets in question for this project will be the SNMP UDP packets, and also the REST TCP packets that comprises on TCP headers packets alongside the TLS encrypted packets for the HTTPs protocol. There are a lot of details contained within the program, from the individual fields exist on every frame to the overall byte size of every single frame, which will be the main focus and also one of the key values that will be extracted and further analyzed. Below shows one example from a frame from SNMP.


```
> Frame 23: 149 bytes on wire (1192 bits), 149 bytes captured (1192 bits)
> Ethernet II, Src: VMware_d3:1d:d3 (00:0c:29:d3:1d:d3), Dst: VMware_09:4
> Internet Protocol Version 4, Src: 192.168.1.100, Dst: 192.168.1.1
> User Datagram Protocol, Src Port: 161, Dst Port: 35491
v Simple Network Management Protocol
    msgVersion: snmpv3 (3)
  > msgGlobalData
  > msgAuthoritativeEngineID: 800000090300000c29d31dd3
    msgAuthoritativeEngineBoots: 2
    msgAuthoritativeEngineTime: 3023380
    msgUserName:
    msgAuthenticationParameters: <MISSING>
    msgPrivacyParameters: <MISSING>
  v msgData: plaintext (0)
    v plaintext
      > contextEngineID: 800000090300000c29d31dd3
        contextName:
      v data: report (8)
        v report
            request-id: 1681914416
            error-status: noError (0)
            error-index: 0
          > variable-bindings: 1 item
```

Figure 5.5.2 Example of detailed information of a frame in Wireshark

CHAPTER 6: TESTING AND ANALYSIS

6.1 Introduction

The testing will be done once the environment is done, and the testing will be mainly utilizing the SRV-HOST with mostly open-source tools provided on the official Linux repository.

The testing comprises of three items, which are the byte size of data transmitted, and also the performance of the protocols, namely the Round Time Trip and also the CPU cycle. The transmission of data is also analyzed in terms of security, such as the encryption method and also the available different options for the security mechanism.

6.2 Interaction between devices

The measurement of data size is gathered using Wireshark that is hooked to the Virtual Switch connecting the devices. The Wireshark is able to capture the whole packets, and in turn display the total byte size of the frame itself.

The structure of the packets should be consistent with the standards that was highlighted on RFC 9293 for TCP and also RFC 768 for UDP. Below summarizes the packet header structure from Layer 1 to Layer 4 for the OSI model, with the byte size of the testing environment. The byte size of the packets is consistent throughout the testing, with the assumption the environment is in IPv4 with no additional header information embedded on the frames.

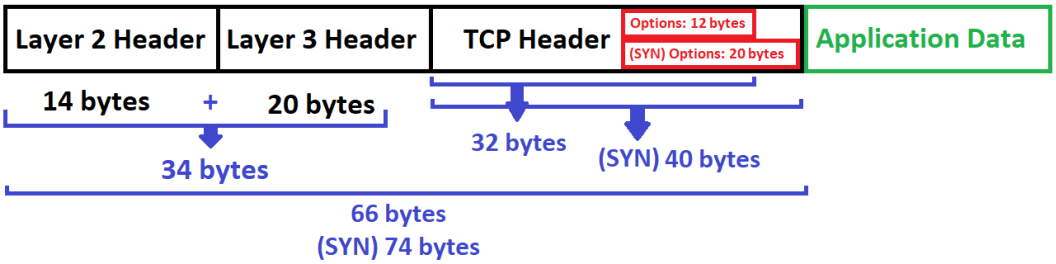

Figure 6.2.1 Header structure of TCP packets

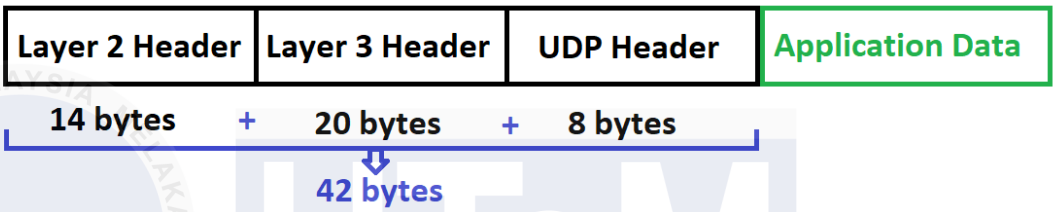

Figure 6.2.2 Header structure of UDP packets

The size of header is different between the two protocols, however when comparing the total size of transmission, the application data makes a big difference. Below is the transmission SNMP packets, with the structure of the SNMP data fields.

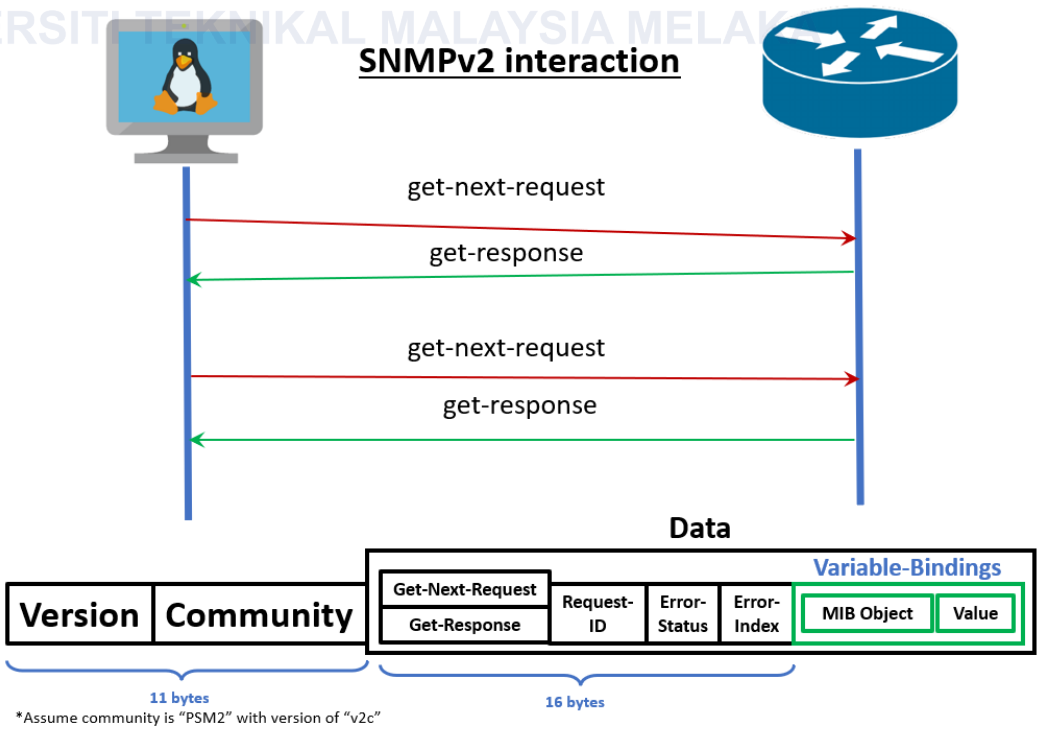

Figure 6.2.3 Data Flow and Structure of SNMPv2

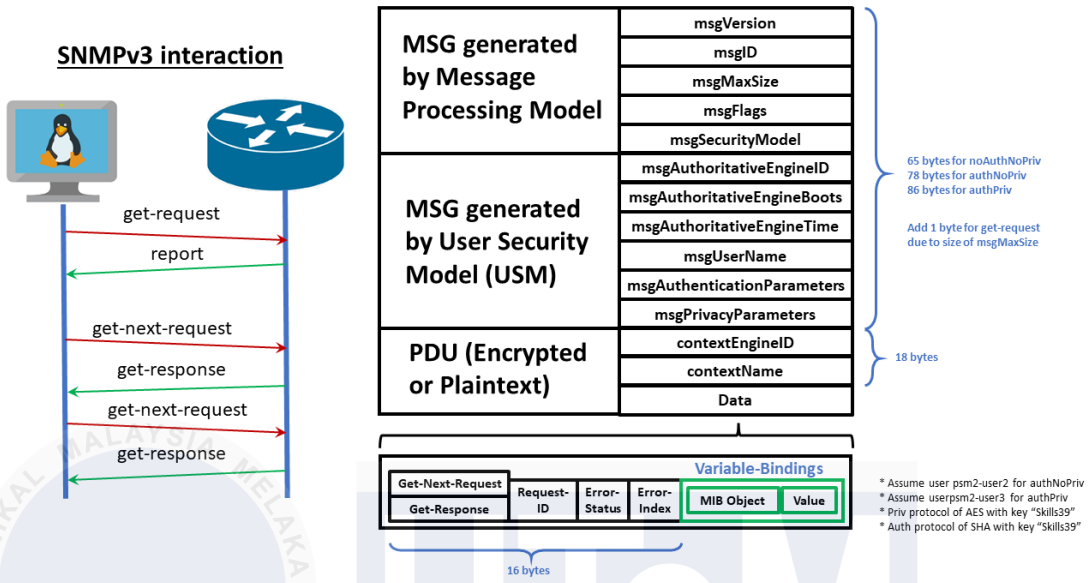

Figure 6.2.4 Data Flow and Structure of SNMPv2

As shown on the diagram, for the SNMPv2, the request of data directly starts with the client sends a GetNextRequest-PDU, and the server will response with GetResponse-PDU, where the MIB and the actual value for the corresponding MIB object is returned. The client then continues to query the next value by sending GetNextRequest-PDU with the current value of MIB object, and asking the server to return the next object. The process repeats until the server return the MIB object that conceptually follows the table in the implementation of the MIB, signaling that the table have been fully traversed.

When talking about SNMPv3, the biggest difference between the previous version is the inclusion of the User-based Security Model (USM). The USM specifies the authentication and encryption of the SNMPv3 packet. It also provides few additional information such as the boot times, up time, the user associated, and more importantly the engine ID for the key localization mechanism.

SNMPv3 starts with the Get-Request-PDU, where the client discover or query the SNMP server for the contextEngineID that is required to generate the SNMPv3 packets. The server replies with Report-PDU, where the PDU will reply

with contextEngineID, the authentication and privacy if the parameters are configured.

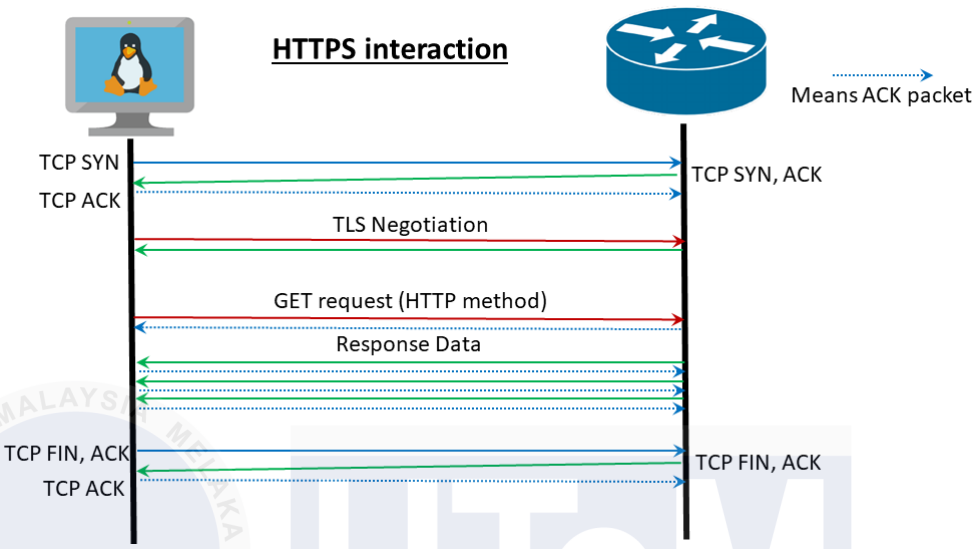

Figure 6.2.5 Data Flow of TCP HTTPS

HTTPS protocol operates very differently than the SNMP protocol, and it has much more component as it was using TCP, compared to the SNMP protocol that uses UDP. The connection starts with the TCP three-way handshake to establish the connection. The next process is the negotiation of the TLS secure channel using the cryptographic key to secure the traffic. The negotiated encryption method is then used to secure and protect the data of the actual traffic, encrypting the traffic and makes it almost impossible to intercept the data. The actual data is also accompanied by the ACK packets, where the receiving side of the communication actually send an acknowledgement packet (ACK) to notify the sender that the packets arrived successfully.

The data is also structured in a way where the packets are split when the size of the data exceeds the MTU of the device. As we are using the default configuration, the MTU should be 1500, means each of the protocol data unit (PDU) should not be more than 1500 byte. The overhead of the traffic will vary between transmission based on a multitude of factors but one of the key indicators is the total size of actual data communicated.

6.3 Size of data transmission

Based on the interaction that was highlighted above, few educated guesses could be made for each of the interactions above. The formula below summarizes how the SNMPv2 interaction translates to the estimations of the byte size of each data.

Packet size SNMPv2c =
Layer 2 data (14 bytes) +
Layer 3 data (20 bytes) +
UDP data (8 bytes) +
SNMP Header Data (11 bytes) +
PDU Header Data (16 bytes) +
Data (x) (OID value + Actual value)

Data sent for each SNMPv2c transaction =
2(69 + x) bytes of data

The value is based on the values that Wireshark provides when analyzing the actual byte size for each segment of the frames. The values are consistent throughout the transmission, hence for this part the values hold true for this particular environment between Linux and CSR1000v queried through snmpwalk command. The figure below shows the growth rate of data for the SNMP packets, with assumption of average actual data size between 15 and 30 bytes per packet, and also the actual data sent by the MIB.

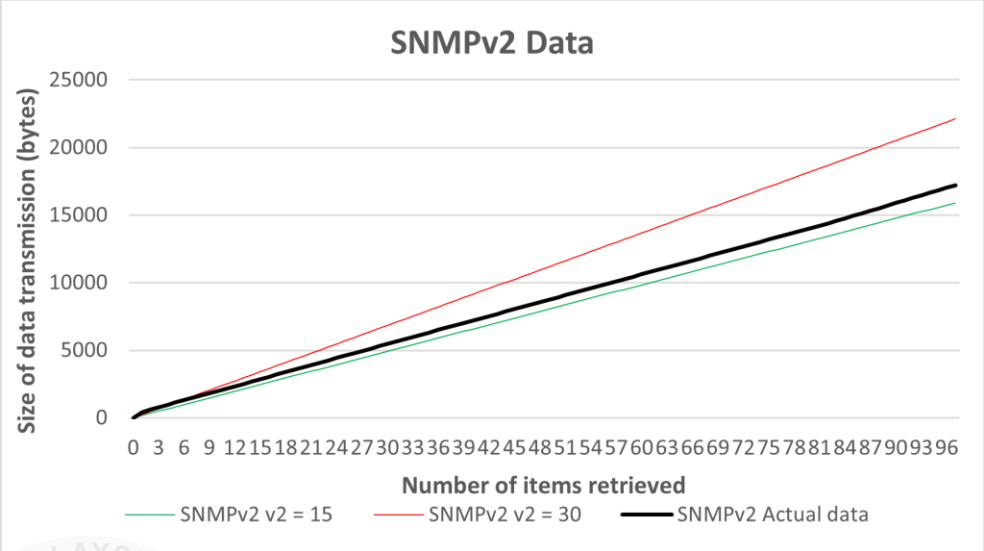

Figure 6.3.1 SNMPv2 Data Transmission

As shown on the graph, the growth of the data is based on a linear fashion, where the data is growing at a constant rate with the extrapolation of the number of data if the data of PDU is 15 bytes and also 30 bytes. The estimated bytes are 16400 bytes and 22800 bytes for the 15 bytes and 30 bytes PDU respectively.

The actual data based on the snmpwalk command also shows a consistent growth in line with the expectation as the average of actual data on the PDU is calculated to be around 18 bytes. The actual size of transmission based on frame forwarded gathered by Wireshark shows to be 17713 bytes, within the expected range.

The same can be applied to SNMPv3 of all configurations, where the data is growing linearly

Packet size SNMPv3 noAuthNoPriv =
Layer 2 data (14 bytes) +
Layer 3 data (20 bytes) +
UDP data (8 bytes) +
SNMPv3 Header Data (65 bytes) +
PDU Header Data (34 bytes) +
Data (x) (OID value + Actual value)

Data sent for each SNMPv3 noAuthNoPriv transaction = (2(141 + x) + 1) bytes of data

Include the get-request (106 bytes) and report (148 bytes)

Packet size SNMPv3 AuthNoPriv =
Layer 2 data (14 bytes) +
Layer 3 data (20 bytes) +
UDP data (8 bytes) +
SNMPv3 Header Data (78 bytes) +
PDU Header Data (34 bytes) +
Data (x) (OID value + Actual value)

Data sent for each SNMPv3 AuthNoPriv transaction = (2(154 + x) + 1) bytes of data

Include the get-request (106 bytes) and report (148 bytes)

Packet size SNMPv3 authPriv =
Layer 2 data (14 bytes) +
Layer 3 data (20 bytes) +
UDP data (8 bytes) +
SNMPv3 Header Data (86 bytes) +
PDU Header Data (34 bytes) +
Data (x) (OID value + Actual value)

Data sent for each SNMPv3 authPriv transaction = (2(162 + x) + 1) bytes of data

Include the get-request (106 bytes) and report (148 bytes)

Based on the formula above, there is an overhead of 106 bytes and 148 bytes of get-request and report packets. This is to request values of Context EngineID, Context Name and also security parameters. Similar to the SNMPv2, the data grows in a linear fashion, with the additional of the SNMPv3 header data that increases quite a bit of overhead compared to SNMPv2.

The size of header is also determined by the type of security parameter configured, where the noAuthNoPriv only uses 65 bytes, the authNoPriv uses 78 bytes and authPriv uses 86 bytes. All others being equal, it should see a same trendline of data with different growth rate.

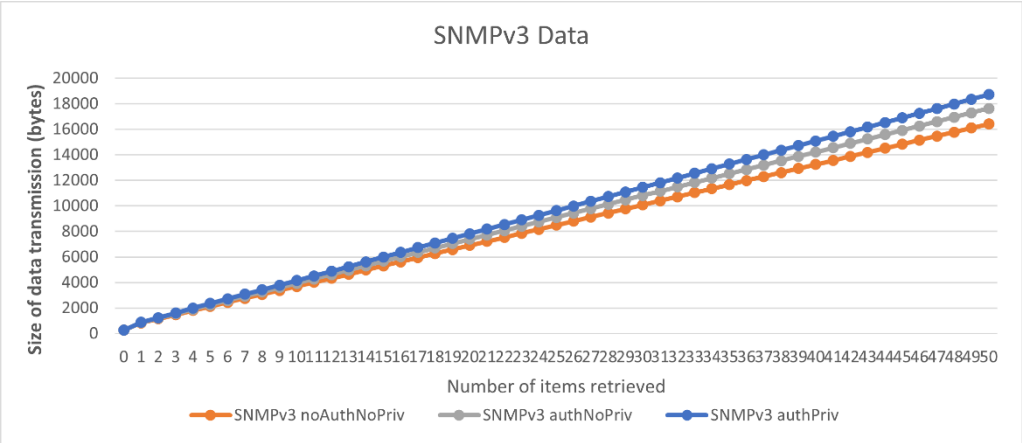

Figure 6.3.2 SNMPv3 Data Transmission

The same thing could not be said with the RESTful API as the TCP interaction is more complex than the connectionless UDP. The calculation will be done in parts, where few assumptions will be made in to predict the growth of the data. TCP is a connection-oriented protocol, hence there is a transmission of SYNACK packets to establish connection and FINACK packets to terminate connection. TLS also requires a close_notify message before ending the connection. The packets are consistent throughout testing and is calculated as shown below.

SYN ACK data -> 74 + 74 +66 = 214 bytes
FIN ACK data -> 66 + 66 + 66 = 198 bytes
Close notify -> 66 + 57 + 66 = 189 bytes

The next part is the TLS negotiation, where the secure channel is created. This only applies to the HTTPS protocol and this does not apply to the HTTP only protocol. There are few steps involve, as shown in the figure below.

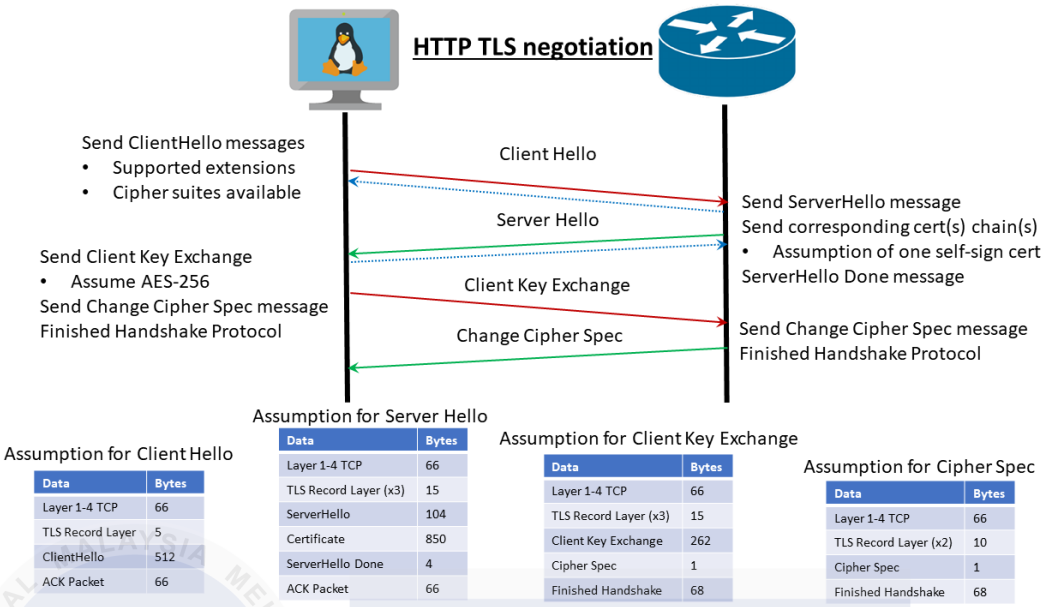

Figure 6.3.3 RESTful HTTPS TLS Interaction

Few assumptions are made during this testing, as different environment yields massive discrepancy, but the numbers will be assumed based on the current testing environment. The ClientHello in this case a 512-byte data is sent with the information of the client such as the supported TLS version and cipher suites.

The certificate is also vary greatly based on the certificate infrastructure, as a typical environment will have a multi-tiered certificate signed by a trusted Certificate Authority provider. The certificate typically requires around an average of 1500 bytes with 2 more intermediate issuer certificates also included. However, as this is an isolated lab environment, the certificate is self-signed and only uses around 850 bytes.

The next big assumption is the cipher suite used, that will affect the Client Key Exchange. In this environment, the cipher used is AES-256, which means it uses a symmetric encryption algorithm consists of a 256-bit key, and which translates to the size of the exchange itself.

The estimated transmission size of the data will be around 2311 bytes, based on the assumption and the calculation below.

HTTPs Negotiation:
Client Hello = 66 + 5 + 512 + 66 = 649
Server Hello = 66 + 15 + 104 + 850 + 4 + 66 = 1105
Client Key Exchange = 66 + 15 + 262 + 1 + 68 = 412
Change Cipher Spec = 66 + 10 + 1 + 68 = 145

Data sent for HTTPs negotiation =
(649 + 1105 + 412 + 145) = 2311 bytes of data

The fragmentation of packets makes it too complicated to make an educated guess of the data transmitted, as it requires to take account of many things such as the MTU and TCP MSS value, the PSH ACK configuration, the optimization of TCP traffic within network, that influences how many packets are generated for each transaction, and in turn the data transmitted, including the overhead of the ACK packets.

Hence, the actual data are taken directly from Wireshark for analysis purposes. Below is the graph of the data transmitted for both HTTP and HTTPS.

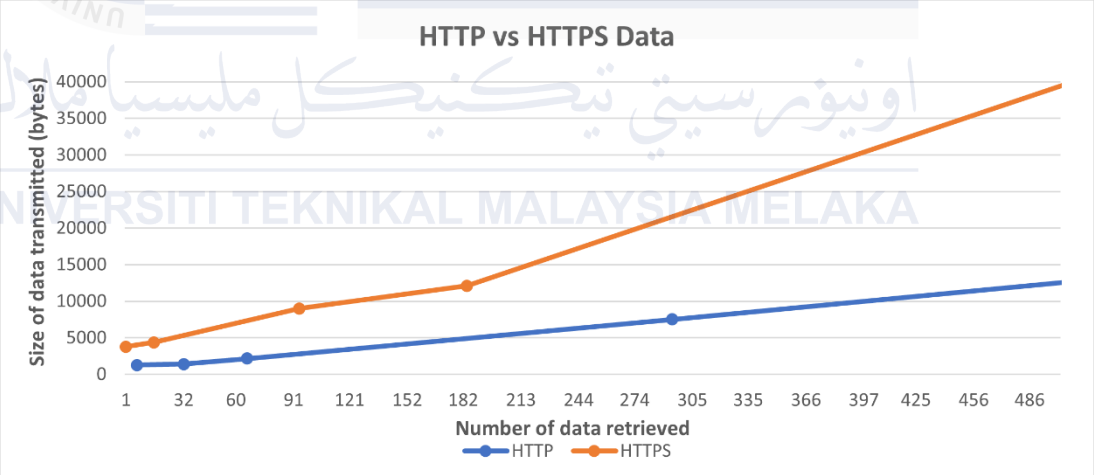

Figure 6.3.4 RESTful Data Transmission

The HTTP will have a much lower bytes required per data, and subsequent data as the traffic is transmitted in plaintext, which eliminates the many inefficiencies of encryption such as padding and chunking the data into blocks. The graph clearly shows that the data of the HTTP grows in a much slower pace than the HTTPS. There are however few limitations as the data points are small and there may be some discrepancy between data points, however it should generally be true that the HTTPS just require much less bandwidth to send the same amount of data.

Based on the data that was gathered above, another graph is plotted for all of the data points, based on the items retrieved and the actual transmitted data size. There are few observations that could be made from this graph itself.

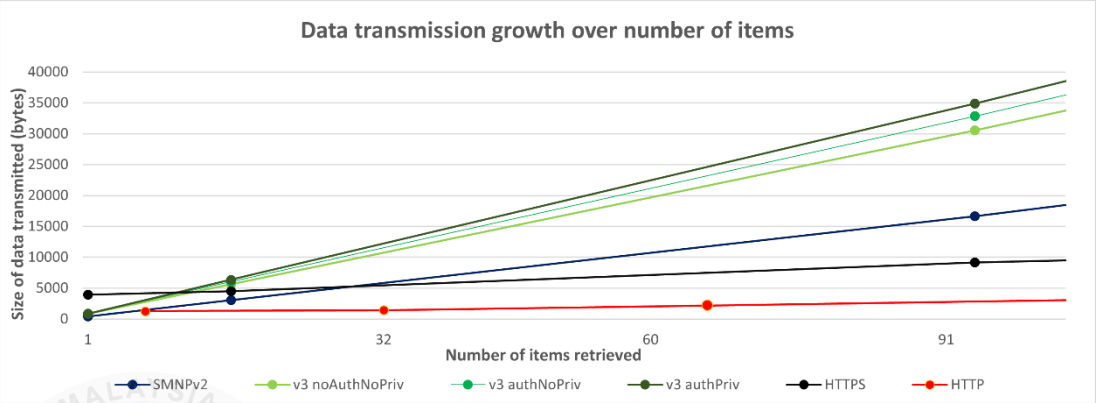

Figure 6.3.5 Overall Data Transmission

The initial observation is that the HTTP consumes the least bandwidth per data retrieved, followed by HTTPS, SNMPv2 and finally SNMPv3, with the authPriv requires the most bandwidth for the same amount of data retrieved. This however does not hold true as HTTPS require a significant amount of data from the encryption negotiation, which makes HTTPS not a very good protocol for small requests. SNMPv3 also requires 254 bytes for the get-request and report, but it does not impact the overall bandwidth required.

Taking the data that was gathered, the efficiency of the protocol is also able to be calculated. The calculation of efficiency is basically taking the data that is intended, the actual data and divide it against the data that is transmitted, making the formula as shown below. This formula gives a general idea of how much the actual data is represented within all of the transmitted data. The table summarizes the result of the calculationefficiency of the protocol

$$\text{Efficiency of data transmission:} = \frac{\text{Actual data}}{\text{Total data transmitted}} \text{ X } 100\%$$

,

Version	Data retrieved	Total Data	Actual PDU	Efficiency
SNMPv2c	2125 bytes	373380 bytes	97451 bytes	26.10%
SNMPv3 noAuthnoPriv	2125 bytes	681931 bytes	97451 bytes	14.29%
SNMPv3 authNoPriv	2125 bytes	734904 bytes	97451 bytes	13.26%
SNMPv3 authPriv	2125 bytes	779520 bytes	97451 bytes	12.50%
RESTful HTTP	1457 bytes	35563 bytes	31135 bytes	87.54%
RESTful HTTPS	2733 bytes	230714 bytes	225201 bytes	97.61%

Table 6.3.1 Efficiency of data transmission

There is a stark difference between the protocols of SNMP and REST. The efficiency of the SNMP is quite abysmal as for every data retrieved, it requires another packet Get-Next to retrieve next packet, that fact decreases the efficiency by a big margin, with even SNMPv3 only able to achieve less than 20 percent of overall data transmitted.

HTTP on the other hand paints a huge difference in regards to the efficiency, where the HTTPS are able to achieve more than 95 percent efficiency despite the massive initial overhead, and the constant ACK packets back to the sender. In terms of the data transmission, RESTful API beats the SNMP by quite a big margin, with the efficiency difference makes little to no sense for SNMP protocol.

The key takeaway in terms of the data transmission is that the HTTP protocol is the most efficient in terms of the required bandwidth to transfer data, and should be preferred in situation that require minimum bandwidth requirement.

The HTTPS equivalent is not preferred when the data set are small especially with the introduction of TLS that makes the initial data query require much more bandwidth than expected. This is made worse when multiple devices are involved, making the header exponentially larger by negotiating TLS for each devices every time the query is done

SNMP returns the most consistent result, where the number of items directly correlate to the bandwidth required. This is a double edge sword as the more data is transmitted, the worse SNMP perform against RESTful API.

6.4 Performance metric

The factor to consider other than the pure bandwidth is the performance metrics. There are myriad of complex performance metrics that measure the protocols extensively, but the focus on this particular environment will be on two main metrics, the time elapsed for each of the process, and also the CPU cycles for the retrieval process.

The testing of the performance metric will be using the program called perf as mentioned earlier. Below shows an example output of a perf command.

```
Performance counter stats for 'snmpwalk -v2c -c psm2 192.168.1.100':

        245.97 msec task-clock                #    0.214 CPUs utilized
         2,187      context-switches          #    0.009 M/sec
             7      cpu-migrations            #    0.028 K/sec
           242      page-faults               #    0.984 K/sec
   615,035,306      cycles                    #    2.500 GHz                      (66.58%)
             0      stalled-cycles-frontend                                       (65.47%)
             0      stalled-cycles-backend    #    0.00% backend cycles idle      (64.93%)
             0      instructions              #    0.00  insn per cycle           (33.42%)
             0      branches                  #    0.000 K/sec                    (34.53%)
             0      branch-misses             #    0.00% of all branches          (35.07%)

     1.150898307 seconds time elapsed

     0.002633000 seconds user
     0.271249000 seconds sys
```

Figure 6.4.1 Example of perf output

The section inside the yellow box shows the command used to test the system. The data that this project will be analyzing is the CPU cycles highlighted in red, and also the time elapsed, highlighted in blue.

The time elapsed in the output refers to the Round-Time trip for the particular command, as mentioned above at Chapter 4. The graph below is the plotting of time elapsed for each command both SNMP and also RESTful API.

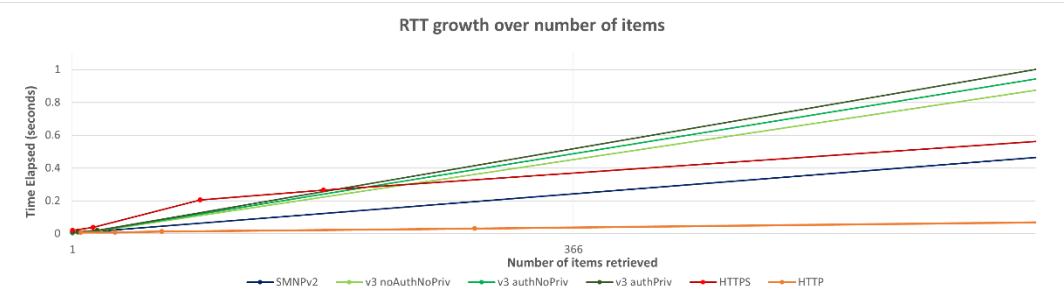

Figure 6.4.2 RTT Growth over number of items

Based on the graph with the items retrieved limited at 750 items, the HTTP protocol seems to require the least amount of time for the same amount of data retrieved, and the SNMPv3 requires the longest time, with the SNMPv3 authPriv requires more than 1 second when the HTTP did not even require 0.1 second of time.

The interesting part is the HTTPS, as observed for small amount of data, HTTPS is extremely inefficient and time consuming as it needs to negotiate the secure channel which makes the initial request time much higher than the SNMPv3. However, as the negotiation only occurs once, the required time is shorter for the subsequent request, and the time required is decreased below the SNMPv3 but still higher than SNMPv2.

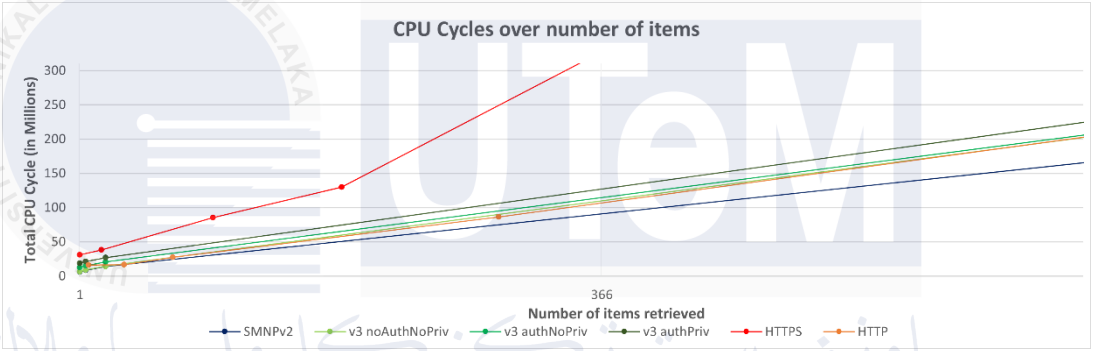

Figure 6.4.3 CPU cycles over number of items

CPU cycles tend to fluctuate as the processor alongside the kernel is constantly optimizing the calculation, hence there will never be a consistent number, especially when different CPU architecture deals with the optimization differently, however the cycles should reflect based on comparison with different command on the same machine and same target.

The result for the CPU paints a slightly different picture for each of the protocol. The SNMP is still the same where the SNMPv2 requires the least amount of CPU cycles, followed by noAuthNoPriv, authNoPriv and lastly authPriv.

RESTful API on the other hand paints a different picture compared to the RTT, where SNMPv2 is slightly more CPU efficient than HTTP based API. The interesting interaction is the HTTPS, where the growth of the CPU cycles requires is quite steep, and based on extrapolation, HTTPS required a staggering 700 million

cycles to complete the same task when the other protocols compared to the other protocols that requires less than 200 million cycles.

The key takeaway for performance is quite interesting, where HTTP gains an edge on the RTT time, where it is extremely responsive, with the SNMP and HTTPS trailing not far behind. SNMPv3 requires more time and REST is more preferred in terms of responsiveness.

SNMPv2 requires the least CPU cycle, with HTTP protocol performing similarly between the SNMPv2 and SNMPv3.

HTTPS on the other hand portrays a very different picture in CPU performance. The CPU efficiency of the HTTPS is sacrificed for the gains in terms of RTT and especially data size that is much efficient than any protocol. TLS and encryption also play in as a factor on why CPU usage is extremely high.

6.5 Security Parameters

SNMP and RESTful API are protocols that transmit monitoring data to the relevant agent and programs, and some information such as the IP address should be kept confidential and encrypted to prevent anyone eavesdropping.

SNMPv2 and HTTP based RESTful API is extremely simple to configure with a robust track record as it was widely used during the infancy of the internet. SNMP does provide some basic security system namely the community-based approach, where a community string is set on the device, and the query requires the exact same community string. HTTP also provides a more robust authentication method, from the Basic username password, token based, OTP or even RSA key to authenticate the request before actually sending the data, hence on the first glance HTTP gains an edge compared to SNMPv2

Although it serves the purpose as networking monitoring protocol, it inherently does not provide much security to the transmission of the data, especially when the data are transmitted through plaintext form, which makes it basically an

insecure protocol, hence it should never be used on internet-facing devices in general.

Below is the example of Wireshark eavesdropping the HTTP and SNMP traffic. Do take note of the plaintext nature of both the protocols, and the plaintext data that was captured inside the red boxes.

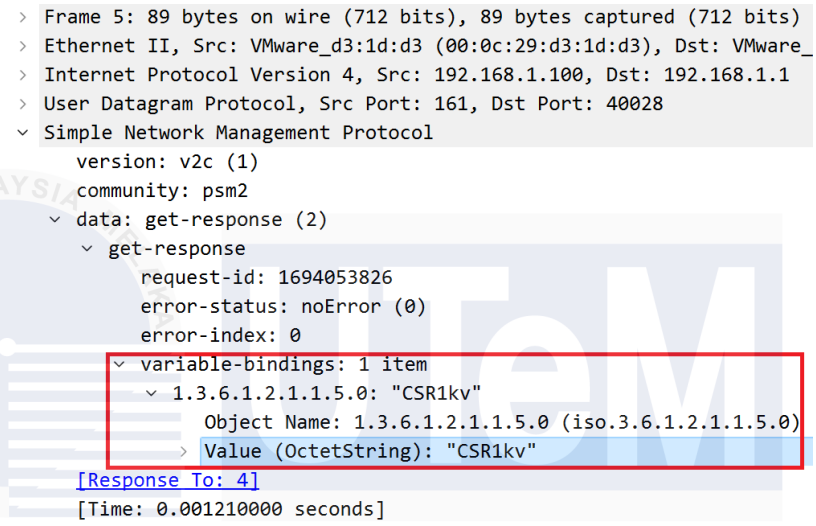

Figure 6.5.1 Example of Wireshark packet capture SNMPv2c

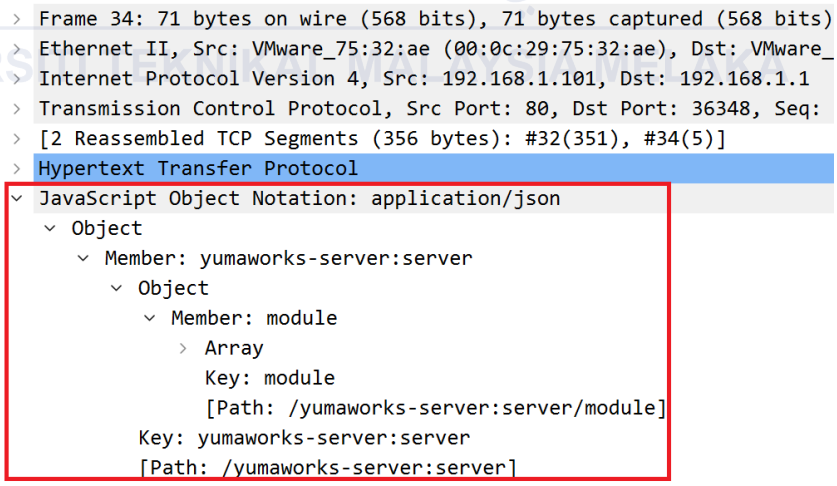

Figure 6.5.2 Example of Wireshark packet capture REST HTTP

SNMPv3 is introduced as the next iteration of the protocol, with improvement of the MIB, but more importantly the security of the data transmission. Although there are three options for the SNMPv3, which are noAuthNoPriv, authNoPriv and authPriv, but the implementation of all three of the protocols yield massively different effect in terms of security.

The noAuthNoPriv have the same problem as the SNMPv2 and HTTP, as the name suggests, no authentication and no privacy or encryption included in the transmission, hence it suffers the weakness of eavesdropping.

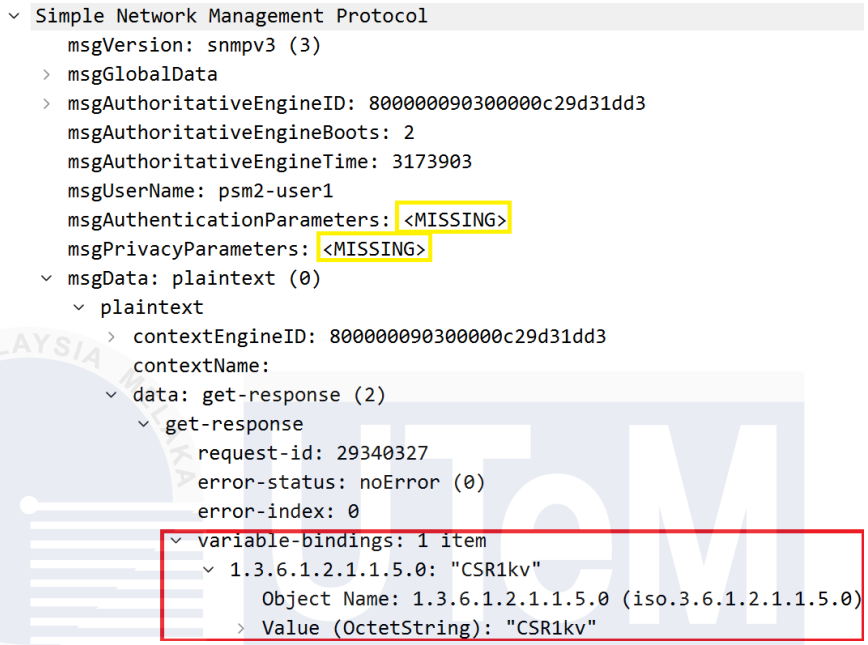

Figure 6.5.3 Example of Wireshark packet capture SNMPv3 noAuthNoPriv

authNoPriv option for the SNMPv3 is a bit secure as it requires an authentication key before the server replies to the query, but still suffer the same problem as all of the protocol above, where the confidentiality is not preserved, and the authentication is just to make sure the credentials provided are correct before passing the values. Notice the difference between the authNoPriv and noAuthNoPriv, where both the data are in plaintext form, the authentication for authNoPriv is included on the green box.

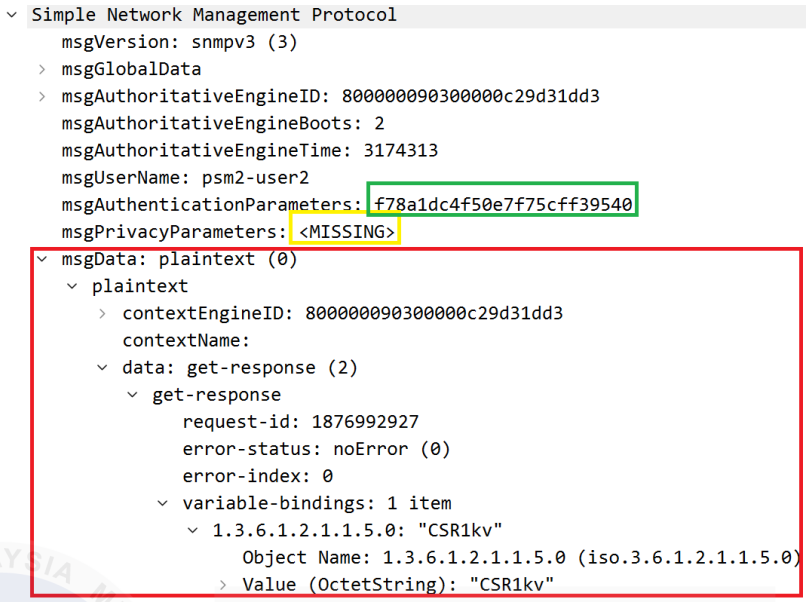

Figure 6.5.4 Example of Wireshark packet capture SNMPv3 authNoPriv

The lack of the encryption or confidentiality makes the 4 options, namely RESTful HTTP API, SNMPv2, SNMPv3 noAuthNoPriv and SNMPv3 authNoPriv not recommended on internet-facing devices, or even in any environment that sends sensitive information through the network. This leaves the desired SNMP option to be the SNMPv3 authPriv configuration and the HTTP Secure (HTTPS) protocol.

SNMPv3 authPriv is considered the golden standard for the SNMP protocol, and as the name suggests, it requires authentication and privacy for the SNMP packets. This means in order for the SNMP to return the monitoring data, it is required to send the authentication data, either in the form of MD5 or SHA. When the request is authorized, the PDU of the data is encrypted by the configuration choice of the device, either in the form of DES or AES. The output for a transaction in SNMP captured from Wireshark is shown below.

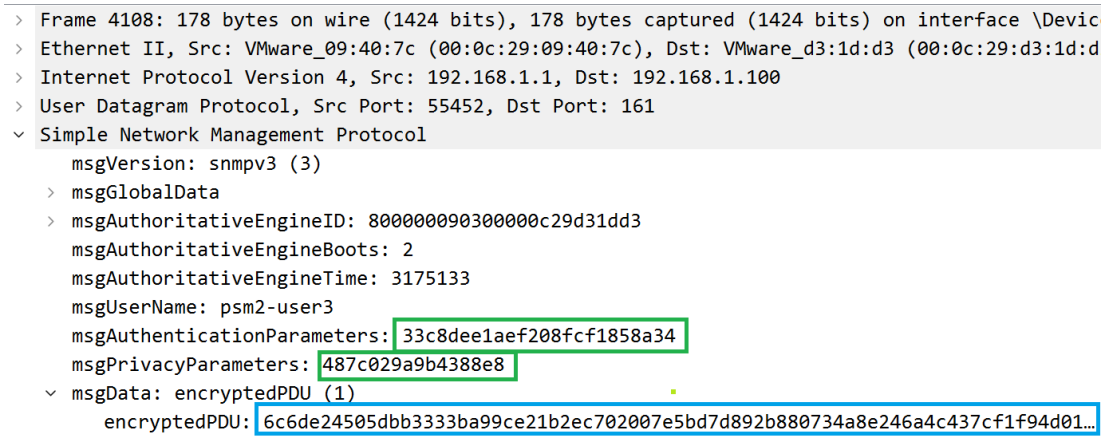

Figure 6.5.5 Example of Wireshark packet capture SNMPv3 authPriv

On first glance, SNMPv3 achieved the initial objective of the authentication and privacy of the data, where authentication is required and the encryption is done on the actual monitoring data, and it mostly done what it was supposed to do. However, there is one quite big problem with this approach in particular, namely the USM of the SNMP.

There is a huge problem with the implementation of the USM model in SNMPv3, with the glaring issue is the inclusion of few really critical information within the fields. The USM model contains the msgAuthoritativeEngineBoots and msgAuthoritativeEngineTime, both of the values represent the number of reboots done on the device, and also the uptime of the device. This value is returned after the device query by sending the get-request packet. The inclusion of the values in effect means that even if the authentication and privacy parameters is incorrect, or any unsolicited messages sent, the device will return both of the values, which does not make much sense.

The intention to include both the fields are to prevent replay attack where the request is replayed on a different time, which the boot time is not consistent and SNMP could reject. The exposure of both the data is quite dangerous as sophisticated attacker could use the data above instead to uniquely identify and fingerprint the SNMP-enabled device.

The problem is made worse when the msgUserName is in plaintext form, essentially making the brute force of the device easier. The figure below basically

simulates an unsolicited request with wrong username and password, and although the data is encrypted similarly as the request above, the fields highlighted are the information that potential attackers are interested instead and could be used against the SNMP device.

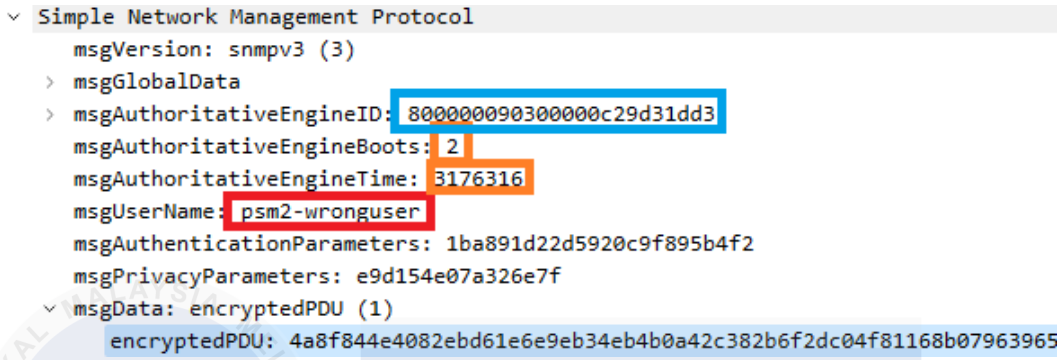

Figure 6.5.6 Wireshark packet capture of unsolicited SNMPv3 authPriv

Another big problem is the implementation of the SNMP Engine ID. Given that the Engine ID is 80:00:00:09:03:00:00:0c:29:d3:1d:d3, taken from the same unsolicited message highlighted in blue, the steps below are the steps an attacker will take to determine more information regarding the device.

1. The first four octets are 80:00:00:09, with the enterprise ID as 9, where the IANA search shows it as Cisco devices.

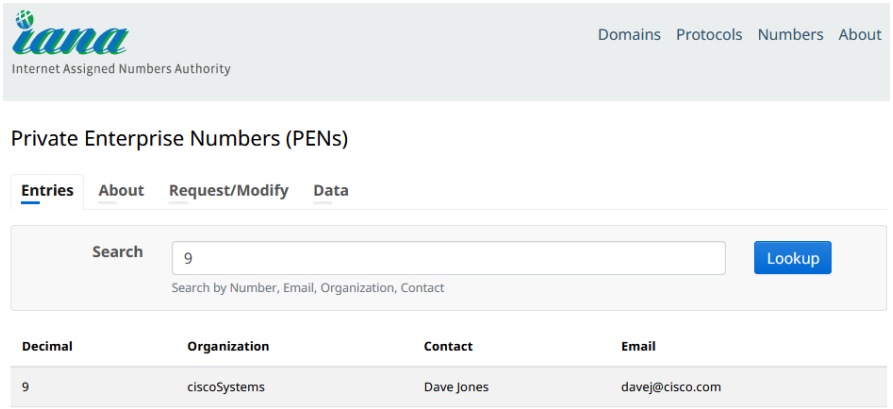

Figure 6.5.7 IANA Search for Private Enterprise ID

2. The fifth octets are 03, which means the SNMP Engine ID is determined by MAC Address.

3. The remaining octets are 00:00:0c:29:d3:1d:d3, considering MAC address is 6 octets, and the leading two octets are padded with 0, the MAC shown is 00:0c:29:d3:1d:d3, and it exactly matches with the MAC address below

```
CSR1kv#sh ip int br
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet1       192.168.1.100   YES manual up                    up
CSR1kv#sh int g1
GigabitEthernet1 is up, line protocol is up
  Hardware is CSR vNIC, address is 000c.29d3.1dd3 (bia 000c.29d3.1dd3)
  Description: VBox
  Internet address is 192.168.1.100/24
```

Figure 6.5.8 MAC Address for the CSR1kv

The attacker could gain quite some information regarding the device just by getting the Engine ID value, which is actually easy to get by just sending an unsolicited SNMP packet.

The mentioned inherent security weakness of the implementation of SNMP creates a problem and the need of an alternative protocol to complement or even replace it. HTTPS is considered to be the replacement and the next protocol to be used as network monitoring.

HTTPS is considered much secure than all aforementioned protocols as the method to secure the traffic does not use any password or pre-shared key, instead it uses RSA cryptographic key, either symmetric or asymmetric public key infrastructure.

Without getting too complicated, the idea is that both of the client and server negotiates a secure channel using a cryptographic key. The key is verified by a third party called Certificate Authority, where the entity is tasked only to create, and revoke the key if necessary. The server will use the keys created by the Certificate Authority where both parties trust. If any of the process is tampered, there are multiple warnings on both sides, from the negotiation process, cipher suite to the issuer of the certificates. This creates another layer of security as the only data transmitted is the security protocol and certificates, which could be consider public information.

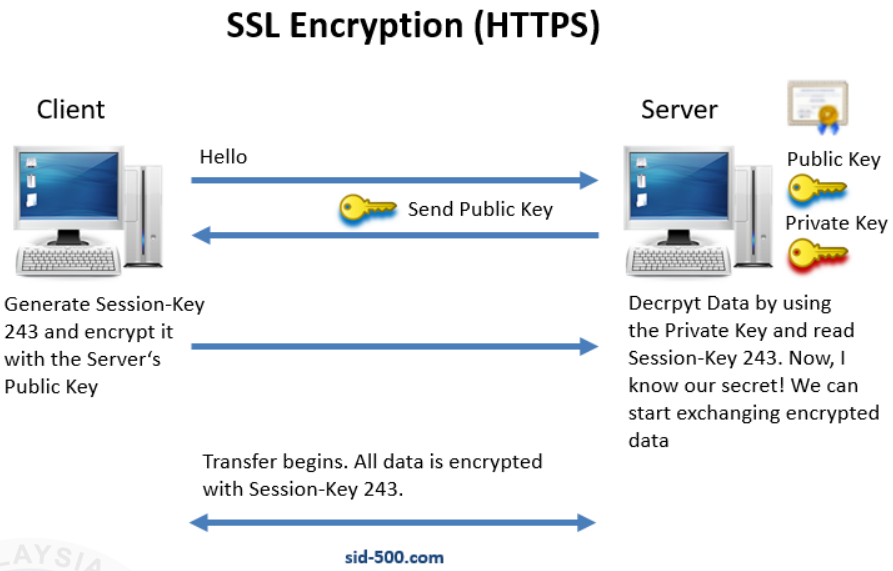

Figure 6.5.9 Cryptographic key negotiation process, courtesy of Patrick Gruenauer

To summarize, when discussing the security factors, HTTPS in most of the situation is the preferred protocol in addition to being used widely for a multitude of purposes. SNMPv3 authPriv is a bit interesting because although it achieves confidentiality and authentication, few fundamental flaws make the protocol less desirable than the HTTPS.

Although SNMPv3 authNoPriv and HTTPS does come with the ability to authenticate the request, the transmission of the information is not encrypted and susceptible to packet capture. SNMPv2 and SNMPv3 noAuthNoPriv is less than desirable, with packet capture able to even capture the community-string in plaintext that is the security basis of both of the packets.

CHAPTER 7: CONCLUSION

7.1 Introduction

This project is research about network monitoring protocols of SNMP and REST API. The project starts with highlighting few of the key objective and scope of the project. It continues with the research and development of idea based on the topic. This project will involve some testing based on a lab environment, hence the testing methodology is highlighted. The design of the environment is then sketched with the relevant configurations. Implementation and also the expected output is mentioned, with relevant screenshots and commands. Based on the implementation, data is gathered and analysis is done in line with the objective and scope.

7.2 Project Summarization

The project basically aims to explore the fundamental differences between the two protocols in terms of the architecture, how the protocol works. The protocol shows a very different approach and all of the detailed findings are explained on Chapter 6: Testing and Analysis

The interaction between SNMP uses an inefficient way of asking for the next data after each data retrieved. The REST API approaches it using a single URL with relevant HTTP Header, and the data is returned in JSON format. The different architecture will result in different bandwidth requirement, and the data sizes are

calculated, taking into account of the number of data retrieved. One of the interesting findings is that SNMP is much better for smaller number of data retrieved, with HTTP and TCP header for REST gaining advantage on a larger number of data retrieved, with HTTPS having an impressive 97 percent compared to SNMP 12.5 percent.

The performance is then measured, based on few basic parameters, namely the Round Time Trip for each protocol, and also the CPU cycles. Round Time Trip is quite straightforward and RESTful API takes the lead as there is only one request and one response, compared to SNMP sends get-next-response after getting the current value, which causes the elapsed time to be much higher.

CPU cycles is a different story, with the difference between HTTPS and SNMP shows a massive gap. HTTPS sacrifices the CPU cycle with complicated encryption and optimization, in turn gains in term of actual bandwidth and time required for the same amount of item retrieved.

Security is a big topic and this project only scratch the surface for each security parameters for each protocol. The HTTP protocol, SNMPv2, SNMPv3 noAuthNoPriv and SNMPv3 authNoPriv does not fare well in terms of privacy as all of the data transmitted is in plaintext. HTTP and SNMPv3 authNoPriv contains the mechanism to authenticate the user that queries the device which makes it slightly more secure. SNMPv2 uses community string and SNMPv3 noAuthNoPriv uses USM username to authenticate without password, which makes them worse than the two aforementioned above.

There are few interesting discussions regarding the SNMPv3 authPriv option, and although the SNMPv3 achieved the authentication and privacy part of the PDU data, the implementation of authPriv actually makes the device more vulnerable, as there are many unnecessary data transmitted within the USM model. Attackers could gain valuable information from the exposed data. The data that is included is also not encrypted which partially defeats the purpose of the authPriv aspect.

The implementation of HTTPS is considered secure as majority of the web from the small blog to the massive conglomerate of the likes of Google and Amazon uses it. The security of HTTPS is always on the radar of many security researchers, constantly finding the vulnerability of HTTPS.

7.3 Project Contribution

This project is helpful for network engineers to understand the differences between the SNMP and API. Based on the information, particularly on the performance, data and security, a comprehensive decision could be made in designing the network for a particular company.

This project also put some emphasis on the flow of the data between two devices for both the SNMP and also the RESTful API. In this ever-changing industry, SNMP is starting to show its age and some institutions are embracing the RESTful API in their network monitoring. Universities should look into the potential of this, and make more contributions on the protocol, improving this idea alongside the industry in making RESTful API a better protocol.

7.4 Project Limitation

There are few limitations exist in this project. One of the glaring issues is that the over-emphasis of the security weakness of SNMP. There are also some weaknesses of the HTTPS if configured incorrectly such as using self-signed certificate, difference between asymmetric and symmetric encryption algorithm, minimum cipher suite and so on. The inclusion of the topics is out of the scope of this project. The security part is also run through a simple analysis that is able to be gathered by any packet capture software.

The lack of variety during performance measurement is also one of the limitations. There are quite some parameters that, although will not affect the overall result, some small changes are not accounted might slightly alter the result, such as the uptime, the testing in a virtualized environment that is also running other tasks. As mentioned, there might also be some discrepancies between Linux and Cisco Systems, with the bandwidth is different in the factor of thousands.

The data points measured are also lacking and some extrapolation is done in order to get a general view, hence the graph functions more like a trendline instead of an actual representation on how the devices perform under the same condition.

7.5 Future Works

The performance could be measured in a more comprehensive way, where physical device is used to replicate the actual production environment and minimize noises due to virtualization of the devices.

The security part of the project could be expanded, increasing the scope of the HTTPS weakness, and more importantly the possible attack surface of both protocols, the flaws or mis-configuration of devices that attacker can exploit. Mitigation techniques could also be introduced to prevent aforementioned attacks from happening.

The data transmission only applies to one VM to another. Few parameters could be tweaked to observe the changes in terms of data size, with one such parameters being the MTU and TCP MSS of the interface. Effects such as high jitter on interface could also be introduced, or even packet loss and how it influences the actual data sent and received by devices.

REFERENCES

Siggins, M. (2020, March 3). Avoid legacy SNMP threats with snmpv3. DPS Telecom. https://www.dpstele.com/blog/avoid-legacy-snmp-threats-with-snmpv3.php

Albakour, T., Gasser, O., Beverly, R., & Smaragdakis, G. (2021). Third Time's not A charm. Proceedings of the 21st ACM Internet Measurement Conference. https://doi.org/10.1145/3487552.3487848

Aravind, H S, et al. International Conference on Signal, Image Processing, Communication and Automation (ICSIPCA 2017). Grenze, 2018, thegrenze.com/index.php?display=page&view=conferenceabstract&absid=1103&id=53. Accessed 16 May 2023.

Roughan, M. (2010). A case study of the accuracy of SNMP measurements. Journal of Electrical and Computer Engineering, 1–7. https://doi.org/10.1155/2010/812979

Adams, T. (2015, March 17). Fire, the art of war, and SNMP vs. API monitoring. LinkedIn. https://www.linkedin.com/pulse/fire-art-war-snmp-vs-api-monitoring-todd-adams. Accessed 16 May 2023.

Welling, J. (2022, February 10).Api Vs SNMP VS CLI: The best choice for network devices. API vs SNMP vs CLI: The Best Choice for Network Devices. https://www.cbtnuggets.com/blog/technology/networking/api-vs-snmp-vs-cli-the-best-choice-for-network-devices. Accessed 16 May 2023.

Gamess, E., & Hernandez, S. (2021). Performance evaluation of snmpv1/2c/3 using different security models on Raspberry Pi. International Journal of Advanced Computer Science and Applications, 12(11). https://doi.org/10.14569/ijacsa.2021.0121101

Arlos, Patrik & Fiedler, Markus & Tutschku, Kurt & Chevul, Stefan & Nilsson, Arne. (2002). Obtaining Reliable Bit Rate Measurements in SNMP-Managed Networks.

Pras, A., Drevers, T., van de Meent, R., & Quartel, D. (2004). Comparing the performance of SNMP and web services-based management. IEEE Transactions on Network and Service Management, 1(2), 72–82. https://doi.org/10.1109/tnsm.2004.4798292

Oh, YJ., Ju, HT., Choi, MJ., Hong, J.WK. (2002). Interaction Translation Methods for XML/SNMP Gateway. In: Feridun, M., Kropf, P., Babin, G. (eds) Management Technologies for E-Commerce and E-Business Applications. DSOM 2002. Lecture Notes in Computer Science, vol 2506. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-36110-3_8

R. Neisse, R. L. Vianna, L. Z. Granville, M. J. B. Almeida and L. M. R. Tarouco, "Implementation and bandwidth consumption evaluation of SNMP to Web services gateways," 2004 IEEE/IFIP Network Operations and Management Symposium (IEEE Cat. No.04CH37507), Seoul, Korea (South), 2004, pp. 715-728 Vol.1, doi: 10.1109/NOMS.2004.1317759. https://ieeexplore.ieee.org/document/1317759

Alnafjan, K. & Khan, Gul & Hussai, Tazar & Ullah, Hanif & Alghamdi, Abdullah. (2012). Behavior based Comparative analysis of XML and JSON web technologies.

Bresciani, M. (2017, March 17). RESTful SNMP over http: Part I - dzone. RESTful SNMP Over HTTP. https://dzone.com/articles/restful-snmp-over-http Accessed at 2023 May 15

O. Bergmann, C. Bormann and S. Gerdes. (2020). "REST-based access to SMIv2-structured information on constrained devices," 2014 IEEE Network Operations and Management Symposium (NOMS), Krakow, Poland, 2014, pp. 1-5, https://doi.org/10.1109/NOMS.2014.6838367

WS-REST '10: Proceedings of the First International Workshop on RESTful Design. (2010). New York, NY, USA: Association for Computing Machinery.

A. Bierman, M. Bjorklund, K. Watsen, and R. Fernando, "RESTCONF Protocol," Internet Engineering Task Force, Internet-Draft draft-bierman-netconf-restconf-03, December 2013, work in progress, https://doi.org/10.17487/RFC8040

Choi, M.-J., Hong, J. W., & Ju, H.-T. (2003). XML-based network management for IP Networks. ETRI Journal, 25(6), 445–463. https://doi.org/10.4218/etrij.03.0103.0062

W. Stallings, "SNMP and SNMPv2: the infrastructure for network management," in IEEE Communications Magazine, vol. 36, no. 3, pp. 37-43, March 1998, doi: 10.1109/35.663326.

J. Liu and G. Liu, "Research and Implementation of SNMP-Based Network Management System," 2011 4th International Conference on Intelligent Networks and Intelligent Systems, Kuming, China, 2011, pp. 129-132, doi: 10.1109/ICINIS.2011.39.

A. Gupta and R. Bartos, "User Experience Evaluation of HTTP/3 in Real-World Deployment Scenarios," 2022 25th Conference on Innovation in Clouds, Internet and Networks (ICIN), Paris, France, 2022, pp. 17-23, doi: 10.1109/ICIN53892.2022.9758130.

Berners-Lee, T., Fielding, R., and H. Frystyk, "Hypertext Transfer Protocol -- HTTP/1.0", RFC 1945, DOI 10.17487/RFC1945, May 1996, https://www.rfc-editor.org/info/rfc1945.

Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

Y. Einav, Amazon Found Every 100ms of Latency Cost them 1 % in Sales, [online] Available: https://www.gigaspaces.com/blog/amazon-found-every-100ms-of-latency-cost-them-1-in-sales/. Accessed at 30 May 2023 22:05 UTC +8

S. M. Sohan, F. Maurer, C. Anslow and M. P. Robillard, "A study of the effectiveness of usage examples in REST API documentation," 2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), Raleigh, NC, USA, 2017, pp. 53-61, doi: 10.1109/VLHCC.2017.8103450.

H. Wenhui, H. Yu, L. Xueyang and X. Chen, "Study on REST API Test Model Supporting Web Service Integration," 2017 ieee 3rd international conference on big data security on cloud (bigdatasecurity), ieee international conference on high performance and smart computing (hpsc), and ieee international conference on intelligent data and security (ids), Beijing, China, 2017, pp. 133-138, doi: 10.1109/BigDataSecurity.2017.35.

M. Jethanandani, "YANG, NETCONF, RESTCONF: What is this all about and how is it used for multi-layer networks," 2017 Optical Fiber Communications Conference and Exhibition (OFC), Los Angeles, CA, USA, 2017, pp. 1-65.

R. T. Fielding, R. N. Taylor, J. R. Erenkrantz, M. M. Gorlick, J. Whitehead, R. Khare, et al., "Reflections on the REST Architectural Style and", Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering ser. ESEC/FSE, pp. 4-14, 2017, 2017, [online] Available: http://doi.acm.org/10.1145/3106237.3121282.

J. R. Erenkrantz, M. Gorlick, G. Suryanarayana and R. N. Taylor, "From representations to computations: The evolution of web architectures", Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering ser. ESEC-FSE '07, pp. 255-264, 2007, [online] Available: http://doi.acm.org/10.1145/1287624.1287660.

A. Archip, C. -M. Amarandei, P. -C. Herghelegiu, C. Mironeanu and E. şerban, "RESTful Web Services – A Question of Standards," 2018 22nd International Conference on System Theory, Control and Computing (ICSTCC), Sinaia, Romania, 2018, pp. 677-682, doi: 10.1109/ICSTCC.2018.8540763

Fielding, R. T. (2000). Architectural styles and the design of network-based software architectures. Publication Unpublished doctoral dissertation , University of California, Irvine .

Blumenthal, U. and B. Wijnen, "User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)", RFC 2574, DOI 10.17487/RFC2574, April 1999, <https://www.rfc-editor.org/info/rfc2574>.

Case, J., Fedor, M., Schoffstall, M., and J. Davin, "Simple Network Management Protocol (SNMP)", RFC 1157, DOI 10.17487/RFC1157, May 1990, <https://www.rfc-editor.org/info/rfc1157>.

Schoenwaelder, J., "Overview of the 2002 IAB Network Management Workshop", RFC 3535, DOI 10.17487/RFC3535, May 2003, <https://www.rfc-editor.org/info/rfc3535>.

ISO: Software product quality model—iso25010. <https://iso25000.com/index.php/en/iso-25000-standards/iso-25010> Accessed 12 September 2023

Abdeen, W., Chen, X. & Unterkalmsteiner, M. An approach for performance requirements verification and test environments generation. Requirements Eng 28, 117–144 (2023). https://doi.org/10.1007/s00766-022-00379-3