Expertise
and insight
for the future

Mats Nordin

Implementing a monitoring system using PRTG

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information and Communication Technology

Thesis

25 August 2021

Tekijä(t) Otsikko Sivumäärä Aika	Mats Nordin PRTG-pohjaisen monitorointijärjestelmän käyttöönotto 38 sivua 25.8.2021
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tieto–ja viestintätekniikka
Ammatillinen pääaine	IoT ja Cloud computing
Ohjaaja(t)	Head of Infrastructure Solutions, Timo Haatainen Lehtori, Marko Uusitalo

Opinnäytetyö tehtiin Empower IM Oy Infrastruktuuri -osastolle. Empower IM Oy:llä oli entuudestaan ulkoiselta toimittajalta valvontajärjestelmä joka oli ajan saatossa muuttunut vaikealukuiseksi ja vaikeaksi ylläpitää. Kun toimittajan muista palveluista luovuttiin, haettiin uutta järjestelmää ja päädyttiin Paesslerin PRTG valvontajärjestelmään. Opinnäytetyön toimeksiannoksi annettiin suunnitella ja toteuttaa yrityksen palvelimien ja verkkolaitteiden valvontajärjestelmä käyttäen PRTG ohjelmaa.

Työn keskeisimpiä tavoitteita oli saada uudesta valvontajärjestelmästä helposti hallittava, jossa kokonaisuudet olisivat loogisesti jaoteltu ja järjestelmän ylläpito olisi mahdollisimman selkeää. Laitteiden ja palvelinten lukumäärän huomioiden oli tärkeää myös saada järjestelmä toimimaan mahdollisimman tehokkaasti. Tätä varten mahdollisimman suuri osa valvonnoista tehtiin hyödyntäen SNMP -protokollaa. SNMP -protokolla mahdollisti valvonnan suorittamisen eri järjestelmissä samalla tavalla järjestelmästä riippumatta.

Valvonnan kannalta olennaisia periaatteita ja toimintapoja hyödyntämällä luotiin toimiva pohja valvontajärjestelmälle, jonka päälle pystyttiin lisäämään valvottavia kohteita hyödyntäen SNMP -protokollaa ja standardisointia. Valvontajärjestelmän ominaisuuksia hyödyntäen pystyttiin luomaan standardisoitu pohja jonka päälle oli helpompaa yksilöidä asiakastarpeisiin sopivat ratkaisut. Lopputuloksena saatiin toimiva ratkaisu yrityksen laitteiden valvontaan, mitä on mahdollista laajentaa ja kehittää uusien hankkeiden mukaiseksi.

Avainsanat	PRTG, SNMP, monitorointijärjestelmä, palvelinvalvonta

Author(s) Title Number of Pages Date	Mats Nordin Implementing a monitoring system using PRTG 38 pages 25 August 2021
Degree	Bachelor of Engineering
Degree Programme	Information and Communication Technology
Professional Major	IoT and Cloud computing
Instructor(s)	Timo Haatainen, Head of Infrastructure Solutions Marko Uusitalo, Senior Lecturer

This thesis was done for the infrastructure team of Empower IM Oy. Empower IM previously had a monitoring solution from a service provider that was hosted on their site. This monitoring software proved to be hard to read and maintain due to lack of strategy in the building phase. This project was necessary when Empower IM changed its service provider and chose to buy their own monitoring software. The chosen monitoring software was Paessler Router Traffic Grapher (PRTG) by Paessler, the same as before, but straight from the vendor. The goal of the thesis was to design and execute structured monitoring software with easy readability and expandability.

The key points of the project were to make sure that the new monitoring software was easily maintained, where the teams responsible for the services would be able to maintain them themselves. This meant that the hierarchy of the servers had to be arranged in a certain way in order for this to be possible. In addition to that, it would be crucial for system administrators in case of emergencies to see the inheritances of servers and to be able to locate all of the necessary servers and services easily in server maintenance situations. Secondly, considering the amount of devices and servers that are in use, it was crucial to be able to monitor them effectively with as low overhead as possible. This meant that the monitoring would be done using Simple Network Management Protocol (SNMP) where applicable. Using SNMP on all servers streamlined the process as well as made it more lightweight on the monitoring software. SNMP makes sure the process is the same across all different distributions and devices.

This project covers the essential rules of monitoring with SNMP and how effective monitoring habits increase productivity. It describes how PRTG operates and how to prepare distributions in advance to populate PRTG. Using the systemic approach in monitoring and SNMP, it was easier to make clear environments to the PRTG of each customer platform.

Keywords	PRTG, SNMP, monitoring system

Contents

List of Abbreviations

API	Application Programming Interface
ASN.1	Abstract Syntax Notation One
CPU	Central Processing Unit
CSV	Comma-Separated Values
DC	Domain Controller
DISM	Deployment Image Servicing and Management
DNS	Domain Name System
GID	Global ID
HTTP	Hypertext Transfer Protocol
I/O	Input/Output
IAB	Internet Activities Board
IETF	Internet Engineering Task Force
IP	Internet Protocol
MD5	Message-digest algorithm
MIB	Management Information Base
NOC	Network Operations Center
OID	Object Identifier
PDU	Protocol Data Unit
PRTG	Paessler Router Traffic Grapher
RFC	Request for Comments
RHEL	Red Hat Enterprise Linux
SED	Stream Editor
SHA	Secure Hash Algorithm
SMI	Structure of Management Information
SNMP	Simple Network Management Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URL	Uniform Resource Locator
WMI	Windows Management Instrumentation

Glossary

Cmdlet	lightweight command that is used in the PowerShell environment

1 Introduction

Network monitoring has become a crucial part in the battle for high availability. High availability is a core concept in the modern information technology as most of the time servers and implementations are running off-premise and hosted by a contractor. Both of these have to be operational for the business to be working. Companies are trying to find out less laborious ways to keep track and react to possible problems regarding the ever-growing server and network amount. With monitoring of the infrastructure, companies like Empower Oy can deliver service level agreements on customer demand. Many have recognized the importance of good monitoring software and have begun offering proprietary and open-source alternatives.

Empower Oy is one of the leading energy sector companies in Finland. Empower Oy has approximately 1,200 employees across four different divisions, each specializing in different aspects of the energy sector. These divisions are responsible for creating, maintaining and monitoring electricity networks and telecommunication infrastructure across Finland. Empower Oy also builds wind turbine farms and has a key role in the energy metering industry in Finland. This study has been commissioned by Empower IM Oy, which is a division of Empower Oy.

Empower IM Oy previously had a monitoring solution from a data center service provider based on PRTG by Paessler. All of the patching and administrative work was done by the service provider, which is one extra step in the supply chain. When Empower decided to part ways with them, the monitoring solution would also be included, which prompted a conversation to find a working solution. The whole swap was to be done swiftly, because the access for the old monitoring software would be cut out after a month, when the deal was closed. The most logical way forward would be to use the same monitoring software than previously, but self-hosted. Since Empower Oy is a bigger company, there are multiple instances of PRTG running simultaneously. This thesis focuses on Empower IM's probe, which is PRTG's way of saying it is a standalone instance. The reason is that it is the only probe that is totally owned by Empower IM Oy and hosted in their data center cluster. The first principal point of developing successful monitoring is to utilize

SNMP as much as possible, given it is lightweight and support is plenty. The second is to arrange the hierarchy smart as it is important to find the correct servers and sensors. Every month there is a maintenance to update the servers. During this maintenance the sensors affected should be paused, and having a smart hierarchy makes pausing the correct machines straightforward. Hierarchy also makes managing the user groups easier since the permissions can be defined mostly on the top level.

2 Theoretical background

Before the times of automated monitoring software, the monitoring was done by system specialists. They were responsible for scouring the machines. This was an immense amount of manual labor placed on one person. Often times, this approach would only yield results when a service or server was down and a customer would be contacting. There was no way the specialist could proactively react to the problems. Information technology has transformed through the years from financial liability into a necessity for businesses. In the past decade or so, there has been a shift into virtual machines, which increases the need for monitoring, when the deployment of servers is easier than ever. Although the raw amount of machines has grown, the teams usually consist of only a few people and in the most extreme cases only of one person. Thus, monitoring effectively has become such a crucial part of administration. Thought out monitoring can provide more insight into the systems, where even a slight downtime could prove too costly. For example, it is estimated that data center outages can result in 8,000 US dollars of loss per minute [1].

2.1 Many ways of monitoring

When trying to build a robust implementation for monitoring with scalability, the most important factors are meaningful metrics, visualization and alarms [2]. Metrics are measures of data provided by the systems being monitored. They need to have a value, timestamp and a state for them to be useful in monitoring, and to even be considered as metrics. The metrics alone are not sufficient enough for monitoring effectively. If specialists had only metrics for monitoring their systems, they would be overloaded with information. Most of the common data is not useful in detecting anomalies and faults. This data is often called blackbox monitoring. Blackbox monitoring can be summed up as follows:

> Monitoring software is using the same data as the user sees. For example, monitoring Central Processing Unit (CPU) usage. [2]

It is important to remember that not all blackbox monitoring is bad, but adding copious amounts should be avoided since it will clutter the view. Being critical about what needs to be monitored is the most crucial part of making a monitoring solution. Monitoring the virtual memory usage of an important business logic server is often required, but having the same sensor for a development machine is probably not needed. Use cases vary which is why a clear plan on what needs to be achieved from the monitoring is strongly suggested. To get the most out of monitoring, the metrics specialists want to spend their time on whitebox monitoring. Sometimes referred to as glassbox monitoring, it is loosely put, the opposite of blackbox monitoring. The data gathered by this kind of monitoring is tailored to specific use-cases and it provides accurate and meaningful data which is useful when detecting faults. For example, to monitor databases effectively there could be a custom-made script that monitors the database and relays the data into monitoring software. These take more planning and time to make, but by using whitebox monitoring more than blackbox monitoring, over-monitoring can be prevented which can quickly lead to alert fatigue [2].

Over-monitoring is a phenomenon that takes place usually in the early development stages of monitoring and is often caused by placing too many unnecessary sensors for data [2]. In monitoring, especially in larger implementations, the quality of the metrics is indefinitely more important than the quantity. If the over-monitoring is not dealt with, it can render the monitoring software useless — at least in the eyes of the users. This can lead to missing of important information and the users responsible for checking the alerts becoming exhausted by the sheer volume of alerts. If the users cannot rely on the messages sent by the monitoring software, what is the software good for?

The basics of monitoring have been covered and short rules have been established on what to avoid and what to look for. Subsequently, the types of metrics utilized in well-balanced monitoring solutions are analyzed with.

2.1.1 Gauges

Starting with the common metric, gauges show the current value of a specific measurement. In this context, the sensor is measuring CPU usage every 5 minutes. Gauges play

an important role in the history of the sensor. Using SNMP, gauges have a maximum value of 2^{32}-1 (4294967295 decimals) [3]. With gauges, the moment when faults happen can be pinpointed very precisely since they represent only one value at a time as shown in Figure 1. In addition, it is one the most powerful tools for visualizing patterns. For this matter, there should be gauges in at least the most crucial parts of monitoring. As shown in Figure 1, Windows Management Instrumentation (WMI) provides extensive information of a disk's Input/Output (I/O). Although gauges can lack depth in terms of accurate data, viewing the pure numerical data tends to be cluttered and busy especially in cases where there are a lot of data coming through, like in Figure 2.

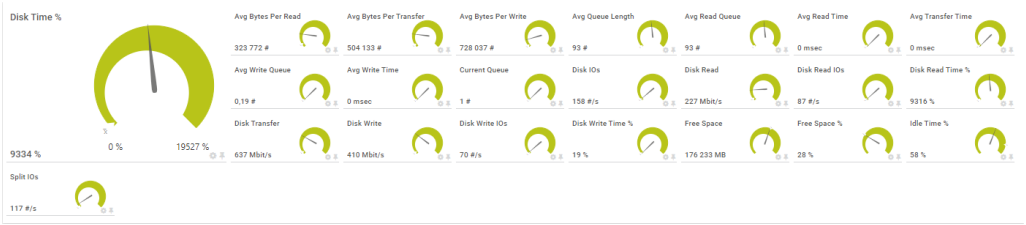

Figure 1: Gauges on a WMI I/O monitoring

A fair consideration that comes from visualization and histories of data is where to store it. For example, in Figure 2 when the sensor polls the CPU every 5 minutes for 365 days there are 1 261 440 entries for all of the sensors in the figure, which have to be stored somewhere. From the entries only a handful are going to be useful in the long run. However, if the non-essential data is deleted to free up space then the history would not be complete and it would become unusable.

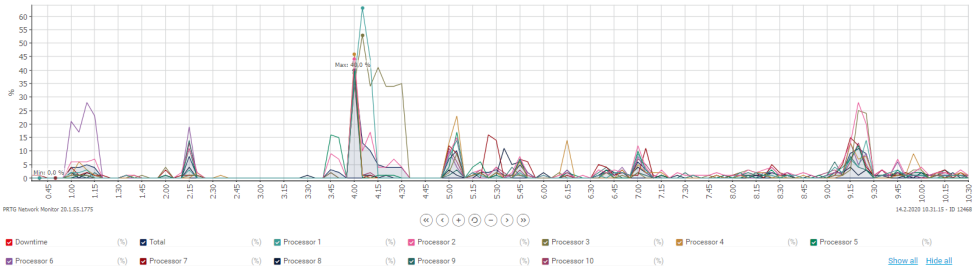

Figure 2: History of CPU gauges using SNMP

2.1.2 Counters

Counters are values that increase incrementally. They will never decrease in value, but they can be reset if the user so desires. Counters by themselves are not particularly useful, but if the delta of two counters is looked at, they start becoming more interesting.

By looking at the rate of change the monitoring software can visualize the number of logins. This can help users see the peak times of their sites or malicious force entries. Another example of counters is system uptime. The usability of uptime varies greatly depending on what the monitoring system is set out to achieve.

2.1.3 Timers

Timers measure how long it took for the action to happen. If the monitored environment has databases, it is advised to place a timer for example to the backup script to see how long it took. It can be used to troubleshoot if the database backup script ran correctly. In PRTG most of the default sensors have a execution time sensor. Most of the timers that are placed by default are not useful by themselves, since most sensors don't process data in such magnitudes where the execution time would be problematic.

2.2 Simple network management protocol

SNMP was created in the 1980s, and it was selected by the Internet Activities Board (IAB) to be the short-term goal of Transmission Control Protocol (TCP)/Internet Protocol (IP) monitoring protocol in 1988 [4]. However, it was selected to be the de facto standard a year later, and in 1990 it was accepted as an internet standard. SNMP was the perfect fit for the growing needs of monitoring as it was not dependant on the operating system. At this point, it was widely accepted among manufacturers and support for it was increasing. Since the core of SNMP is standardized and the development has to just bridge the gap of SNMP and the device which needs to be monitored, it was easy for organizations and businesses to adapt SNMP for their own systems [5]. Still, like with all monitoring there needs to be humans that react to the growing amount of systems. This changed how businesses would approach system administration. Rather than having one person doing all the heavy lifting, there would be staff to maintain the manager. Alongside this there were people who were handling the actual devices and making sure that they were functioning properly and transmitting data to the manager. On top of these two roles, there was a new role added whose sole purpose was to monitor the systems. It was called Network Operations Center (NOC) and it is still to this day heavily utilized. NOCs are usually sourced workforce that monitor the systems 24/7 and their purpose is to fix

and alert the specialists of problems.

SNMP consists of layers: 1) manager and agent, 2) managed devices, and 3) Structure of Management Information (SMI) [6]. SMI is the top level concept where Management Information Base (MIB) is located [6]. Thus, in theory SMI does not do anything by itself; it is simply the host of what makes SNMP work. The agent provides the information of the corresponding MIB database to the application and accepts information from the manager. In April 1999 the Internet Engineering Task Force (IETF) established a new version of SMI, called SMIv2. SMIv2 functioned similarly, but expanded the list of base types to include integers, counters, IP address types, for example [7]. These types are defined and mapped to match the existing Abstract Syntax Notation One (ASN.1) protocol to avoid any discontinuity. MIB defines the collectable information that can be used by the application. Object Identifier (OID), is the node in the hierarchical tree where the MIB components are located. This hierarchical tree utilizes ASN.1 [8–10]. Without the help of OID, it would be cumbersome to try to discuss different stages of the hierarchy. In Figure 3 below, there is a name, which is the official name of the nodes. Next to this name, there is a number. This number is used for dot notation, which is to make the navigation easier. The people who work with SNMP regularly, start to remember the numbers like IP-addresses which makes the discussions and troubleshooting easier. For example, the MIB module of the Windows operating system's free disk space in percentage is as follows:

ASN.1 version of the hierarchy

> {iso(1) identified-organization(3) dod(6) internet(1) private(4) enterprise(1) Microsoft(311) software(1) systems(1) os(3) windowsNT(1) workstation(1) ldisklogicalDiskTable(5) ldisklogicalDiskEntry(1) ldiskPercentFreeSpace(3)}

The same hierarchy as above, but using the dot notation.

> 1.3.6.1.4.1.311.1.1.3.1.1.5.1.3

Figure 3 below demonstrates the hierarchy graphically.

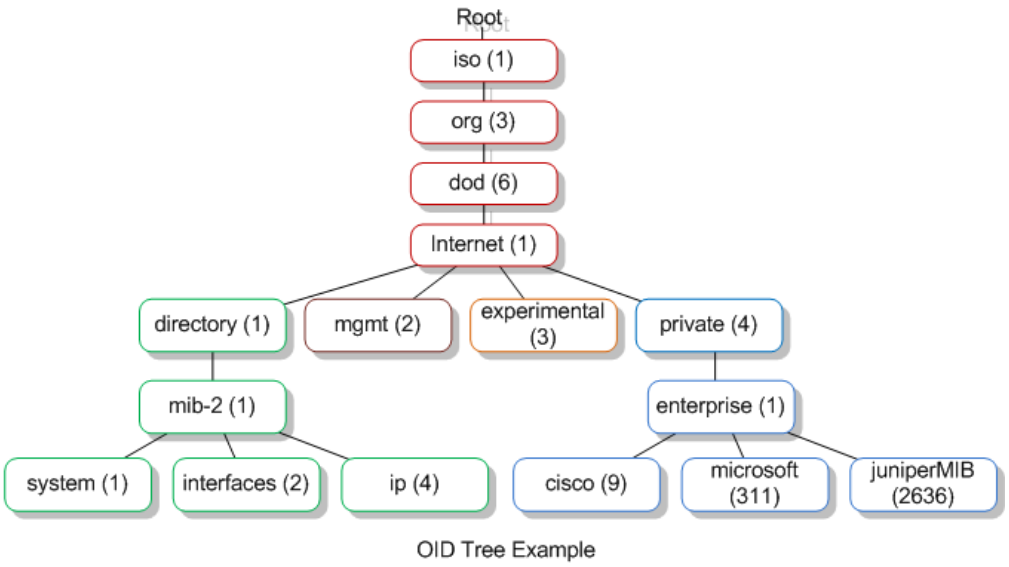

Figure 3: Hierarchy of the OID structure (SMI) [11]

In the end, when we have navigated to the end we find the corresponding MIB module. MIB is a database in which the data types are located. In this example we find the following data [12].

> ldiskPercentFreeSpace OBJECT-TYPE
> SYNTAX INTEGER
> ACCESS read-only
> STATUS mandatory
> DESCRIPTION
> “Percent Free Space is the ratio of the free space available on the logical disk unit to the total usable space provided by the selected logical disk drive”

2.2.1 TCP/IP protocol

SNMP uses User Datagram Protocol (UDP) for transportation of data between the agent and the manager. By using UDP, SNMP places minimal load on the network ensuring that if the network is failing SNMP does not cause further problems. At the same time, the strengths of UDP are its weaknesses. It being a connectionless protocol there is no reliable way to tell if the packets have reached their target. The confirmation and retransmitting is based on the SNMP application. This approach usually works well since the application will timeout notifying the agent to respond. One exception is the SNMP

trap, which sends a message but does not expect one back. In these cases it is impossible to tell if the trap has gone through. SNMP uses UDP ports 161 and 162, the former for queries and the latter for receiving traps [5].

2.2.2 Access policies

Access polices in SNMP dictate if a device's object variables can be changed in the MIB through a SetRequest call [13]. The objects that belong to the corresponding device are called the MIB view. One of the variables in the MIB view are called access modes, which represent the privileges. Possible values for access modes can be seen in table 1 with their function. These access mode values together with the MIB view form a community profile. Access policy is the combined effort from the community string and the community profile [13].

Table 1: Different access mode values in SNMP.

Access modes	Function
Read-only	Requests can read the object.
Read-Write	Requests can read or modify the objects data.
Write-only	Requests can only modify the objects data. This was phased out with SMIv2.
Non-accessible	Objects values cannot be read or modified.

The read-only access mode permits the manager to use the GET and TRAP commands to fetch the devices' objects that are found in the MIB. Read-write expands the scope by allowing the set commands to modify these objects. For example, if a routers physical location has changed. Administrators can set a new location to the device using the SNMPSET command. This thesis includes write-only, even though it has been deprecated from the current and future versions of SMI. It is included in the text because some of the older MIB views still have it in them. That being said, the write-only value is not used often. This is due to how SNMP operates. For example, having write-only access prevents the manager from reading the machine's status when GET is initiated. This potentially leads to a 'NoSuchObject' error. Even on the SET command write-only access has its troubles. The manager cannot verify that the machine or object exists; it will exit with an error. The implementation of the write-only access mode did not meet the expectations so the

IETF deprecated it in favor of read-write. The last access mode is non-accessible, which prevents the manager from reading or modifying the values.

SNMP comes with its own set of commands, like GET and SET. Using these commands, SNMP is able to gather information about the devices. Table 2 below shows the available commands and functions in SNMP.

Table 2: Different commands and their functions in SNMP.

Command	Function
GET	Returns one or more values to the manager.
GET NEXT	Returns the next OID's value based on the MIB hierarchy.
GET BULK	Returns all of the values in specified MIB hierarchy. Only available in SNMPv2 and SNMPv3.
SET	The manager can modify or set value to the device.
TRAP	The agent initiates the call and sends a value to the manager independently.
INFORM	Used for manager-to-manager communication. Only available in SNMPv2 and SNMPv3.
REPORT	Allows SNMP engines to communicate with each other. Only available in SNMPv2 and SNMPv3 although the SNMPv2 version was never implemented.

The GET command is the most basic form of SNMP communication. The manager initiates the communication and the agent in the system is trying to piece the request together. In order for the agent to understand what it is being asked, the manager needs to supply a variable binding with the request [5]. Variable binding is in essence the OID value of the object the user wants to output. The GET command can only retrieve a single object at a time. This makes it highly inefficient in larger scale operations, such as in monitoring. In order to make the information gathering less of a hassle users can employ the GET-NEXT operation. GET-NEXT can be used to retrieve a group of values from an MIB. The operation is virtually identical to GET regarding how the sequence works. It makes requests and receives get-response calls back for every object. Where this differs from GET is that, GET-NEXT goes through the SMI tree in alphabetical order and prints the objects. It stops either when the MIB or there are no more objects to print [5]. GET-NEXT is also known as SNMPWalk in the command line tools. In SNMPv2 a new form of information gathering was introduced, called GET-BULK. It can be used to retrieve a large section of a table at once. needs specifications in the command, which allows it to tell the agent to send as much information as it can. Usually the limiting factor is the agent, which throttles the

amount of data it can send to prevent problems. Using GET-BULK the user can expect to receive incomplete responses, which do not stop the query. This is in stark contrast to GET-NEXT, which stops the query if such an occasion arises.

The SET command, at its core, is very similar to GET. Main difference is that when using SET, the user is changing something in the managed device. Using SET, the user can modify multiple objects at the same time as opposed to the GET command's limit to one object. This operation can only be used when the object has been defined as read-write or write-only in the MIB, as it modifies the system's data [5]. The SET operation poses a security threat to the system and possibly the network. If the community string is weak or left at the default value, attackers have a vector to exploit when combining the SET and GET commands in read-write environment.

Trap is different from other commands since it is asynchronous, meaning it can be independently sent to the manager. Traps can be daunting at first, but since they are also defined by an MIB anyone can go and view them for clarification. They are also the only traffic that is just one-way and that paired with the fact that SNMP uses UDP, this means that there is no way of telling if the trap has reached its destination since there is no confirmation. This does not mean that traps are not efficient, quite the opposite, since the sequence does not have to wait for the initiation of the manager. When the agent notices that something is wrong with the device, it sends a message to the manager instantly.

SNMPv1 was missing manager-to-manager communication in case of multiple managers in the network. Version 2 introduced the INFORM command which does exactly that. This command requires an acknowledgement from the receiving manager. It can also be used to send trap messages to the manager. Using INFORM will notify the agent when the manager receives the message [5].

2.2.3 Versions of SNMP

There are a total of three different versions of SNMP. The latter two have been building on top of the original implementation. Every change goes through a committee and is written in a formal report. These reports are called Request for Comments (RFC). These committees include, but are not limited to, IETF and IAB. They are responsible for over-

seeing the future of SNMP, and they have made significant efforts to make sure SNMP is still a relevant tool in network monitoring. SNMPv1 was the first standardized version of SNMP and the later revisions are based on this. The biggest problem of version 1 and 2 was the fact that there was no encrypted authentication. Community strings, which are SNMP's version of a password, are used like typical passwords would be used in common machines, to establish trust between the systems [14]. It is worth noting that the community key does not replace passwords. Community keys are transmitted in plain text, so this means that if someone were to intercept the packets they would see the community key being passed [5]. During the lifetime of SNMPv1, it was decided that MIB-I is to be replaced with MIB-II in 1991, which was a welcomed change to the bare bones MIB-I version [15]. MIB-II introduced expandable subtrees and the ability to make an experimental MIB database which could be moved to the mgmt subtree later. The biggest change was the private subtree which introduced the ability to make system specific MIB's and store them.

2.2.3.1 SNMPv3

This chapter will discuss and highlight the key differences between the older versions and delve deeper into the specifics of SNMPv3. SNMPv3 was introduced in 1999 and it is widely considered as the security patch SNMP needed, adding cryptography and hashed passwords [16]. It had no effect on the protocols, although the naming conventions and terminology changed [5]. The agents and managers were changed to be the SNMP entity. The engine inside of SNMP consists of pieces [17]. These pieces are pictured in Figure 4.

The applications contact the dispatcher. The dispatcher's job is to send and receive messages. It requires information from the second piece, called Message Processing Subsystem. It prepares the messages to be sent. It also handles the data extraction from the received messages. Message Processing Subsystem can contain many versions of SNMP as shown in Figure 4. It tries to determine the correct version based on the data of the messages. The third piece of the engine is Security Subsystem and it is responsible for the authentication. In earlier versions the authentication used was a community string, but this was upgraded in SNMPv3 to be user-based. It can be configured to use Message-digest algorithm (MD5) or Secure Hash Algorithm (SHA) to authenticate users [5]. The

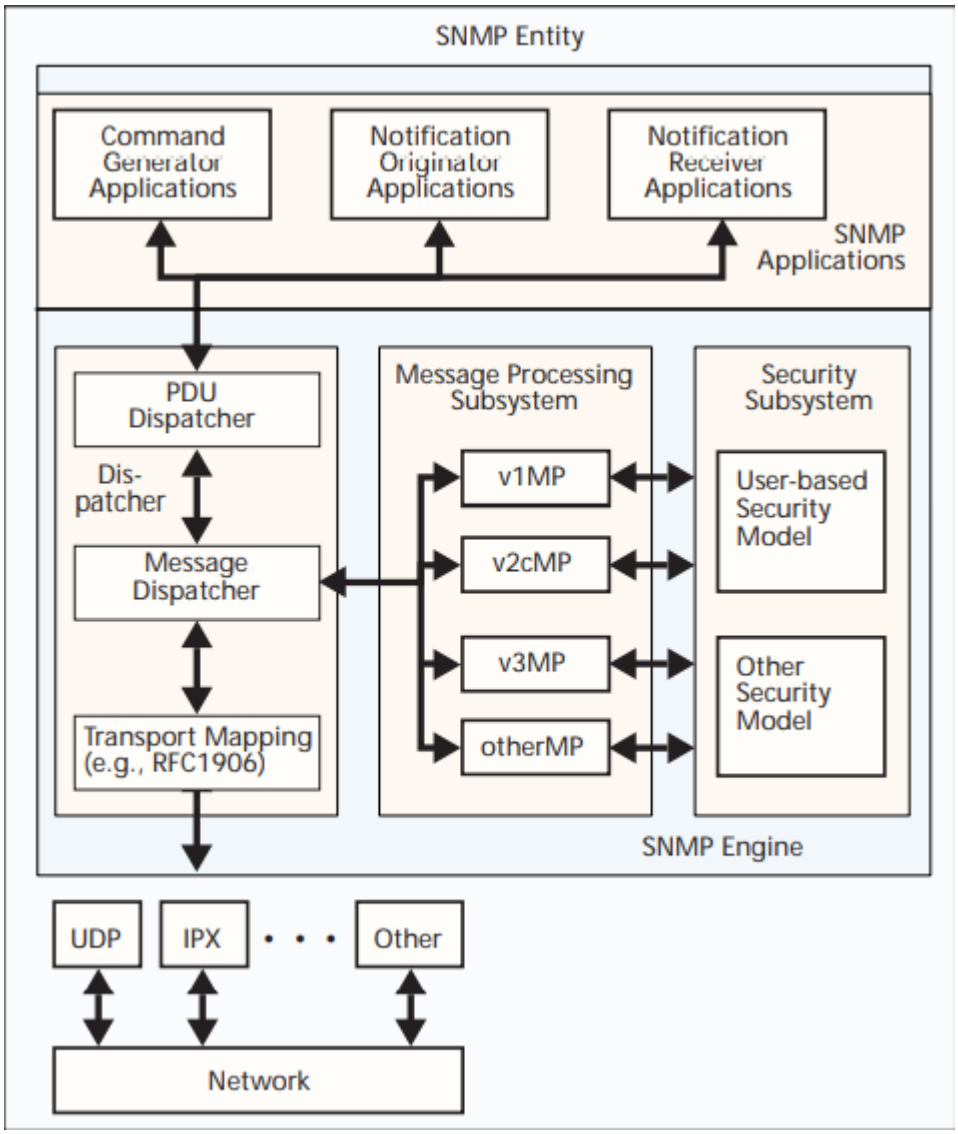

Figure 4: SNMP manager [17]

last piece of the engine is Access Control Subsystem which is responsible for controlling access to MIB objects. The access is done based on the contents of the Protocol Data Unit (PDU).

In SNMPv3 the aforementioned commands, for example GET and SET, are actually part of a message called PDU. The PDU specifies what the message does when it is handled by the pieces of the SNMP engine. Figure 5 shows the message protocol. The table 3 shows the functionality based on Figure 6.

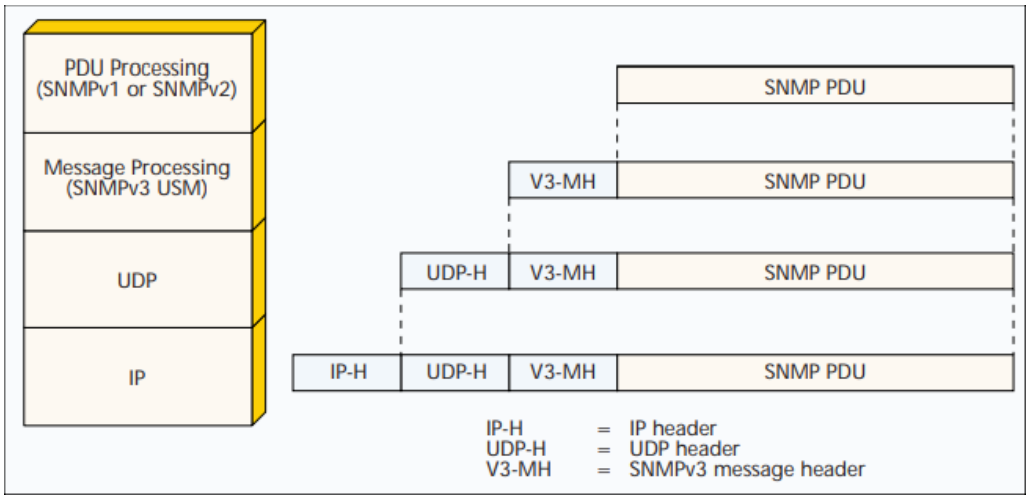

Figure 5: SNMP message protocol [17]

The figure 6 and the table 3 show the message protocol contents in a more detailed way.

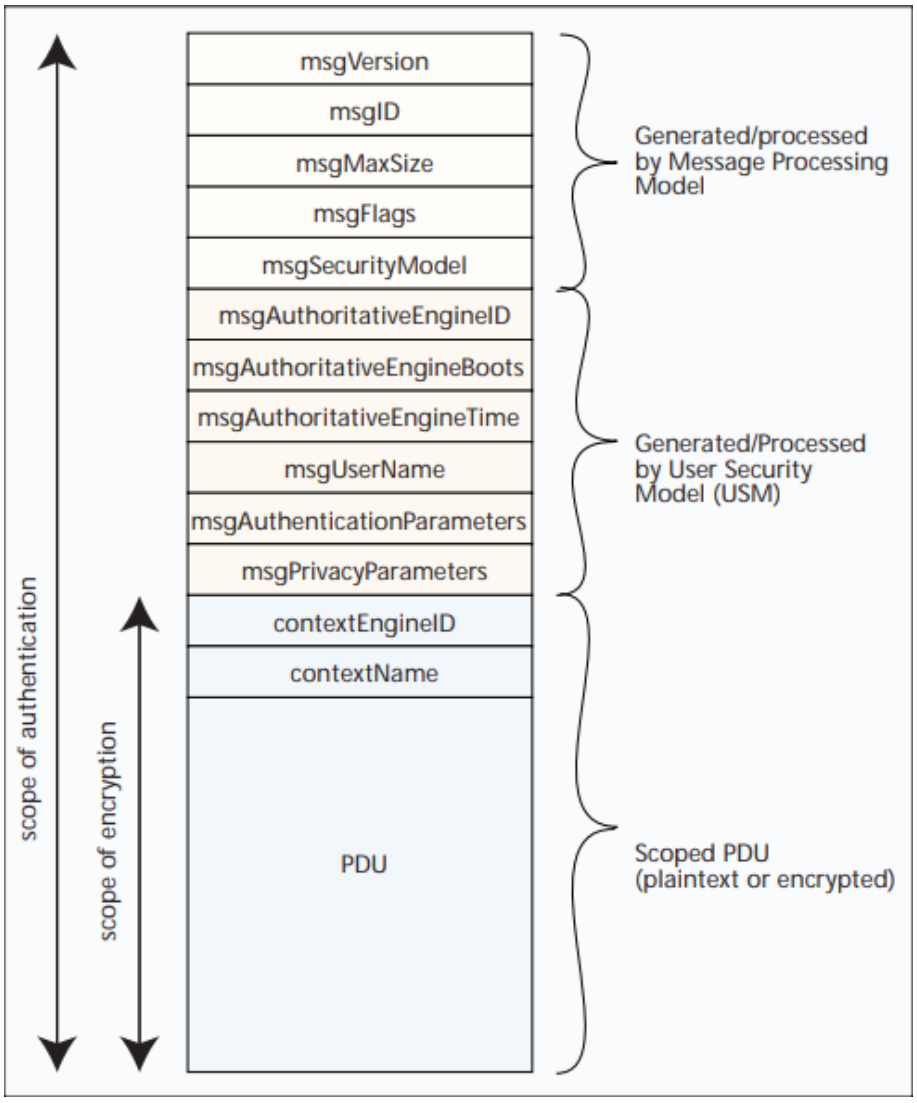

Figure 6: SNMPv3 message format [17]

Table 3: The content of the SNMPv3 message with their respective functions.

Field	Function
msgID	Unique identifier that is used between two SNMP entities, so the message does not get mixed.
msgMaxSize	Dictates the maximum size of the message in octets.
msgFlags	Failover in case the PDU contents cannot be read and the receiving end needs confirmation as to what the message contains. Possible values in octets are reportableFlag, privFlag and authFlag.
msgSecurityModel	Specifies what security model the sender used, so the receiver can correctly process the message.
msgAuthoritativeEngineID	Unique ID of the SNMP engine involved in the message.
msgAuthoritativeEngineBoots	Logs the amount of initializations the SNMP engine has done.
msgAuthoritativeEngineTime	Shows how long the SNMP engine has been on based on the last initialization.
msgUserName	The name of the user who owns the message.
msgAuthenticationParameters	Signals if the exchange of this message contains authentication. This value can be null.
msgPrivacyParameters	Signals if the exchange of this message contains privacy algorithms. This value can be null.
contextEngineID	Unique ID for the SNMP entity.
contextName	Identifies context in an engine in conjunction with contextEngineID. This is moves to the dispatcher and access control subsystem as a parameter.
PDU	Indicates the type of management action, for example GET.

3 Comparing monitoring software

Since monitoring is a crucial part of business's capability to ensure continuous services, it is no surprise that there are more implementations for monitoring than PRTG. This chapter will introduce the most relevant alternatives to PRTG and compare them.

Most of the core monitoring concepts have been implemented in a similar manner in these alternatives. However, this chapter will highlight the most stark differences, for example pricing, plugin or extensibility aspect. Because Empower Oy had already purchased PRTG license before the thesis work began, all of the information is based on the supplier's documentation and information on the internet.

3.1 PRTG

The tool that was decided to use for monitoring was PRTG made by Paessler. PRTG stands for Paessler Router Traffic Grapher. PRTG offers capability to extend the servers over many probes. The previous implementation was also made using PRTG, the only difference being that it was hosted and administrated by the data center provider. The goal of the new system was that it would not be reliant on external suppliers and controlled in-house. PRTG is mainly an agentless network monitoring solution although it also supports agent based if the user needs to monitor networks outside of PRTG installation.

The licensing in PRTG is based on a number of sensors. The contract is minimum one year subscription, with possibility for longer terms upon request. The pricing ranges from EUR 1,300 to EUR 12,500. Before committing to a contract it is possible to use a 30 day trial of PRTG, which allows the user to add an unlimited amount of sensors to the installation. After that Paessler offers custom contracts to suit the customer's needs. The difference between the basic packages lies in the amount of sensors the user can deploy in their PRTG instance. For example, in PRTG an XL1 package that costs EUR 12,500 at the time of writing, supports unlimited sensors and one core installation. While PRTG 1000, like the name suggests supports, 1000 sensors over a one core installation. It is

possible to get more core installations, but these require the custom offer packages. For a medium sized company such as Empower IM Oy, the PRTG XL1 package offers enough flexibility since probes are not limited by the licenses. Paessler also offers what is called maintenance in their basic packages. This essentially provides an update guarantee and a email support. They offer 12 months free, but after that it has a quite steep pricing on it, especially considering that without this there is no way to get updates to the instance. If the maintenance subscription has ended and has been restored later on, companies can charge retroactively.

The initial phase of the project was on a time crunch. As previously mentioned, the former monitoring system was part of the old data center provider's plan. When the project was initialized in October 2019, there was only a couple of weeks to get the systems transitioned from the old monitoring software to the new one. This meant that the job had to be automatised mostly. These automation scripts were crude in nature, but worked in the time frame. Figure 7 below shows the logic that the monitoring was built upon. Business lines have been categorized into their respective sections. Quality assurance and development machines have also been categorized by business lines to avoid confusion as shown in Figure 7. In the categories, systems have been sectioned further into customers. These customers sometimes overlap, but the point was to keep customers in their own groups in case of maintenance work. When all of the customer's systems are in the same place hierarchically, the systems can be paused and the administrator can be sure that the correct systems are now paused. Network monitoring is done similarly except, instead of customers, the devices are categorized using technologies. Unless there are multiple connections going to the same customer, then it makes sense to put these devices into their own group. For example, all of the firewalls are in their own category. Inside this section, the firewalls have been further categorized into sites.

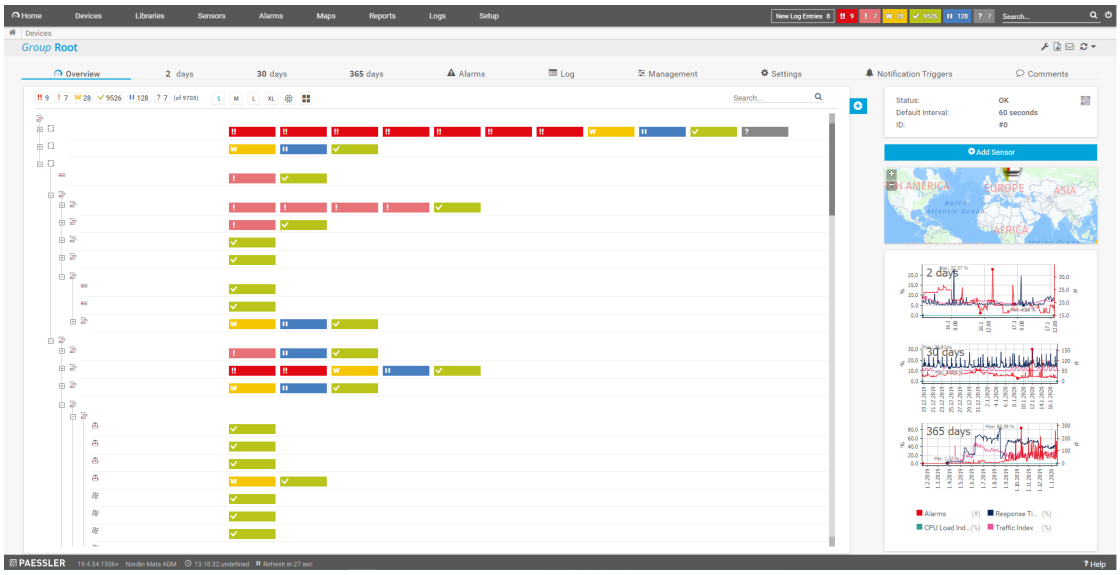

Figure 7: Devices tab after the groups had been populated. (Names retracted)

3.1.1 Probes

Figure 8 shows an example of the topology and how the whole network monitoring works on top of probes. When the core server is initialized, PRTG automatically makes a local probe if run on premises and a hosted probe if hosted by Paessler [18]. It is also possible to run monitoring straight from the core, but this can cause problems in performance.

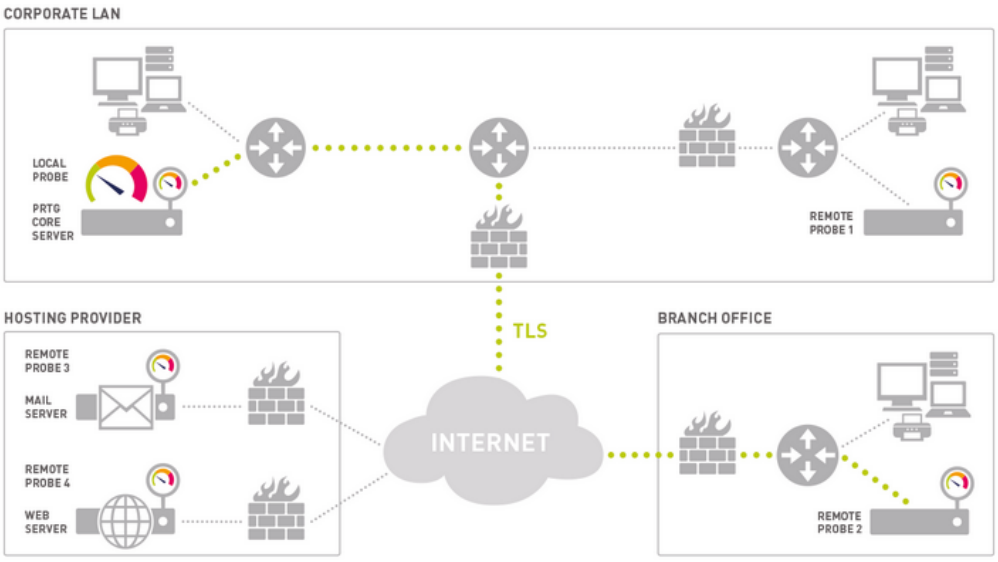

Figure 8: Official example of remote probe scenario by Paessler

Remote probes can be used to balance the load from the sensors, as well as allow monitoring over the internet. In this particular case, remote probes were used mainly for load balancing. Load balancing becomes increasingly important when more sensors are de-

ployed. At the moment, one probe is used for monitoring. In the early stages it was discussed whether business logic sensors should be on a separate probe. However, it was decided that one probe was enough, because the load the sensors were creating was not creating any issues in performance. Paessler provides guidelines, for example on WMI sensors. According to their tests, it is advised to stay under 200 high-load WMI sensors in a single probe. They have also rated the sensor's impact on the servers. Figure 9 shows the impact of various sensors in PRTG. SNMP sensors generally have less of an impact to the probe when compared to the WMI counterpart. Since the data is processed in the server itself, it is advised to keep the amount of high impact sensors as low as possible.

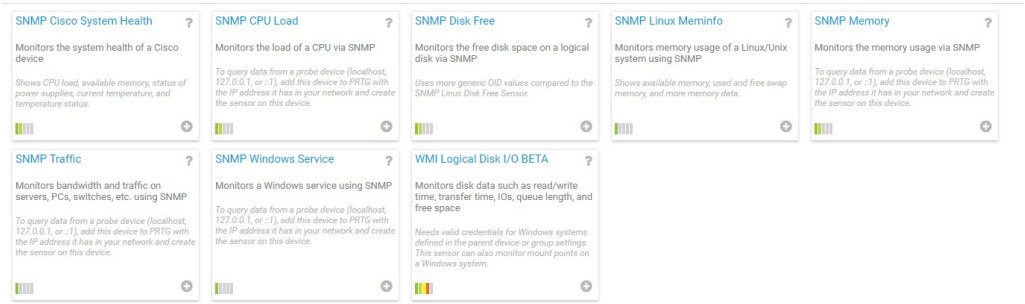

Figure 9: List of sensors with a different rating system

The probe is running alone in a virtual machine, so that it does not put any unnecessary overhead to the monitoring. It is located in local network and is responsible for relaying the data to the core, which is outside of said network. Using multiple probes, especially in business logic monitoring would be beneficial, but it would raise the cost of the monitoring. The business logic probe would ensure that these sensors would be running even if the less critical probe went down. There would even be a case for mirroring – having two probes mirror each other, but this was deemed unnecessary at this point. Remote probes are separate entities from the core server, which means that if the core server went down, the probe would continue to monitor its own sensors. When the core server is back online, the probe sends its results back [18]. Ensuring continuous data is critical for logging purposes. Paessler recommends using SNMP sensors as much as possible, due to them being lower impact in general. Using SNMP over WMI, one can monitor 10 times as many nodes [18]. Installing SNMP and starting to use it is going to be discussed in detail in later chapters.

PRTG has made it easy to link the probe to the core, by providing administrator tools

which make connecting to the core server is made simple as shown in Figure 10. The user can point to the core, by just giving either the IPv4 address or Domain Name System (DNS) name. From the tool, the probe can be configured to have a scheduled maintenance, either only the services or the whole system. The probe can sometimes go into a lock down or the DNS can stop working. In these cases restarting the probe fixes the issues. Restarting the probe does not restart the whole PRTG core instance, and vice versa restarting the core instance does not affect the probe.

The probe Global ID (GID), shown in Figure 10, is a crucial part of the probe's identity. In case the probe needs to be migrated from a core to another, the GID is the unique identifier and should be saved for the remainder of the migration. The access key is a pre-shared key between the core and the probe to establish a connection when creating an environment. These keys need to match in order for the probe to be able to connect to the core.

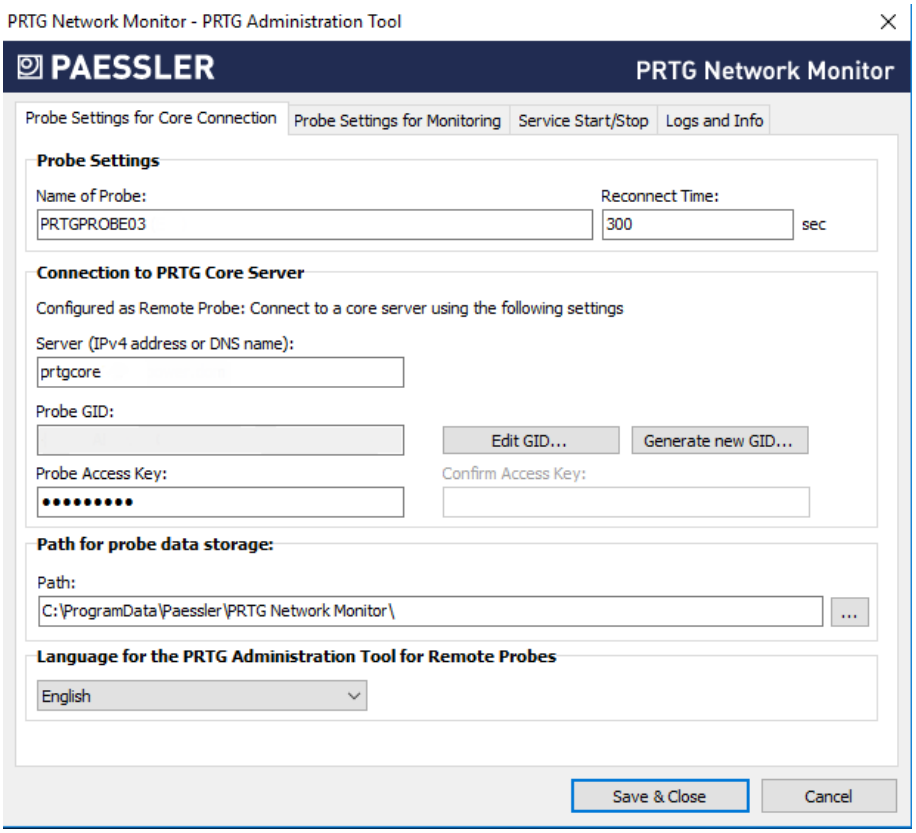

Figure 10: PRTG Probe admin tool

Updating the probe is done at the same time as updating the core, even though they are two separate instances. Updates are initialized in the core server, but once it has been updated it is passed down to the probes as soon as they re-establish the connection to the core. This means that for a brief time the probe stops monitoring while it reboots itself, so maintenance windows have to be planned accordingly.

3.2 Nagios XI

Nagios XI is an enterprise version made from Nagios Core. Nagios Core is open source software made by Ethan Galstad that dates back to 1996, but the name Nagios was established in 2002 due to a copyright issue.

Nagios works similarly to PRTG as it can be configured to work agentless or with agents. Where Nagios differs from PRTG is the monitoring principles. Nagios requires plugins for monitoring. These plugins are essentially executables or scripts in various programming languages that can be run to check the status of a host or service. The core service then uses the data from these plugins to do the real monitoring. Visualization of the workflow is shown in a Figure 11. This workflow gives Nagios absolute flexibility and it can be used to monitor anything the users need. There is a drawback to this though. Nagios is only the vessel for the data. It has no idea what the data does. In addition, in the case of more complex business monitoring it can become cumbersome to create specific scripts to each unique system. It is also worth noting that Nagios XI can be installed to a Linux system. It can monitor various systems, like Windows and switches, but the core needs to be on a Linux system. The features that are missing in Nagios when compared to PRTG are for example a built-in ticketing system and Active Directory monitoring. The latter can be detrimental if the company uses several Windows environments. Pricing on Nagios XI is from $1,995USD to $3,500USD. The latter one includes maintenance and support and has an annual renewal price.

3.3 Zabbix

Another competing implementation is Zabbix, which is developed by Zabbix LLC and which is also open source like Nagios Core. Zabbix, like the two other software, can work

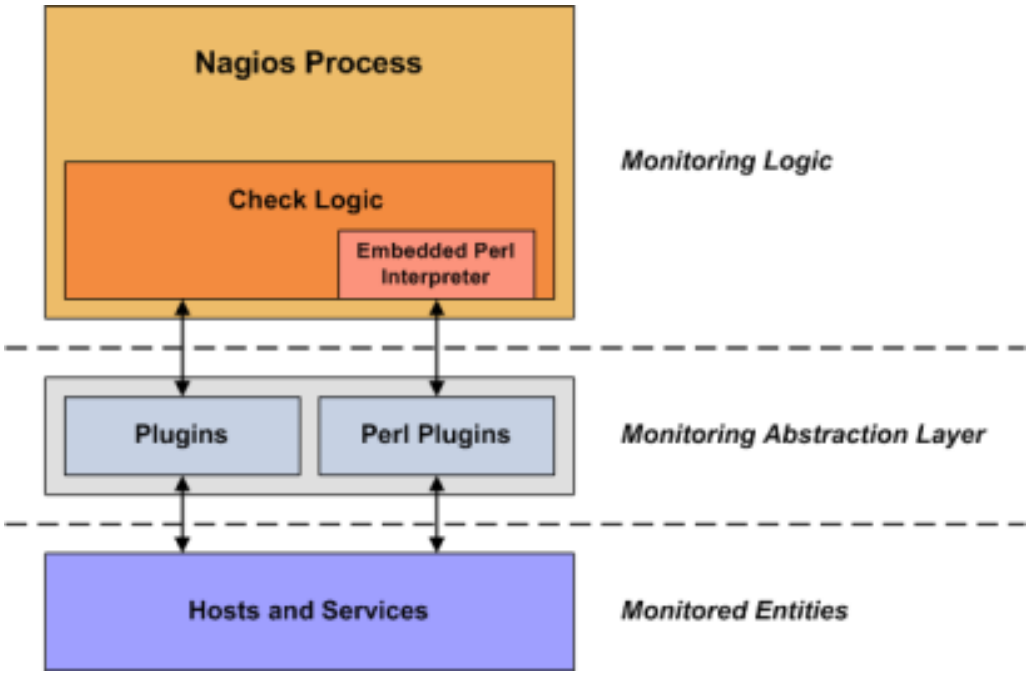

Figure 11: Nagios plugin infrastructure

agentless and agent-based. The architecture resembles the likes of PRTG, when comparing Figure 12 and Figure 8. The UI Zabbix is implemented using PHP, as opposed to PRTG's AJAX/REST. Zabbix supports network discovery like PRTG, which makes adding new systems easier. Notifications in Zabbix are similar to PRTG, as Zabbix can send automatic notifications in browser and via SMS, email or IM applications. Pricing in Zabbix goes from $18,700 to $43,900, according to the Red Hat pricing model.

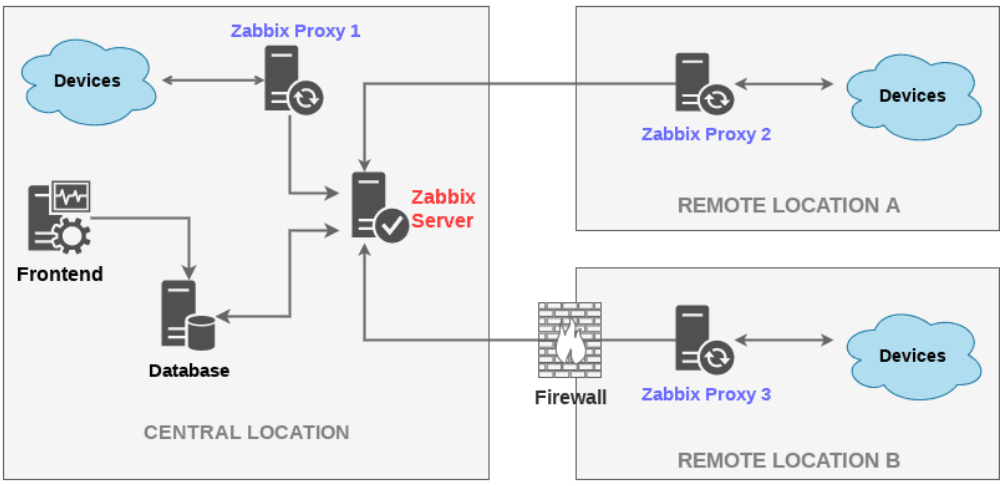

Figure 12: Zabbix architecture

Overall, all of these software achieve the same result. The choice boils down to which features are being focused on by the companies. If the company has developed many software of its own and knowledge of building monitoring on those, Nagios XI is a prime candidate for them. Reversely, if the company has many network devices and needs something fast to be able to monitor them, PRTG or Zabbix is a good choice. Unfortunately, the reality is usually that when comparing the software the pricing is usually the most concerning part of the system.

4 Implementing PRTG

This chapter outlines the steps it took to install the necessary parts of SNMP to servers and networking devices as well as how sensor populating was done in PRTG. After the default sensors have been installed, an advanced monitoring method was used with Application Programming Interface (API) manipulation.

4.1 Installing SNMP

On both operating systems, SNMP needed to be activated manually. In Windows this consists of modifying and adding register keys to accommodate the community key. Since Windows does not support SNMPv3 officially, the choice of SNMPv2 was made easy. If we were to install an unofficial SNMP client like Net-SNMP, we could have had SNMPv3

in Windows machines. This adds an extra layer to the system administrator's workload if there was an update. The update would not be done automatically and a test would be needed to make sure it works as intended. The windows feature is great as it updates alongside the normal server maintenance. The security considerations are also negligible since the monitored servers are not accessible from the internet and are segmented in the local area network. Initially Linux machines were distributed with SNMPv2, due to time constraints and to streamline the deployment of PRTG. Since then, efforts have been made to migrate from v2 to v3 in Linux. Newer development machines are being tested with v3.

4.1.1 Windows

It was wiser to make a Powershell script that could be run from the domain controller to the rest of machines in that domain.

The first thing the script in Listing 1 does is gathering all Windows machine names in the domain into a Comma-Separated Values (CSV) file and declares the $servers function. Subsequently, the script continues down the list using a for-loop and check connectivity. If connection is established the script runs remote commands to install SNMP services and modify registry keys using a community key and the IP-address of the probe.

This script provided to be the fastest approach to do this. It didn't come without problems. There was a problem regarding the older servers. The script used a 64-bit shell and since some of the servers were either 32-bit or they simply lacked the new Cmdlets, they had to be done manually. One possibility aside from adding SNMP by hand would have been to use Deployment Image Servicing and Management (DISM). It was decided not to use this approach since the amount of legacy servers was rather low and there was a time window in which we needed to get the monitoring to work.

```
1    Get-ADComputer -filter * | Select-Object -Property Name | Export-Csv
       'c:\temp\server.txt'
2   $servers = Get-Content -Path c:\temp\server.txt
3   ForEach ($server in $servers) {
4      Write-Host $server
5      if ( Test-Connection $server -Count 1 -TimeToLive 2 -Quiet ) {
6       invoke-command -computername $server { Install-WindowsFeature
       SNMP-Service -IncludeAllSubFeature -Verbose }
7       invoke-command -computername $server { Install-WindowsFeature
       RSAT-SNMP }
8       invoke-command -computername $server { New-ItemProperty -Path "
       HKLM:\SYSTEM\CurrentControlSet\Services\SNMP\Parameters\
       PermittedManagers" -Name "2" -Value "PROBE IP ADDRESS" }
9       invoke-command -computername $server { New-ItemProperty -Path "
       HKLM:\SYSTEM\CurrentControlSet\Services\SNMP\Parameters\
       ValidCommunities" -Name "COMMUNITY KEY" -PropertyType DWord -Value
       4 }
10     }
11  }
```

Listing 1: Powershell script used to install SNMP on Windows

4.1.2 Linux

On Linux the approach can also be made into a bash script as seen in Listing 2. Since the Linuxes are running Red Hat Enterprise Linux (RHEL) as the main distribution, the installing of SNMP was done using the package manager. Due to the nature of these commands, we had to disable command line history while modifying the files. Modifications were done using Stream Editor (SED), since a couple of lines needed to be replaced in the snmpd.conf configuration file. After the modifications, the service was ready to be run.

The configuration happens in the SNMP configuration file, which is definitely a better approach compared to Windows registry keys. After the net-snmp packages were installed successfully, the history was disabled and SED was used to modify the lines required. The first part is the original line and the latter part is the replacement text. Most of the lines look identical, since the configuration is really just adding user and community keys to the file. After these modifications it is important to enable history again.

```
1       yum -y install net-snmp net-snmp-utils
2       set +o history
3       sed -i -e "s/com2sec notConfigUser default public/com2sec empsnmp
            default COMMUNITY KEY/g" /etc/snmp/snmpd.conf
4       sed -i -e "s/group notConfigGroup v1 notConfigUser/#group
            notConfigGroup v1 notConfigUser/g" /etc/snmp/snmpd.conf
5       sed -i -e "s/group notConfigGroup v2c notConfigUser/group empsnmpgrp
            v2c empsnmp/g" /etc/snmp/snmpd.conf
6       sed -i -e "s/view systemview included .1.3.6.1.2.1.1/view systemview
            included .1./g" /etc/snmp/snmpd.conf
7       sed -i -e "s/view systemview included .1.3.6.1.2.1.25.1.1/#view
            systemview included .1.3.6.1.2.1.25.1.1/g" /etc/snmp/snmpd.conf
8       sed -i -e "s/access notConfigGroup \"\" any noauth exact systemview
            none none/access empsnmpgrp \"\" any noauth exact systemview none
            none/g" /etc/snmp/snmpd.conf
9       set -o history
```

Listing 2: Bash script used to install SNMP on RHEL

4.1.3 Networking

For network monitoring SNMP has to be configured on each device separately. Figure 13 is a part of the Fortigate configuration panel that shows the SNMP configuration. Empower Oy has multiple firewalls which are configured in a similar way. Trap calls are not being used at the moment, which should be a consideration when improving the monitoring.

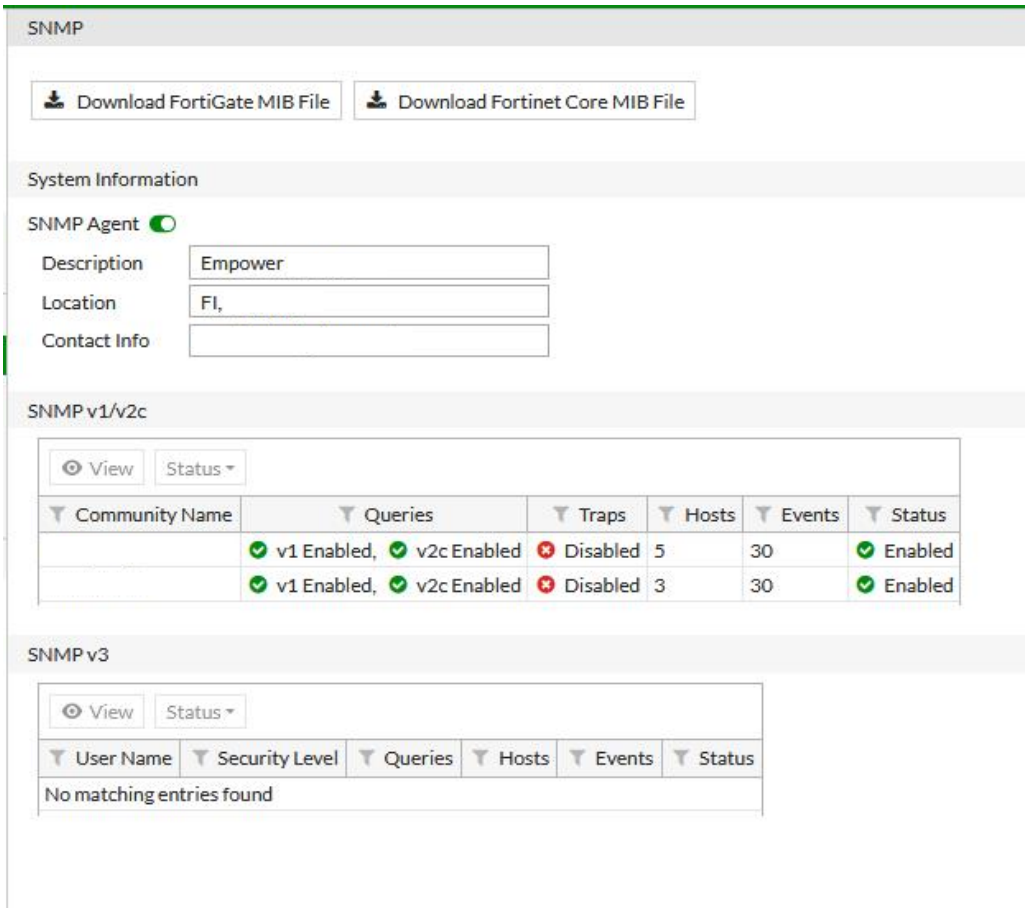

Figure 13: The Fortigate configuration tool showing SNMP

Listing 3 shows a relevant part of Cisco's ASA VPN device configuration. Since it is Cisco's product, the syntax is very similiar to the one shown in Listing 4. Snmp-server at lines 4 and 5 represents the PRTG's core and probe servers. Contrary to the configuration of the switch, in ASA VPN, the topological location has to be specified in the command. In this case, both the servers are in LAN, which requires inside. As with the firewall configuration, ASA VPN has to be configured to use SNMPv2 due to limitations in PRTG. In this specific ASA VPN, the specialist has chose not the disclose the location and the contact information. As opposed to the firewall configuration, traps are enabled in ASA VPN devices. They are not actively being used. This should not be the case since they could provide valuable monitoring data.

```
...
...
...
snmp-server host inside xxxxxxxxxxxx community ***** version 2c udp-port
    161
snmp-server host inside xxxxxxxxxxxx community ***** version 2c udp-port
    161
no snmp-server location
no snmp-server contact
snmp-server community *****
snmp-server enable traps syslog
snmp-server enable traps ipsec start stop
snmp-server enable traps ikev2 start stop
sysopt connection preserve-vpn-flows
service resetoutside
...
...
...
```

Listing 3: ASA VPN configuration

Listing 4 is a running configuration of a switch showing vlan configuration. In this vlan, the PRTG's core and probe servers are added to the switche's access-list as permits. Below the access-lists are the community keys with the read/write privileges for PRTG. The community keys are inputted in PRTG's settings at root level. RO stands for read-only and RW stands for read-write. Snmp-server location represents the physical location of the switch.

```
...
...
...
interface Vlan13
 ip address 10.xxx.xxx.xxx 255.255.255.xxx
 no ip route-cache
!
ip default-gateway 10.xxx.xxx.xxx
ip http server
ip http secure-server
kron occurrence scconftomgnet at 9:00 recurring
 policy-list conftomgnet
!
kron policy-list conftomgnet
 cli show run | redirect tftp://10.xxx.xxx.s01
 cli show vlan brief | redirect tftp://10.xxx.xxx.s01_vlans
!
kron policy-list conftoprtg
!
logging host 10.xxx.xxx.xxx
access-list 13 permit 10.xxx.xxx.xxx
access-list 13 permit 10.xxx.xxx.xxx
access-list 13 permit 10.xxx.xxx.xxx
snmp-server community xxxxxxxx RO
snmp-server community xxxxxxxx RW 13
snmp-server location xxxxxxxx
!
!
line con 0
 password 7 xxxxxxxxxxx
 login
line vty 0 4
 exec-timeout 60 0
 password 7 xxxxxxxxxxx
 login
 length 0
line vty 5 15
 login
!
ntp server 10.xxx.xxx.xxx
end

xxxx-xxxx-xxxx-s01#
```

Listing 4: Part of a switch configuration showing SNMP

4.2 Populating PRTG with sensors

Most of the sensors were built into PRTG and they started to work nicely after the devices were configured with their corresponding IPv4/DNS addresses. PRTG offers capability for an auto-discovery group which scans the network segment. After it has reached the devices it starts to create sensors using mostly SNMP and WMI. Although this would make the tree structure a lot faster, we opted not to use this. Since the default auto-discovery finds all of the devices in a network segment, it would create devices and sensors even to the devices that are not crucial. There are definitely use cases for the automation which will be deatl with later. For now, we are going to discuss the auto-discovery templates, because these offer the most flexibility granted the servers have been added to the monitoring system. PRTG offers plenty of ready-made templates, but after having a look at them, it was apparent that these templates would be too extensive. Luckily, they also offer a tool to make a personal template by excluding sensors from a device's ready sensor tree. With this tool we would be able to build a template for Windows and Linux systems separately and use the sensors we were set out to use in the first place. These were not perfect but having to remove seven sensors, most of which were tied to traffic, was easier than having to delete 30 sensors per device.

Auto-discovery groups, like seen in Figure 14, are perfect for when there is a network segment specifically made for a customer. In the configuration users can for example specify the range of IPv4 addresses where PRTG creates a device to every reachable system in the range. This gives some control over the discovery groups, but the depth of the sensors is still large. To change this depth the user has to change the auto-discovery level, which by default is set to standard auto-discovery. This populates the devices with PRTG's own set of sensors. Detailed auto-discovery is often unnecessary as it populates the devices with all default sensors and some custom sensors which are built into PRTG [18]. As a result, there are sensors that are neither working nor redundant. Like in the earlier use case, there is also the option to use device templates. These templates are the same templates as discussed before; only this time the discovery is done automatically from the selection method. This is the go-to way to populate groups with the same kind of machines as it is fast and gives the most control.

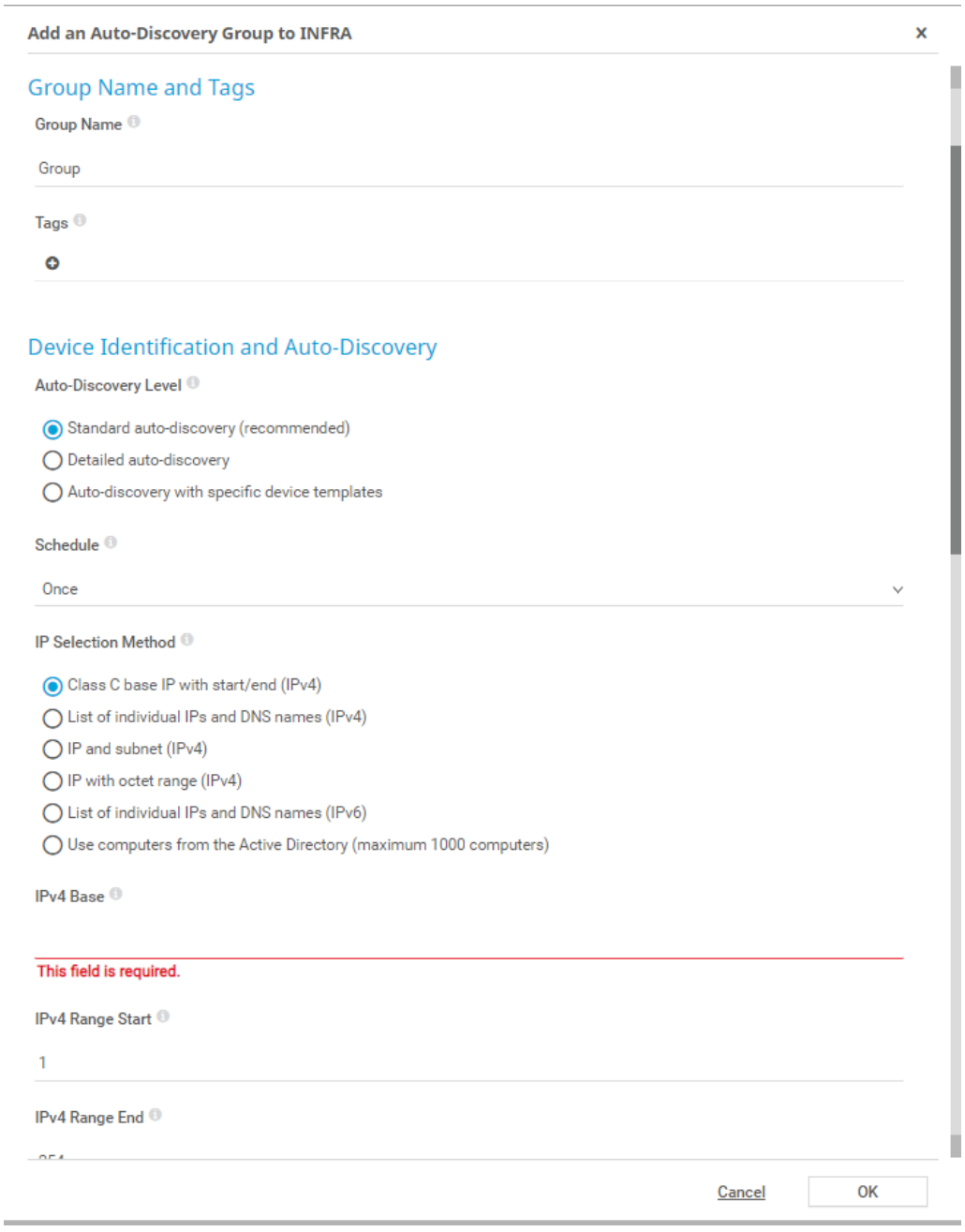

Figure 14: Auto-discovery group tool

4.2.1 Advanced methods

Creating meaningful implementations, where alert fatigue can be avoided and make them extensible by anyone. Creating documentation on the source code and giving access to other people to modify the sensors. For these advanced methods of implementing monitoring, PRTG didn't either have a solution or it was not applicable in our use case.

4.2.1.1 Load balancing between three routers

One of the customers has fail-over routers which are crucial to monitor. The issue with them is, that only one of them is responding continuously. A lazy approach would be to have all of them on constantly; however this is not advisable due to redundant warnings in the system, and as was established in the theory chapter, alert fatigue is to be avoided. The backup routers are online for sometime for testing purposes, even when the primary router is online. This was causing the backups to reset their state and to alert when they went down again. Leaving the backup routers without alerts was not an option, in case one of them was the primary router in use.

There was a proposal to have the other two routers, which are not responding, paused when there is one router responding. PRTG didn't have a ready-made solution for this. Therefore, we made use of PRTG's built-in API calls [18]. The approach would be having the connected router to pause the two other routers for set amount of time. The polling time is five minutes, after which the pause ends. In addition, in the next 60 seconds if there is one router connected, the two other are paused again for five minutes. This approach has one downside to it — the device log is practically unusable, due to all of the pauses that are happening every five minutes. Although PRTG does not offer a ready-made solution to this, it offers a Hypertext Transfer Protocol (HTTP) API which can be used to call the elements of the website. Below is a example of the pausing in action.

```
/api/pauseobjectfor.htm?id=123456\&pausemsg=Primary router in use\
&duration=5\&username=retracted\&passhash=retracted
```

This Uniform Resource Locator (URL) is then passed onto the notification trigger area in PRTG. Notification triggers allow actions to be performed to the device. Usually these actions are either alerting and/or sending messages about the status of the sensor. It also allows a way to run the API calls via state triggers. In this case when the sensor state is up for at least 300 seconds, an API call should be performed, which runs the aforementioned URL. To make the refreshing work, there needs to be a second state trigger with the same parameters that run every five minutes. This approach proved to be functional, but quite crude and leaves room for improvement. Due to company policy, passwords need to be changed every periodically. This essentially means that the passhash also changes every time the password changes and it needs to be updated.

5 Conclusions

The goal of the project was to design and distribute a new solution as a part of the daily routines for internal teams. The solution was to create a centralized view of the available assets. It had to be easily manageable and permissions hassle free, while still showing important information at a glance. When the project was presented as a possible subject by Empower IM Oy, it sounded like the bulk of the project would be quickly over and the scope would be a little shallow. The monitoring software the project was built upon was already decided before the project had started as part of a bigger project so there was no need for testing different monitoring solutions beforehand. However, it became apparent that designing the monitoring system was more complex task than perceived. The monitoring software was fitted into a moderately sized company so the amount of devices required a meticulous approach to make sure there would be little to no unwanted alarms going off. It is imperative that the users can trust the alarms, since the monitoring was running on different machines with databases and services with dependencies that are supposed to be online at all times.

The specifications with which the hierarchy and the alarms were made before starting the project, made the transition to a new monitoring system effortless. However, there were some problems that arose during the development and implementation of the monitoring system. Development issues regarded specifically the monitoring of the logical disk I/O. For some servers it was crucial that logical disk I/O would be monitored. In Windows machines, this was only possible using WMI. There is an entry in Windows MIB, but the agent was missing from the systems due to Microsoft wanting to drive their WMI forward in comparison to SNMP. In order to make the sensor work, one would need to write their own agent. Agents are coded most commonly in the C language and they are responsible for using the data that is stored in the MIB. There is, even to this day, still documentation and templates to make custom agents. Although this would have been valuable learning experience, while still being tedious, it was opted not to approach this any further due to time constraints.

WMI-based logical disk I/O sensors are advanced in terms of technology and provide as

much metrics as any specialist would want. That comes with its drawbacks though. They are more heavy weight in comparison to SNMP implementation. PRTG advises one probe to have about 200 of WMI based sensors, which means one has to be mindful of where to use them. This compromise was ready to be made, since it was apparent from the start that in the time frame of the migration it would not be feasible to program the agent from scratch using SNMP.

Another problem that arose during the implementation of the new monitoring system was the script used to install SNMP into the Windows machines. Some of the machines were in a demilitarized zone which meant that there was no connection from the Domain Controller (DC) to the machines. Another problem was that some of the machines had an older version of Windows which meant that the Cmdlet used was not available at that time. To solve this, SNMP had to be installed manually to the machines.

Because of the simple nature of the monitoring hierarchy it was easy to add more devices to the logical tree monitoring system. This is an advantage for possible future changes, as new devices can be effortlessly added to the monitoring system. In the previous monitoring system, the device's sensors were split into multiple places and the management was slow and often there were mistakes. The business line sensors were split into their separate folders away from the device's other sensors, which made troubleshooting the machines harder if the business line sensors were not working. After some deliberation it became clear that it is easier to put all of the sensors in the same section so expanding would be easier. This also provided logical groups for all of the products Empower IM Oy offers. One possible way to still separate business line sensors from the bulk is to use PRTG's library feature. It was out of the scope of study in this project, but could be studied in the future. With the library function the business line sensors could be divided into their own groups without it affecting the hierarchy and there is even a possibility to set library specific notifications.

To conclude, this thesis reports a successful development and implementation process of the PRTG monitoring system for Empower IM Oy. The new system is light-weight and easily maintained to ensure longevity of successful service maintenance of Empower IM Oy. However, there is still room for improving the monitoring strategy by taking advantage of SNMP's trap commands, which will provide even faster reaction time to alarms, and

ensure further customer satisfaction in Empower IM Oy's services.

Bibliography

1 Ponemon L. Cost of data center outages; 2016.

2 Turnbull J. The art of monitoring. Turnbull Press; 2014.

3 Rose MT, McCloghrie K. Structure and identification of management information for TCP/IP-based internets. RFC Editor; 1990. RFC1155. Available from: http://www.rfc-editor.org/rfc/rfc1155.txt.

4 Cerf VG. IAB recommendations for the development of Internet network management standards. RFC Editor; 1988. RFC1052. Available from: http://www.rfc-editor.org/rfc/rfc1052.txt.

5 Mauro DR, Schmidt KJ. Essential SNMP. Sebastopol, CA: O'Reilly; 2005.

6 Harrington D, Presuhn R, Wijnen B. An architecture for describing Simple Network Management Protocol (SNMP) management frameworks. RFC Editor; 2002. RFC3411. Available from: http://www.rfc-editor.org/rfc/rfc3411.txt.

7 Presuhn R. Version 2 of the protocol operations for the Simple Network Management Protocol (SNMP). RFC Editor; 2002. RFC3416. Available from: http://www.rfc-editor.org/rfc/rfc3416.txt.

8 Case J, Fedor M, Schoffstall M, Davin J. Simple Network Management Protocol. RFC Editor; 1988. RFC1067. Available from: http://www.rfc-editor.org/rfc/rfc1067.txt.

9 Case J, McCloghrie K, Rose M, Waldbusser S. Coexistence between Version 1 and Version 2 of the Internet-standard Network Management Framework. RFC Editor; 1996. RFC1908. Available from: http://www.rfc-editor.org/rfc/rfc1908.txt.

10 Rose MT. Management Information Base for network management of TCP/IP-based internets: MIB-II. RFC Editor; 1990. RFC1158. Available from: http://www.rfc-editor.org/rfc/rfc1158.txt.

11 Pritoni M, Nordman B, Piette M. Accessing Wi-Fi Data for Occupancy Sensing; 2017.

12 OID-Info. OID-Info, editor. ldiskPercentFreeSpace. OID-Info; 2018. http://oid-info.com/get/1.3.6.1.4.1.311.1.1.3.1.1.5.1.3, Last accessed on 22 January 2020.

13 Case JD, Fedor M, Schoffstall ML, Davin JR. Simple Network Management Protocol (SNMP). RFC Editor; 1990. RFC1157. Available from: http://www.rfc-editor.org/rfc/rfc1157.txt.

14 Case JD, McCloghrie K, McCloghrie K, Rose MT, Waldbusser S. Introduction to community-based SNMPv2. RFC Editor; 1996. RFC1901. Available from: http://www.rfc-editor.org/rfc/rfc1901.txt.

15 McCloghrie K, Rose MT. Management Information Base for network management of TCP/IP-based internets:MIB-II. RFC Editor; 1991. RFC1213. Available from: http://www.rfc-editor.org/rfc/rfc1213.txt.

16 Case J, Mundy R, Partain D, Stewart B. Introduction to Version 3 of the internet-standard network management framework. RFC Editor; 1999. RFC2570. Available from: http://www.rfc-editor.org/rfc/rfc2570.txt.

17 Stallings W. SNMPv3: A security enhancement for SNMP. IEEE Communications Surveys. 1998;1(1):2–17.

18 Paessler. Paessler, editor. PRTG Network Monitor User Manual. Paessler; 2020. Online; accessed 17 September 2020. https://manuals.paessler.com/prtgmanual.pdf.