TALLINN UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology
IT College

Joosep-Kristjan Välk 175164IDAR

BUILDING NETWORK MANAGEMENT AND MONITORING SOLUTION FOR AN ENTERPRISE USING FREEWARE COMPONENTS

Diploma thesis

Supervisor: Truls Tuxen Ringkjob
Master of Science

Tallinn 2018

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
IT Kolledž

Joosep-Kristjan Välk 175164IDAR

ETTEVÕTTE ANDMESIDEVÕRGU HALDUSE JA MONITOORINGU LAHENDUSE LOOMINE VABAVARALISTE VAHENDITEGA

Diplomitöö

Juhendaja: Truls Tuxen Ringkjob
Magistrikraad

Tallinn 2018

Author’s declaration of originality

Author: Joosep-Kristjan Välk

10.01.2018

Abstract

This thesis work takes on and completes the task of building a custom network management and monitoring system for an enterprise. The problems of automation, minimizing footprint, scalability, adequate error handling and reliability receive attention. As defined by the enterprise, the project will use free software and tailor the solution to match a previously compiled list of requirements and expectations.

As a result of the work, a new network management and monitoring system is built so that the current commercial and outdated network management software can be decommissioned. The new system covers the functions of event monitoring, tendency graphing, automated backup, traffic analysis, remote logging, channel visualization and aggregated dashboard.

The author concludes that building an all-round network monitoring and management solution by utilizing freeware components is feasible and practical, as compared to other common approaches – using commercial software packages or programming the whole solution in-house.

This thesis is written in English and is 46 pages long, including 6 chapters and 12 figures.

Annotatsioon

Ettevõtte andmesidevõrgu halduse ja monitooringu lahenduse loomine vabavaraliste vahenditega

Käesoleva lõputöö raames luuakse ettevõttele andmesidevõrgu halduse ja monitooringu lahendus, kasutades vabavaralisi vahendeid. Töö algab peamiste võrguhalduse alamülesannete ja ettevõtte poolt seatud piirangute ja eeltingimuste analüüsimisega, jätkub uue lahenduse implementeerimisega ning jõuab selle juurutamisel tasemeni, mis võimaldab likvideerida seni kasutusel olnud kommertsiaalse võrguhaldustarkvara. Töö tulemusel vabanetakse ka mitmete senise töökorralduse juurde kuulunud korduvate toimingute käsitsi tegemisest.

Töös käsitletakse probleeme automatiseerimise, ressursisäästlikkuse, skaleeruvuse, dubleerituse ja veahalduse ümber. Vabavaralised, süsteemi ressurssidega säästlikult ümber käivad võrguhaldus- ja monitooringu vahendid ei oma sageli kogu funktsionaalsust, mistõttu tuleb programmeerida täiendavaid lahendusi nagu konfiguratsiooni automaatne genereerimine ja seadmete ning teenuste automaattuvastus.

Töö tulemusel valmis distributeeritud võrguhalduse ja monitooringu lahendus, mis katab intsidentide monitooringu, tendentside graafimise, automaatse varundamise, võrguliikluse analüüsi, keskse logimise, kanalite koormuse visualiseerimise ja agregeeritud monitooringu vaate kuvamise funktsioonid.

Lõputöö tulemusel järeldatakse, et vajalikku funktsionaalsust pakkuva võrguhalduse ja monitooringu lahenduse loomine vabavaraliste vahenditega on praktiline ja saavutatav eesmärk. Selline lähenemine konkureerib arvestatavalt alternatiividega, milleks on kommertsiaalsete tarkvarapakettide kasutamine või kogu lahenduse programmeerimine oma ressurssidega.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 46 leheküljel, 6 peatükki ja 12 joonist.

List of abbreviations and terms

API	Application Programming Interface
ARP	Address Resolution Protocol
CGI	Common Gateway Interface
CI	Configuration Item
CLI	Command Line Interface
CMDB	Configuration Management Database
CPU	Central Processing Unit
DoS	Denial of Service
GSM	Global System for Mobile Communications
HP	Hewlett-Packard
HTML	Hypertext Markup Language
HTTPS	Hyper Text Transfer Protocol Secure
HUP	Hangup
IOS	Internetwork Operating System
IPDB	IP Database
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
IRB	Integrated Routing and Bridging
ISP	Internet Service Provider
ITIL	Information Technology Infrastructure Library
JSON	JavaScript Object Notation
LDAP	Lightweight Directory Access Protocol
MAC	Media Access Control
MNTOS	Multi Nagios Tactical Overview System
NOC	Network Operations Center
OID	Object Identifier
OS	Operating System
PHP	Personal Home Page

PNG	Portable Network Graphics
PPDIOO	Prepare, Plan, Design, Implement, Operate, Optimize
RRD	Round Robin Database
SMS	Short Message Service
SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol
SNMPv2	Simple Network Management Protocol version 2
SNMPv3	Simple Network Management Protocol version 3
SSH	Secure Shell
SVI	Switch Virtual Interface
SVN	Subversion
UDP	User Datagram Protocol
URL	Uniform Resource Locator
VLAN	Virtual Local Area Network
VPN	Virtual Private Network
WAN	Wide Area Network
XML	Extended Markup Language

Table of Contents

1 Introduction....................11
2 The task of complex network management solution....................15
2.1 Monitoring....................15
2.1.1 Event monitoring....................15
2.1.2 Tendency graphing....................16
2.1.3 Redundancy and distribution considerations....................16
2.1.4 Aggregated dashboard....................17
2.1.5 Out of band notifications....................17
2.2 Configuration management....................18
2.2.1 Automated backup....................18
2.2.2 Supervision and change visibility....................19
2.3 Channel visualization....................19
2.4 Traffic analysis....................20
2.5 Multi-tier logging....................21
2.6 Baseline data collection....................21
2.7 Various utilities....................21
2.8 Central CMDB and automation....................22
3 Overview of components....................23
3.1 Nagios for event monitoring....................23
3.2 Cricket for tendency graphing....................24
3.3 MNTOS for aggregated dashboard....................24
3.4 Opengear ACM for out of band notifications....................25
3.5 Rancid for automated backup....................25
3.6 Rancid as supervision and change visibility tool....................26
3.7 Rancid as baseline data collection tool....................26
3.8 PHP Weathermap for channel visualization....................27
3.9 NfSen with NFDUMP for NetFlow traffic analysis....................27
3.10 Rsyslog for remote logging....................28

4 Implementing the solution....29
4.1 Base installation....29
4.2 CMDB and configuration generation....30
4.3 Autodiscovery....32
4.4 Setting up the components....34
4.4.1 Nagios....34
4.4.2 Cricket....35
4.4.3 Rancid....38
4.4.4 PHP Weathermap....39
4.4.5 MNTOS....40
4.4.6 NFSen....41
4.4.7 Programming custom utilities....42
5 Results and conclusions....43
6 Summary....45
References....46

List of Figures

Figure 1. A network weathermap. Source: https://traffic.lan.switch.ch..........................20
Figure 2. Example records of managed devices in JSON format...................................30
Figure 3. Help screen and usage examples of the hostlist utility....................................31
Figure 4. Excerpt of Nagios configuration written by the autodiscovery script.............33
Figure 5. Excerpt of Cricket configuration written by the autodiscovery script............33
Figure 6. Nagios overview page..35
Figure 7. Multiple interface parameters on a single graph produced with Cricket.........37
Figure 8. Text summary generated by Cricket to explain each graph............................37
Figure 9. WebSVN interface displaying a RANCID configuration repository...............38
Figure 10. Part of the enterprise network weathermap...39
Figure 11. MNTOS provides an overview of all Nagios instances................................40
Figure 12. NFSen web interface displaying a stacked logarithmic scale graph of all sampled UDP traffic..41

1 Introduction

Intriguingly, data communication networks (from hereon referred to simply as "network") are perhaps the most invisible, transparent and unseen component of the Internet – or any computer network – from the user's perspective. When a web page does not load or an e-mail client fails to complete an SMTP (Simple Mail Transfer Protocol) operation, the most common and immediate assumption is that the server or service is to blame, overlooking the network in between. Given this, it is even more so expected that when systems work flawlessly, the network between the endpoints goes unnoticed. Until something happens (and it will).

When it does, it is crucial to have a good network management toolkit at hand. Monitoring may be the obvious answer, but in fact, monitoring is only a fraction of the solution. When something happens, the network administrator needs, for example, device configuration backups. Sometimes an administrator debugs a fault and sees a suspicious yet inconclusive diagnostic output, leading to the question – "was it different before the issue started (and if, then how exactly)?" – hence the need for baseline data collection, logging and tendency graphing. Any network management tool is only of any good if it knows which targets it has to manage or monitor, leaving no gaps, never forgetting or overlooking a new addition, change or legitimate removal – hence the need for central CMDB (Configuration Management Database) and automatic discovery. The list of requirements and expectations to a full-coverage network management toolkit goes on and on [1, pp. 25-26].

This thesis takes on the task of implementing one such network management toolkit for an enterprise, using freeware components. It is meant to be applied work, not seeking to discover new theoretical aspects of network management or prove/disprove any statements. Rather it takes a look into the implementation of one complex IT system, using the skills and practices obtained through courses at IT College and internship. It capitalizes on the current trend of automation, integration and the blurring of boundaries between system administration and software development stemming thereof.

The network administration team at the enterprise has held meetings and discussions about the requirements and expectations for the new network management solution. The current situation has been mapped and documented and plans for the new system drawn up. Author of the thesis is tasked with building a network management system with the following features:

- Event monitoring
- Tendency graphing
- Aggregated dashboard
- Out of band notifications
- Automated backup
- Supervision and change visibility
- Channel visualization
- Traffic analysis
- Multi-tier logging
- Baseline data collection
- Various utilities
- Central CMDB and automation

The enterprise has set the following guidelines to the network management software choices:

- Reuse software already present in other parts of the IT infrastructure to minimize later operational complexity
- Use software with OS (Operating System) integration (available in repository)
- Use mature, tried and proven software with visible track record of community support

- Use the smallest-footprint software possible to complete any given task

Given this, the focus of the work is not to compare a wide variety of software components for each task. Instead, it focuses on the overall functionality, following the guidelines where possible. Functions performed by one software component/product here (e.g. Nagios for event monitoring) can be most probably achieved by an alternate product (e.g. Zabbix for event monitoring). The four expectations listed above do not pose strict restrictions but act as guidelines. If a suitable component satisfying the guidelines cannot be found, an exception will be made and documented.

The remaining part of the work is divided into 4 main chapters.

Chapter 2 – The task of complex network management solution – describes the tasks and functions that need to be covered and discusses their position in the overall network management problem.

Chapter 3 – Overview of components – discusses the components used, mentions some of the choice criteria and describes details and considerations to take into account before proceeding to implementation.

Chapter 4 – Implementing the solution – documents the implementation process and details, gives examples and describes the results of implementing each individual component.

Chapter 5 – Results and conclusions – looks at the overall result and discusses afterthoughts, lessons learned, deficiencies discovered and next steps to take.

Methods and approaches used in this project are loosely based on the Cisco PPDIOO (Prepare, Plan, Design, Implement, Operate, Optimize) methodology [2, pp. 12-13]. The Prepare phase has already been completed by the enterprise network team. Chapters two and three map to the Plan and Design phases of the methodology and chapter four corresponds to the Implement phase. Though the Operate and Optimize phases are not strictly in the scope of this work, chapter 5 and the summary touch on some operational aspects and future optimizations.

Approaching the automated network management in a systematic and comprehensive manner, beyond the capabilities and functions of any single product, is new for the enterprise. The final result is expected to bring along a principal qualitative change and innovation compared to the previously tried methods. It will, for the first time, introduce true linear scalability for the network management process.

2 The task of complex network management solution

This chapter looks at the overall task of network management, the individual functions that the enterprise needs to have covered and the expectations of added value. In this chapter, main functions of network management software are discussed and mapped out, before going into the specific products and solutions (which is done in chapter 3).

Most statements and conclusions in this chapter trace back to the body of common knowledge throughout the industry, built piece by piece through experience and practice over decades. Nevertheless, the path to filtering and accepting that vast and disorganized collection of best practices, *de facto* standards and typical caveats is different for each organization [1, p. 25].

The preliminary analysis done before this thesis work produced the following outline of the new network management toolkit.

2.1 Monitoring

In broad sense, network monitoring typically falls into two subcategories – event monitoring (real-time alerting, event handling) and tendency graphing (collecting time-series data for graphical rendering). While trying to fulfill these functions, infrastructure-, redundancy-, resiliency- and worst-case scenario-related issues need to be considered. This section discusses each of these aspects.

2.1.1 Event monitoring

The first and most obvious type of network monitoring the enterprise needs is fault detection with active notifications. Network devices usually offer comprehensive SNMP (Simple Network Management Protocol) interface, which can be polled for immediate state information and delta calculation [1, p. 27]. It was decided that event monitoring should not only be reactive, i.e. detect true error states, but also pro-actively detect events leading up to possible service interruptions. This requires rather detailed monitoring of values like memory usage, interface counters, link utilization, etc.

Given these considerations, the event monitoring part of the solution has to be scalable, lightweight (more about this in 2.1.3 in the context of redundancy and distribution considerations), customizable and generic. A strict requirement is the feature of differential monitoring, also known as delta calculation, because values like link utilization can only be observed by examining static counters over a known period of time.

2.1.2 Tendency graphing

In addition to event monitoring, the enterprise needs a tendency graphing solution to monitor, visualize and store the developments in network over long time periods. As with event monitoring, network devices offer static counters which need to be polled. From there, delta values can be calculated, results stored, graphs rendered and web interface provided for the operator. Good long-term graphs are expected to be a valuable tool in growth prediction and network gear scaling for future upgrades.

Tendency graphing solution must once again be lightweight, scalable and customizable, requirements descending from additional considerations given in separate sections below. The back-end database used for graphing purposes must be of open and common architecture. The data should be re-usable by other utilities like Weathermap (channel visualization tool) and manual examination/correction (e.g. spike removal).

2.1.3 Redundancy and distribution considerations

It has been accepted by the enterprise that monitoring and management systems, out-of-path systems by nature from the revenue service point of view, can be initially deployed without full redundancy properties. However, the initial design and component choices must accommodate for future high-availability requirements. Especially when implemented through active polling, monitoring is a function that can be carried out independently by multiple instances. There is no need for complex solutions like state-synchronized clusters and copying of monitoring state data from one server to another, because a number of monitoring servers can poll target systems individually and achieve redundancy that way. However, a lightweight active/passive status coordination still needs to exist for notifications. After introducing redundancy, only one monitoring instance can actively send out notifications.

Another concern that needs to be addressed is the globally distributed nature of the enterprise network. When WAN (Wide Area Network) connections, individual sites or global enterprise routing fail, a central monitoring system is not useful. On the other hand, a complex and fragile hierarchical system of monitoring agents, where monitoring data is sent between servers and aggregated, is not desirable. Once again, it was decided that distributed monitoring instances can work in parallel with the global instance. Remote sites will each have a monitoring server that will track local targets, providing visibility and postmortem data at times of global connectivity interruptions.

From the above-mentioned stems a core requirement for most of the software components used – the components need to be lightweight, resource-conserving, able to work on small remote virtual machines with 1GB or less of memory.

2.1.4 Aggregated dashboard

As described, the overall monitoring solution is realized using several distributed instances. The initial design uses multiple instances for redundancy and distribution, but future developments may also require more than one central monitoring server for performance scaling. Given this, the operator needs a central dashboard that would, at minimum, aggregate and visualize the overview of all event monitoring instances on one screen.

As the enterprise desires simple and lightweight components throughout the system, a simple stateless web harvester is a preferred solution. It is sufficient if the dashboard solution polls the web interfaces or APIs (Application Programming Interface) of the monitoring instances every minute and visualizes the summary.

2.1.5 Out of band notifications

Most of the monitoring software available defaults to delivering notifications (alerts) by e-mail. E-mail notifications are cheap, quick and there are no restrictive length and detail constraints. On the other hand, they may not get through if the network is down. Also, the operator needs quick and generic way of delivering notifications to their phone during off-hours. Due to these considerations, the enterprise requires SMS (Short Message Service) notifications in parallel with the e-mail solution.

There are several options to integrate SMS into a software solution. At one end of the spectrum lies the usage of an external SMTP or other API accessible over the IP (Internet Protocol) network (e.g Twilio[1]). That, however, does not solve the issue as it is not out-of-band from the network's perspective. At the other end of the spectrum lie the physical cellular modems that can be attached to a monitoring server. This option is problematic due to the virtualization layer in today's systems. An in-between solution would be a device that is available on the local network, accessible to the monitoring server over a local IP subnet, sending the messages independently. Because only a minimal network connectivity would be required, such option was considered acceptable.

2.2 Configuration management

This section explains which aspects of configuration management are required from the new solution. Configuration management itself is a broad term and an entire process, compared to which the context and scope here is very limited. The author was tasked with implementing a toolkit component that would automatically back up device configuration, track changes, keep a revision history (a repository back-end to the system) and provide automated notifications of configuration changes.

2.2.1 Automated backup

At the core of the system described above is regular, automated configuration backup. Successful implementation of such features also sets requirements to the devices and their configuration format. Devices need to provide a communication channel or an interface that can be used to automatically fetch the whole configuration (e.g. CLI (Command Line Interface) access over SSH (Secure Shell)).

Minimally, the new configuration management solution would need to regularly fetch device configurations and store them in a configuration repository. It should log its actions and generate notification, if configuration back-up attempt was unsuccessful.

[1] https://www.twilio.com

The configuration repository should have a web interface usable by the network operators.

2.2.2 Supervision and change visibility

Once automated configuration backup is implemented, supervision over configuration changes and their visibility can be easily implemented. This, however, sets some additional requirements to the devices and their configuration formats/interfaces. While a closed-format binary configuration file can be fetched, checked into a repository and the fact of its hash change acknowledged, individual changes can not be tracked and understood easily. Thus devices need to provide a well-structured, human-readable text configuration format.[1]

Once configuration files are checked into the repository, the system needs to detect configuration changes, generate differential reports and send notifications to the network team e-mail. This way every member of the team can keep up with the changes and mistakes can be detected through peer review.

2.3 Channel visualization

Channel visualization is a form of real-time network health/status overview that is most commonly used to display load on WAN links (often dubbed “weathermap” in the industry). As the enterprise uses VPN (Virtual Private Network) and WAN links between its remote locations, such links form potential bottlenecks and a weathermap solution is required for good operational overview. Figure 1 displays an example of a weathermap solution. In this case, channel loads are measured in megabits per second and approximate load levels are expressed in color codes.

As interface counters will be polled, deltas calculated and time-series data stored for tendency graphing, the weathermap solution should be able to re-use the same data, i.e. the weathermap should be regularly generated from existing data, without additional polling of the devices.

[1] In the future, this may be solved by the YANG data modelling language and NETCONF, described in (among others) IETF RFC 6020 (https://tools.ietf.org/html/rfc6020). Currently, the network devices used in the enterprise do not fully implement such a universal solution.

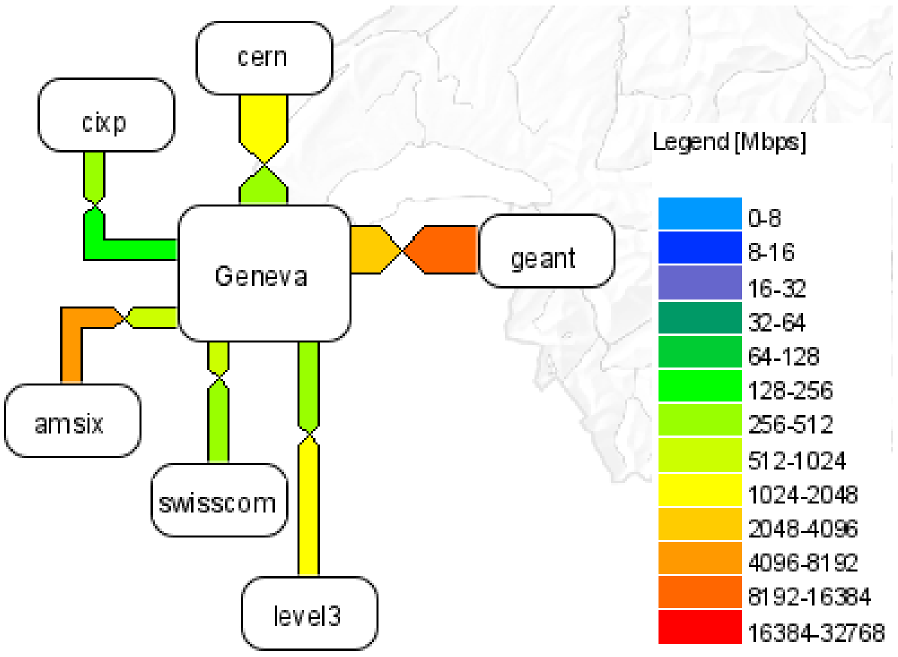


Figure 1. A network weathermap. Source: https://traffic.lan.switch.ch

2.4 Traffic analysis

The enterprise is using routers and firewalls with NetFlow support. The new network management solution should take advantage of this feature and include a NetFlow based traffic analysis software. Traffic analysis will enable the network operators to identify top talkers, discover and examine flood-type DoS (Denial of Service) attacks and plan for growth and upgrades.

Currently, the whole global routing and VPN tier of the enterprise network supports NetFlow. Because Juniper SRX branch series firewalls are widely used, NetFlow must be sampled and the analysis software must support NetFlow sampling and scaling[1]. The traffic analysis software has to provide a modern and convenient web interface for the network operators.

[1] Juniper Knowledge Base article KB16677 states that “Activation of flow collection can have a significant impact on the performance of the SRX Series device. The smaller the sample rate, the bigger the impact. It is recommended to not use a sampling input rate of 1.” (https://kb.juniper.net/InfoCenter/index?page=content&id=KB16677)

2.5 Multi-tier logging

The new network management toolkit must provide a multi-tier logging solution. All network devices are required to support Syslog and make use of both local and remote logging. Logging is arranged in three tiers: local logging to device internal storage, site-local logging to a log server on site and global (central) logging to a central log server. Lower tiers provide temporary log data at times of network interruptions.

Logging and log data storage is a critical process which needs to be protected by redundancy. The nature of Syslog makes it possible to avoid complex cluster, replication and forwarding solutions by configuring the devices themselves to log to multiple destinations. That is, all replication of log data will happen closest to the source and log servers will not have to coordinate or synchronize.

2.6 Baseline data collection

Some level of baseline information is collected from the network as byproduct of monitoring and configuration management. However, the new network management solution should provide additional means to collect custom baseline data. The operators should be able to define diagnostic commands that will run regularly on the managed devices. Output of such commands can then be saved to a repository and changes tracked. When an error condition occurs, the current state of the network can be compared to the previously established baseline and common deviations from the norm can be spotted quickly [3, p. 27].

A lightweight solution combined with backup and configuration management is highly preferred. As the automated backup solution will, in most general terms, log into the devices, issue commands and save their output, the aim is to customize it as much as needed to include necessary baseline collection functionality. If that does not produce satisfactory results, a separate baseline collection software needs to be considered.

2.7 Various utilities

Though full discussion of establishing network management access and security is beyond the scope of this document, it becomes obvious that the new network management solution will be positioned in full view of the network devices' management interfaces. Therefor it will be a proper place for hosting various custom and future utilities.

Some examples of custom utilities in the network management toolkit are a layer 2 map of the network, automated device software version tracking, configuration validation scripts, etc. Placed at the network management server, utilities and scripts like these can pull and link data from the network devices (e.g. MAC (Media Access Control) address table) as well as enterprise databases like IPDB (IP Database) and CMDB. This enables comprehensive, informative overviews for the IT infrastructure operators.

2.8 Central CMDB and automation

All of the requirements and desired features described in this chapter come together and will be viewed in the context of this final requirement. All of the different components must honor one single CMDB that contains the managed targets (network devices). The administrator will not manually create configuration files for different monitoring and management components. Moves, adds and changes in the network will only be registered in one single database. From this database, automation takes over and makes sure that devices are added and removed from the backup software targets, that devices, interfaces and links are added and removed from monitoring and so on [4, p. 45].

This goal sets requirements to most of the components used. As configuration needs to be machine-generated, every software component used must provide a clean and simple configuration format or API. To program and verify automated configuration generation, a format that is both machine-readable/writable and human-readable is highly preferred.

3 Overview of components

Chapter 2 outlined the discreet requirements for the network management solution without mentioning specific software products or components. This chapter introduces and explains the software components that will be used in the implementation phase. Most of the choices are derived from enterprise requirements discussed in the Introduction.

3.1 Nagios for event monitoring

Different event monitoring solutions have been repeatedly compared in previous thesis works of IT College. Nagios has been found to be lightweight, versatile and generic, which also suggests it should be highly customizable [5, p. 22]. These conclusions coincide with those of the author and the understanding in the enterprise. Therefore, Nagios was chosen as the event monitoring component of the network management solution. Also, Nagios has been used in the enterprise to monitor other parts of the IT infrastructure, so it leaves open the possibility to merge different monitoring systems in the future.

An important requirement for all the components was that their configuration could be machine-generated. Nagios uses well-structured text files for all of its configuration, so this requirement is easily satisfied. The configuration is both machine- and human readable/writable, so developing scripts for automated configuration generation is simple and time-conserving. Additionally, Nagios comes with a configuration validation/verification functionality, which is helpful for arranging fail-safe unattended configuration changes[1].

[1] https://www.unix.com/man-page/debian/8/nagios3

3.2 Cricket for tendency graphing

Cricket is a high performance, extremely flexible system for monitoring trends in time-series data. Cricket was expressly developed to help network managers visualize and understand the traffic on their networks, but it can be used for all kinds of other jobs as well[1]. Essentially, Cricket is a set of wrapper scripts around RRDtool[2] to provide the simplest and thinnest possible solution for tendency graphing of network parameters. Cricket exists in the enterprise for monitoring other parts of the IT infrastructure. Because no specific reason exists to introduce a new software package, Cricket was chosen as the tendency graphing component of the new network management solution.

Cricket uses RRD (Round Robin Database) files for its data storage. It is expected that the future weathermap component can re-use the data collected by Cricket for channel visualization. Cricket, like Nagios, provides a text file configuration interface, equally readable and writable by human and machine. This satisfies the requirement for simple and effective automation scripting.

3.3 MNTOS for aggregated dashboard

MNTOS (Multi Nagios Tactical Overview System) is a lightweight PHP (Personal Home Page) web application that aggregates multiple Nagios web interfaces into a single overview page. It is the simplest and smallest similar solution the author was able to find and successfully test. It satisfies the minimal requirement defined in section 2.1.4.

MNTOS is a stateless solution that harvests the Nagios web interfaces for information, stores it in XML (Extended Markup Language) format until next run only and provides a simple web interface to visualize the aggregated dashboard. Data collection from different Nagios instances can be initiated using cron. For sufficiently up to date dashboard view, a one minute or less polling interval is desired. It is easily achievable with MNTOS because, as later tests showed, polling six Nagios instances took only a few seconds.

1 http://cricket.sourceforge.net
2 https://oss.oetiker.ch/rrdtool

3.4 Opengear ACM for out of band notifications

For out of band notifications via SMS, the midrange option was chosen – the sending of SMS will be handled by a physical GSM (Global System for Mobile Communications) terminal., The terminal will be attached to an integrated device residing on the local network (relative to the monitoring server). As Opengear ACM5500 devices[1] were available for use, the monitoring component of the new network management solution will use these to send SMS notifications.

Opengear ACM5500 series are small devices running a customized version of the Linux kernel and a set of core utilities. The device is connected to the local network over an Ethernet interface and to the GSM network over an integrated modem. The operating system provides utilities to send SMS messages, as well as SSH interface to access the device itself. Sending an SMS from the monitoring server then becomes a matter of making SSH connection and issuing proper commands. This interface was deemed suitable for the project because the scripting needed to interface it to Nagios is simple and straightforward.

3.5 Rancid for automated backup

RANCID[2] is a traditional tool for automated configuration backup throughout the network management field. It started out as Cisco IOS (Internetwork Operating System) specific tool, but due to its modular and customizable nature, functionality to fetch configurations from HP (Hewlett-Packard), Juniper and other network devices has been added. RANCID is built around the concept of configuration repository that handles incremental changes and version history. RANCID proved to sufficiently satisfy the requirements of fault detection/notification and logging placed upon the backup solution. If a device is unreachable or the regular backup fails for other reasons, the administrator is notified by e-mail.

The configuration of RANCID comprises mainly of the targets list it has to work with. This configuration is expressed in simple text file format, so it satisfies the needs of automated configuration generation feature. Targets will not be added or removed

[1] http://opengear.com/download/sdt-connector/Opengear%20User%20Manual.pdf

[2] http://www.shrubbery.net/rancid/

manually, rather a central CMDB is used to keep the list of targets covered under backup up to date. RANCID supports the required SSH transport method and it also provides means to arrange public key authentication (as per enterprise requirements, public key authentication must be used to access the network devices over SSH).

3.6 Rancid as supervision and change visibility tool

RANCID also acts as supervision and change visibility tool, part of the change management process in the enterprise. After fetching the configuration from the target devices, RANCID checks the changes into a repository an calculates the difference between the current and previous versions. If configuration changes are detected, an e-mail notification is sent with the differential changes highlighted. This assures that changes can not happen in the network without the knowledge of the whole team and that the quality and compliance of changes can be verified.

As RANCID uses SVN (Subversion) for its repository back-end, the requirement for web interface can be satisfied. Independently of RANCID, WebSVN[1] will be set up for the operators to view and manage device configuration backups.

3.7 Rancid as baseline data collection tool

Though a separate baseline data collection tool or custom script is planned for future, RANCID will be initially used to collect the output of additional diagnostic and informative commands. Due to modular and open architecture, RANCID is easy to modify and amend for such purposes. Commands used to fetch data from devices are represented in a single list called “commandtable”. By amending the list, commands like “show mac address-table” and “show interfaces status” can be added for baseline data collection.

Because RANCID comes with differential change detection and e-mail notifications, changes in baseline data can be tracked and monitored similarly to configuration changes. That will act as complementary means to event monitoring through SNMP by Nagios. Some more complex states and conditions may not be accessible through the SNMP interface or their monitoring would be troublesome to realize using the core

[1] https://websvnphp.github.io

plug-ins available to Nagios. In such cases, RANCID's features discussed here will become useful.

3.8 PHP Weathermap for channel visualization

For the weathermap-style channel visualization, Network Weathermap[1, 2] was identified as the simplest and most lightweight component available. PHP Weathermap is a simple web application that re-uses data from RRD files, in this case, stored by Cricket. This satisfies the requirement that, if at all possible, the channel visualization software should not deal with additional polling and time-series data storage.

Due to the nature of the WAN links used by the enterprise, the suitable unit of measure is load percentage on a link. Monitoring the nominal throughput itself would not be practical, because different links have different speeds and so the nominal throughput is not comparable in the context of detecting problematic bottlenecks. PHP Weathermap supports channel load visualization according to percentage and manual configuration of each link's speed, so it was deemed suitable for the task.

3.9 NfSen with NFDUMP for NetFlow traffic analysis

NfSen (NetFlow Sensor)[3] makes use of NFDUMP tools[4] to analyze and visualize NetFlow data and provide a web interface for the operators. After some researching and testing, the NfSen/NFDUMP combination was found to be the only standalone freeware solution to match the requirements of the enterprise.

As outlined in the requirements section, the NetFlow analyzer has to be able to work with sampled NetFlow data (because of the performance considerations of the network devices in use). FlowTools[5] was also considered, but attempts to handle sampled NetFlow with it failed.

The nfcapd NetFlow capture daemon listens to NetFlow exports from the devices and stores the data in hierarchically organized file system structure. Tests and calculations

---

1 Called PHP Weathermap historically and in documentation (e.g. https://network-weathermap.com/manual/0.98)

2 https://network-weathermap.com

3 http://nfsen.sourceforge.net

4 http://nfdump.sourceforge.net

5 https://github.com/adsr/flow-tools

show that with the current network traffic, one year worth of NetFlow data will weigh around 30GB. The NfSen web interface uses the nfdump utility to read and parse this file repository and visualize the flow data as charts. NfSen provides a quick and convenient way for operators to search and filter flows, find top talkers, spot traffic anomalies and more.

3.10 Rsyslog for remote logging

RSYSLOG[1] is a versatile, high-performance syslog server. The requirements call for centralized and multi-tiered logging solution and as RSYSLOG is already used in other parts of the enterprise IT infrastructure, it was chosen to fulfill the required log server functionality. RSYSLOG has support for a wide variety of back-end databases, leaving a path open for future expansion. Initially, it is estimated that a simple file-based storage with proper logrotate[2] and compression set-up will suffice.

RSYSLOG instances will be installed on site-local network management servers and on the global/central server. As with event monitoring, the replication of the log messages will happen closest to the source (i.e. the devices will have multiple log targets), avoiding the need for coordination and synchronization between log servers. RSYSLOG instances on remote sites will collect log data from local devices, providing visibility into events that happen during possible WAN interruptions. The central RSYSLOG instance will collect all log data globally from all network devices.

1 http://www.rsyslog.com

2 https://linux.die.net/man/8/logrotate

4 Implementing the solution

This chapter describes the actual implementation of the new network management toolkit. Installation and configuration is discussed together with discovered issues, necessary modifications and programming of the automation and interfacing scripts. The installation, configuration and scripting done for this work was an elaborate process with copious amounts of input and output. It included numerous small issues and fixes, considerations and choices, candidate material for figures and examples. As it is not practical and feasible to represent all that in written form, the sections in this chapter aim to give an all-round overview and present figures that show the final result.

4.1 Base installation

The Debian Linux distribution was chosen as the operating system for the new network management servers (i.e. the central server and the site-local nodes). At the planning phase of this work, Debian 9 was just released and Debian 8 was the mainstay stable release used throughout the enterprise, so Debian 8 is still used at the time of this writing. Before the support ends in May 2018, the base installation will be upgraded to Debian 9.

As defined among the various requirements and expectations, sourcing the network monitoring and management software through the distribution's package management system is highly desirable. Whenever possible, installing the development dependencies, compiling applications from source and performing custom installation was avoided. At times, this means using a more conservative version of software. As long as the requirements were met, such a trade-off was knowingly accepted for the benefits of concise package management features.

Because small footprint server installation was desired, the central monitoring server was allocated 4 virtual CPU (Central Processing Unit) cores and 4GB of memory. The site-local nodes were allocated a single core and 1GB of memory. As testing indicated, most of the resources are consumed by Nagios. Still, with the central Nagios initially tracking more than 4200 values (while running all the other components and scheduled tasks) the load was moderate. Tests were performed with up to 10000 monitored values with no performance issues. This indicates good scalability properties for future growth.

4.2 CMDB and configuration generation

The central CMDB requirement was satisfied by implementing a small subset of the ITIL (Information Technology Infrastructure Library) CMDB concept. Figure 2 shows an example of device records and the information typically included for every device[1].

```
[
  { "hostname": "ea5-sw1.mg", "ip": "10.4.6.250", "location": "ea5",
"os": "junos", "snmpcomm": "sk5idlQu", "snmpver": "2c", "tags":
["live", "sw", "office"] },

  { "hostname": "ea5-sw2.mg", "ip": "10.4.6.250", "location": "ea5",
"os": "ios", "snmpcomm": "sk5idlQu", "snmpver": "2c", "tags": ["live",
"sw", "office"] },

  { "hostname": "ea5-sw3.mg", "ip": "10.4.6.251", "location": "ea5",
"os": "ios", "snmpcomm": "sk5idlQu", "snmpver": "2c", "tags": ["live",
"sw", "office"] },

  { "hostname": "ea5-ap1.mg", "ip": "10.4.6.248", "location": "ea5",
"os": "ruckus", "snmpcomm": "sk5idlQu", "snmpver": "2c", "tags":
["live", "sw", "office", "wifi"] },

  { "hostname": "ea5-ap2.mg", "ip": "10.4.6.249", "location": "ea5",
"os": "ruckus", "snmpcomm": "sk5idlQu", "snmpver": "2c", "tags":
["live", "sw", "office", "wifi"] }
]
```

Figure 2. Example records of managed devices in JSON format.

Each monitored/managed network device is considered as a CI (Configuration Item), having a record in the central database. Scripts were written to generate configuration for all the monitoring and management components based on these records, meaning, the information about devices and their access parameters will never have to be manually repeated in individual configuration files. JSON (JavaScript Object Notation)

[1] On this and all consecutive figures, some sensitive data like IP addresses, host names, SNMP communities, geographical location names, user account names etc. have been changed to protect confidential data of the enterprise.

file was chosen as the storage means for the device database, as it is a lightweight human-readable format with support in most major scripting languages [5] .

Figure 3 shows the help screen and a few use cases of a command-line utility called “hostlist”, which was created to list and filter the device database based on tags. As it’s not desirable for every script to read and parse JSON, the hostlist utility becomes useful.

```
user@host:~$ hostlist.rb --help
Usage: hostlist [-tofd]
    -f, --file <filename>           Hostlist file in JSON format
    -t, --tag <tag>                 Tag
    -o, --os <os>                   Operating System
    -d, --dump                      Dump all data, not just hostnames
    -n, --numeric                   Print IP addresses instead of
                                    host names
    -h, --help                      Display this screen
user@host:~$ ./hostlist.rb -t sw
ea5-sw1.mg
ea5-sw2.mg
ea5-sw3.mg
ea5-ap1.mg
ea5-ap2.mg
user@host:~$ ./hostlist.rb -t wifi
ea5-ap1.mg
ea5-ap2.mg
```

Figure 3. Help screen and usage examples of the hostlist utility

The key question around the hostlist set-up was, which data to include in the CMDB. Basic information like host name, IP address and SNMP community name are obvious candidates. Because site-local network monitoring servers need to manage only their local infrastructure, a location attribute is needed. As in the future we will want to include support for SNMPv3 (Simple Network Management Protocol version 3), it is also a good idea to provide information about SNMP version used by the device. Much of the technical information like SNMP version, device type (operating system) could also be automatically detected, so including information in the configuration database becomes a balancing act between script performance/complexity and manual data entry/storage. For easy expandability, an array of tags is used in every device entry. This allows to attach information which some automation scripts recognize for their own purposes (e.g. “do something with only ‘lab’ and not ‘live’ devices”) but which do not have any global meaning for other parts of the network management toolkit.

4.3 Autodiscovery

Network interface autodiscovery was by far the largest automation task in this project. For each network interface, the monitoring of four conceptual values was required: interface status, interface errors, interface discards and interface utilization. Because errors, discards and utilization are individual values for both in- and out traffic through an interface, there will be seven individual values tracked for each network interface.

The autodiscovery scripts that generate necessary configuration for Nagios and Cricket first use the hostlist utility to read the device database. They then proceed to poll each device with SNMP, learning about all the interfaces available on the device, their names, descriptions, speeds and current status. Based on interface status and description, an interface may or may not qualify for monitoring. For example, virtual interfaces like SVI (Switch Virtual Interface) and IRB (Integrated Routing and Bridging) are not monitored for discards and errors, whereas physical interfaces like GigabitEthernet and ge-0/0/0 are. Interfaces with disabled administrative status are not monitored, along with interfaces that have the string “NOMON” included in their description. As utilization is tracked and alarms set based on percentage (e.g. warning at 75%, critical at 95%), interface speed is recorded and maximum allowable input/output deltas per time period calculated. Eventually, configuration entries called “target” and “service” are generated for Cricket and Nagios, respectively. Then, configuration files are assembled and written out.

Figure 4 shows a machine-generated excerpt of Nagios configuration for monitoring one interface’s utilization. As can be seen, Nagios is not aware of the overall speed of the interface. The Warning and Critical counter increase values (93750000 and 118750000) have been calculated by the autodiscovery script, indicating a gigabit interface[1].

[1] For utilization tracking purposes, it is not sufficient to decide that interface speed is one Gigabit per second simply because the interface name or type suggests so (e.g. GigabitEthernet 0/8). A Gigabit Ethernet interface may be negotiated or configured to run at 10 or 100 Megabits per second. In such case, utilization monitoring would elegantly fail when relying on interface name. Instead, the ifSpeed or ifHiSpeed SNMP OID needs to be polled to determine the real speed of the interface.

```
define service {
   host_name ea5-swc.mg
   service_description Gi0/8 (ea5-swb gi0/8) HiUtilIn
   check_command check_snmp_rate!Ei1okv!1.3.6.1.2.1.31.1.1.1.6.10108!
93750000!118750000
   use network-service
   notification_interval 0
}

define service {
   host_name ea5-swc.mg
   service_description Gi0/8 (ea5-swb gi0/8) HiUtilOut
   check_command check_snmp_rate!Ei1okv!1.3.6.1.2.1.31.1.1.1.10.10108!
93750000!118750000
   use network-service
   notification_interval 0
}
```

Figure 4. Excerpt of Nagios configuration written by the autodiscovery script.

Figure 5 displays an excerpt of Cricket configuration, written by the autodiscovery script.

```
target GigabitEthernet0-8
  interface-name = "GigabitEthernet0/8"
  target-type    = standard-interface
  inst           = map(interface-name)
  short-desc = "ea5-swc;;gi0-8"
  long-desc = "ea5-swc.mg GigabitEthernet0/8 ea5-swb;;gi0-8"
```

Figure 5. Excerpt of Cricket configuration written by the autodiscovery script.

A number of operational details were considered and solved while implementing the whole automated configuration generating solution. Error handling was of paramount importance, because if any single device becomes unresponsive or slow, the whole monitoring and autodiscovery process should by no means fall apart. If, for some unforeseen reason, the resulting configuration becomes invalid at some point, the monitoring should be able to continue with the last known configuration and notify the operator. An optimal run frequency for the autodiscovery scripts needed to be determined, a task which is a balancing act between operational convenience and performance hit to the monitoring system. A selection of such issues is discussed in more detail in the following sections, which explain the set-up of individual components.

4.4 Setting up the components

The following sections give overview of installing and setting up the main components of the network monitoring and management solution.

4.4.1 Nagios

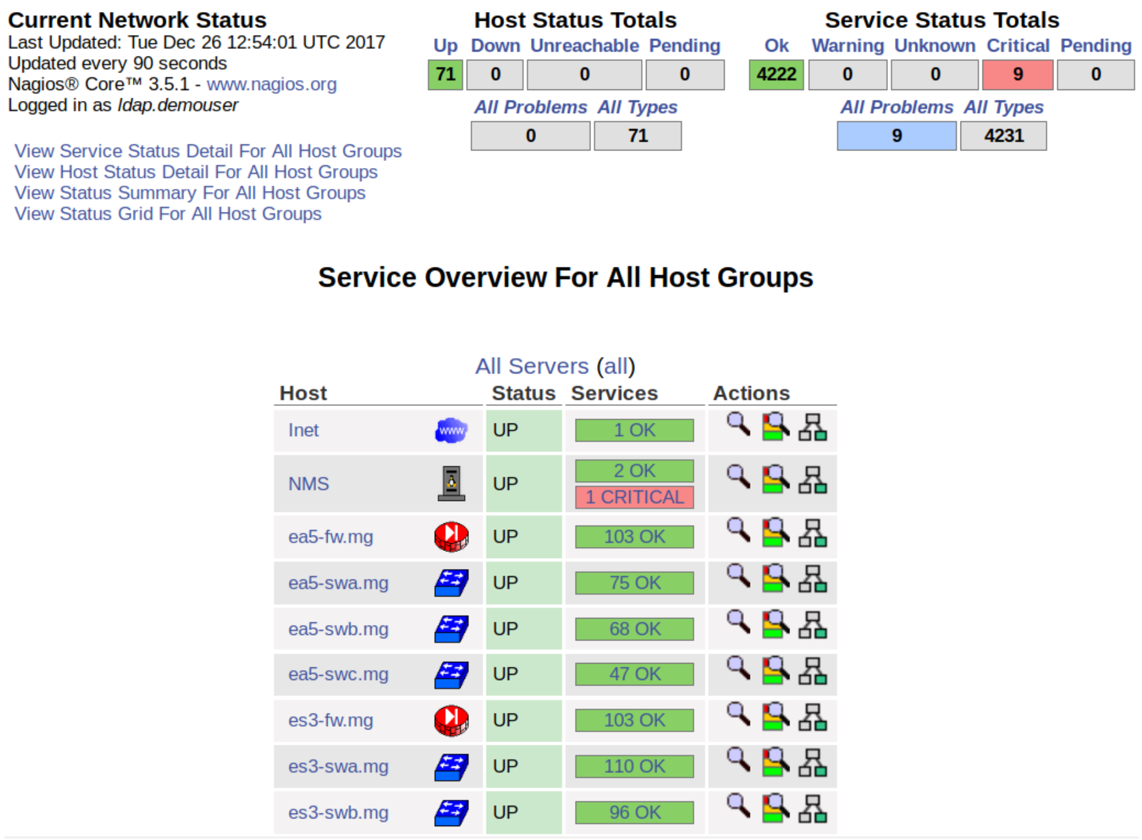
At the time of this writing, major version 3.5 of Nagios is the matured and still widely used release. Though Nagios 4 has been released, it is still not in the main Debian repository, so Nagios 3.5 was used for this project. Nagios was installed from package along with its components (plug-ins, web interface scripts, etc). Main post-installation administrative tasks associated with Nagios were configuring the Apache web server to run Nagios web interface and adjust Nagios base configuration (contact information for notifications, web authentication parameters, etc). For the web interface, LDAP (Lightweight Directory Access Protocol) authentication and valid HTTPS (Hyper Text Transfer Protocol Secure) virtual host were set up.

To satisfy the requirement of SMS notifications, custom notification scripts for Nagios were written. Nagios comes with a configuration file named commands.cfg, which defines commands used for tasks like checking host status or sending an alert. Through this file, the custom scripts which send SMS through Opengear ACM devices, are tied to Nagios. In other words, Nagios does not have knowledge of the phone numbers or any other specifics of sending SMS. Instead, it invokes a script, passes on the notification data and the script takes care of transmitting the message.

The autodiscovery script discussed earlier regularly generates the main part of the Nagios configuration – the hosts and services to monitor. Autodiscovery is invoked through cron every hour. After doing its job and generating the configuration, the autodiscovery process tests the configuration validity (by running “nagios -v” on the new candidate configuration). In case of success, the autodiscovery process moves new configuration in place and sends a HUP (Hangup) signal to the Nagios process.

An important feature of the autodiscovery script is error handling. Each device is handled sequentially by the script and if exceptions occur, the script moves to a new device. An unresponsive device will appear in Nagios as having no services, not disturbing the monitoring process of other devices. Once the device recovers and next scheduled run of configuration autodiscovery happens, the device will once again be fully monitored. It should be noted that if a device is unresponsive by the beginning of scheduled autodiscovery, it would have been noticed by Nagios while still using last good configuration from previous discovery run. Nevertheless, the autodiscovery notifies the NOC (Network Operations Center) by e-mail of any failed device polls and other problems with generating the configuration.

Figure 6 shows part of the summary page of Nagios web interface. At the time of this writing, 4231 individual values were monitored. Tests were run with up to 10000 monitored objects and no significant performance impact was seen. This gives adequate scaling properties for future growth and changes.



Current Network Status
Last Updated: Tue Dec 26 12:54:01 UTC 2017
Updated every 90 seconds
Nagios® Core™ 3.5.1 - www.nagios.org
Logged in as *ldap.demouser*

View Service Status Detail For All Host Groups
View Host Status Detail For All Host Groups
View Status Summary For All Host Groups
View Status Grid For All Host Groups

Host Status Totals

Up	Down	Unreachable	Pending
71	0	0	0

All Problems	*All Types*
0	71

Service Status Totals

Ok	Warning	Unknown	Critical	Pending
4222	0	0	9	0

All Problems	*All Types*
9	4231

Service Overview For All Host Groups

All Servers (all)

Host	Status	Services	Actions
Inet	UP	1 OK	
NMS	UP	2 OK 1 CRITICAL	
ea5-fw.mg	UP	103 OK	
ea5-swa.mg	UP	75 OK	
ea5-swb.mg	UP	68 OK	
ea5-swc.mg	UP	47 OK	
es3-fw.mg	UP	103 OK	
es3-swa.mg	UP	110 OK	
es3-swb.mg	UP	96 OK	

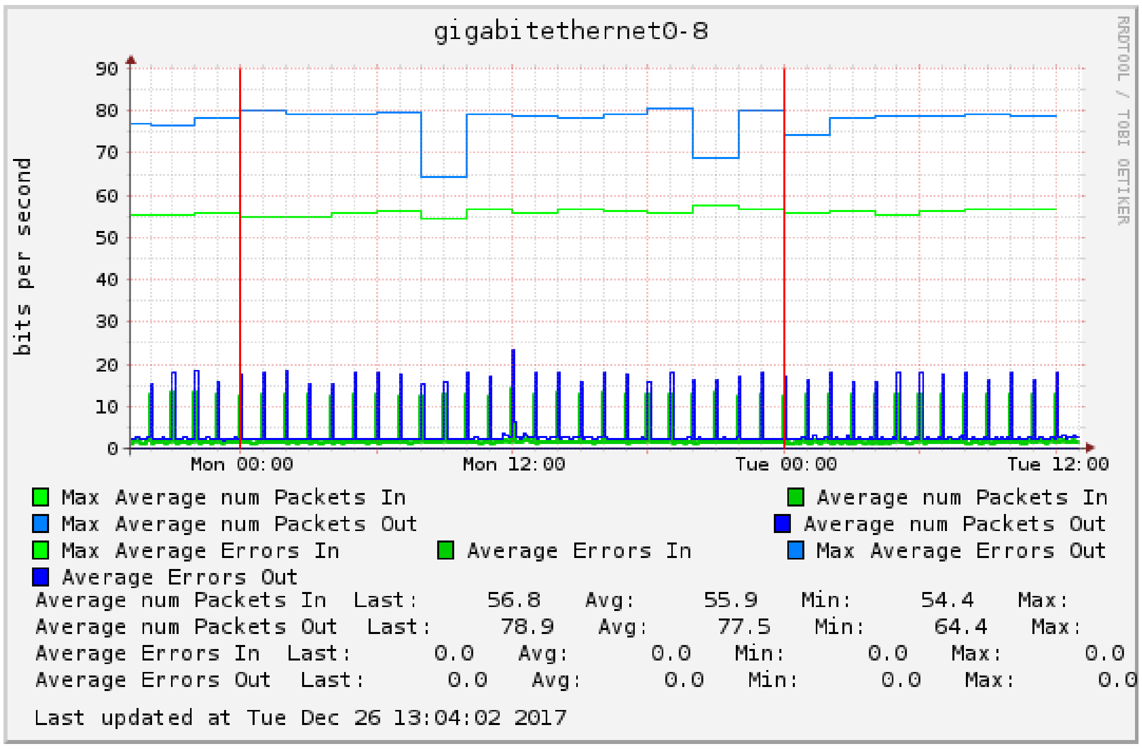
Figure 6. Nagios overview page.

4.4.2 Cricket

Cricket is a mature software package, available through Debian main repositories. Cricket along with its Perl modules and other dependencies was installed and postinstall customizations performed. Cricket web interface runs as a collection of CGI (Common Gateway Interface) scripts under Apache, so proper Apache configuration for LDAP authentication and permissions was performed. Cricket comes by default with the configuration to monitor a few most common network interface OIDs (Object Identifier; ifInOctets and ifOutOctets, i.e. to monitor interface throughput). The enterprise requirement was to monitor much more – interface discard- and error counters, CPU load and memory usage of the devices, routing-engine CPU temperatures, etc. So the main part of post-installation Cricket configuration consisted of defining all the SNMP OIDs and writing the data-source and target definitions that use them.

The autodiscovery solution for Cricket works similarly to that of Nagios’s. Cricket configuration is written in text format but then compiled into a binary database with an included utility. When the generated configuration is invalid, the compilation process fails and Cricket will continue using the last known good configuration. Due to this, a separate verify process and safe restart strategy is not needed (in fact, a restart strategy is not at all an issue, because Cricket is not run as a daemon but regularly invoked by cron). The run interval used for Cricket in this project is 1 minute, the smallest resolution easily achievable. As devices are arranged in a tree-like structure in Cricket configuration (for example, devices in each physical location residing on their own branch), Cricket is easily scalable. Each branch of the tree can be polled by a separate Cricket process, meaning that extremely large installations can be polled within tight time limitations by using parallelism. This is important, because a single run cannot last more than one minute when calculation of a one minute average is desired. The same holds true for any other numbers, of course. The polling process always needs to be faster than the resolution used for averaging. Large or scalability-oriented installations need to plan around this issue diligently.

Figure 7 shows a resulting graph of a network interface parameters generated by Cricket. The web interface enables the operator to choose, which values (measured in different units) to stack on a single graph and which to view separately. All of the tracked values are stacked on one graph here for illustration purposes only.

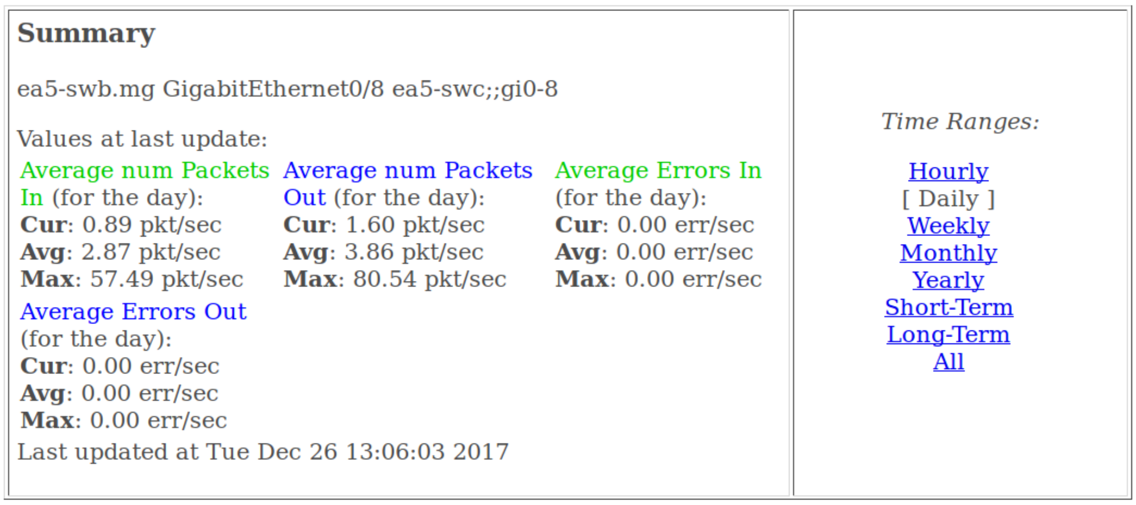


```
Average num Packets In  Last:     56.8   Avg:     55.9   Min:     54.4   Max:
Average num Packets Out  Last:     78.9   Avg:     77.5   Min:     64.4   Max:
Average Errors In  Last:      0.0   Avg:      0.0   Min:      0.0   Max:      0.0
Average Errors Out  Last:      0.0   Avg:      0.0   Min:      0.0   Max:      0.0

Last updated at Tue Dec 26 13:04:02 2017
```

Figure 7. Multiple interface parameters on a single graph produced with Cricket.

Figure 8 shows a text summary of the graph represented in figure 7. For the hourly graph, the initially calculated one minute average deltas are saved. For all longer time periods (seen in the right column on figure 8), consecutive averages based on shorter period data are calculated.



Summary

ea5-swb.mg GigabitEthernet0/8 ea5-swc;;gi0-8

Values at last update:

Average num Packets In (for the day):	Average num Packets Out (for the day):	Average Errors In (for the day):
Cur: 0.89 pkt/sec	**Cur**: 1.60 pkt/sec	**Cur**: 0.00 err/sec
Avg: 2.87 pkt/sec	**Avg**: 3.86 pkt/sec	**Avg**: 0.00 err/sec
Max: 57.49 pkt/sec	**Max**: 80.54 pkt/sec	**Max**: 0.00 err/sec

Average Errors Out (for the day):
Cur: 0.00 err/sec
Avg: 0.00 err/sec
Max: 0.00 err/sec

Last updated at Tue Dec 26 13:06:03 2017

Time Ranges:

Hourly
[Daily]
Weekly
Monthly
Yearly
Short-Term
Long-Term
All

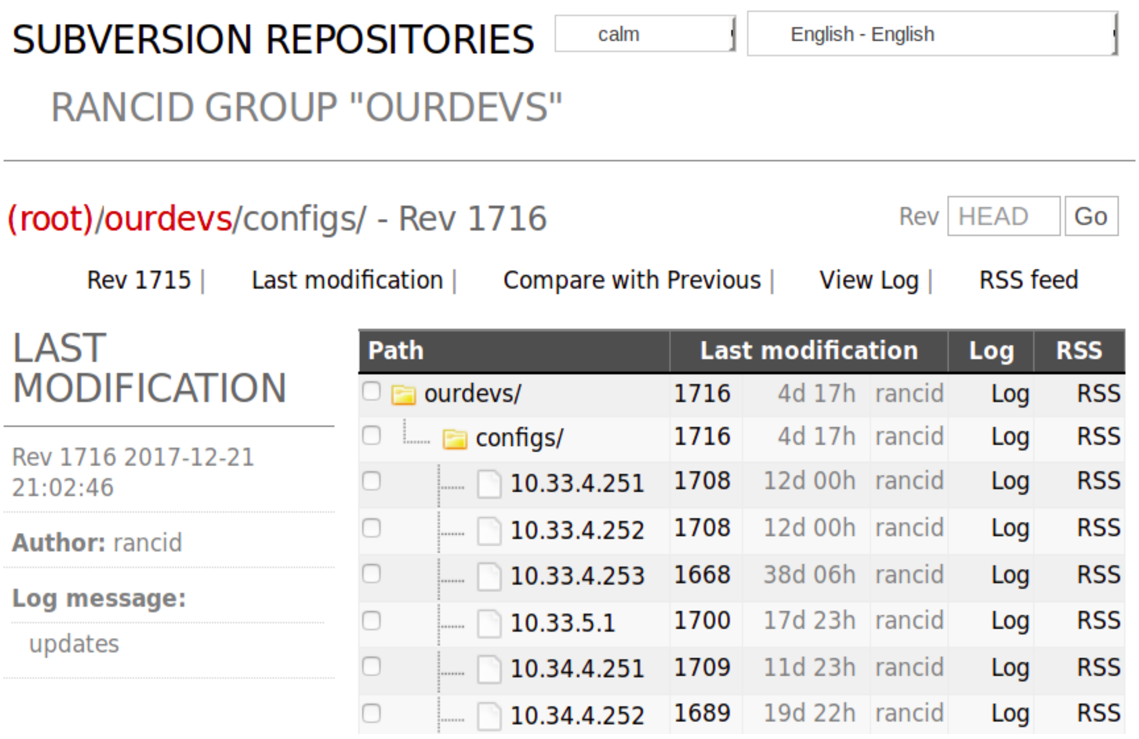
Figure 8. Text summary generated by Cricket to explain each graph.

4.4.3 Rancid

Rancid was installed from Debian package. Rancid is written in Perl and defaults to SVN (Subversion) as its back-end storage, so dependencies include the SVN package and some Perl libraries.

As with previously discussed components, the configuration for Rancid is generated automatically, based on the hostlist information. The auto-generated part of Rancid configuration is a simple human-readable text file with a line of information for each managed device: its IP address or host name, type and status.

To satisfy the web front-end requirement for the configuration management system, WebSVN was installed and configured. Figure 9 shows a portion of the WebSVN interface, displaying the repository of configuration files. The operator can view and compare revisions and changes, download the latest configuration snapshot for device recovery or any earlier known good version, if configuration error is suspected.


SUBVERSION REPOSITORIES
calm
English - English
RANCID GROUP "OURDEVS"
(root)/ourdevs/configs/ - Rev 1716
Rev HEAD Go
Rev 1715 | Last modification | Compare with Previous | View Log | RSS feed

LAST MODIFICATION

Rev 1716 2017-12-21 21:02:46

Author: rancid

Log message:

updates

Path	Last modification			Log	RSS
ourdevs/	1716	4d 17h	rancid	Log	RSS
configs/	1716	4d 17h	rancid	Log	RSS
10.33.4.251	1708	12d 00h	rancid	Log	RSS
10.33.4.252	1708	12d 00h	rancid	Log	RSS
10.33.4.253	1668	38d 06h	rancid	Log	RSS
10.33.5.1	1700	17d 23h	rancid	Log	RSS
10.34.4.251	1709	11d 23h	rancid	Log	RSS
10.34.4.252	1689	19d 22h	rancid	Log	RSS

Figure 9. WebSVN interface displaying a RANCID configuration repository.

Configuration for Rancid is generated once per hour, a process initiated by cron. After this, Rancid itself is likewise invoked by cron and device information is fetched every hour. All managed devices support SSH with public key authentication, which Rancid uses to log in, using its own account. This solves the issue of password security and no individual access credentials configuration for each device needs to be generated.

4.4.4 PHP Weathermap

The PHP Weathermap web application was downloaded and installed under Apache web server. Configuration for weathermap is static, because only a small number of key links and channels will be visualized. Main WAN (Wide Area Network) links and VPN tunnels of the enterprise were chosen for visualization and the data recorded in RRD files by Cricket was re-used to obtain load information.

PHP Weathermap configuration is based on the concept of links and nodes. This does not inherently cater well for visualizing the load on Internet connections, because the many ISPs (Internet Service Provider) used by the enterprise are not “nodes” from the enterprise point of view. To resolve this, “virtual nodes” ISP-1, ISP-2 and so on were used. Figure 10 shows a portion of the completed weathermap.

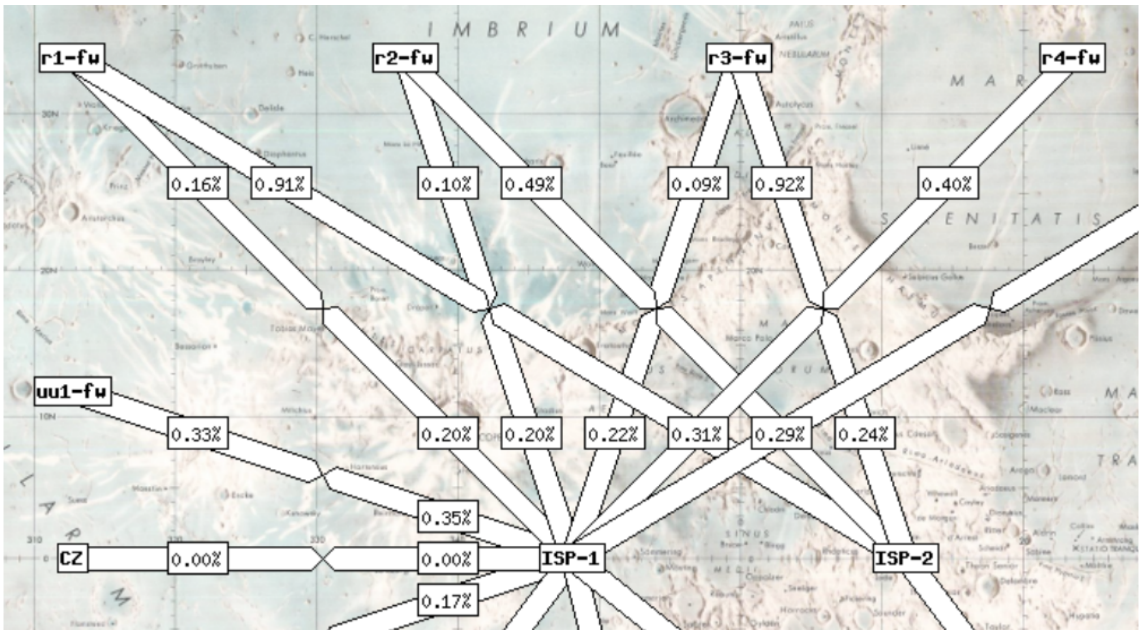


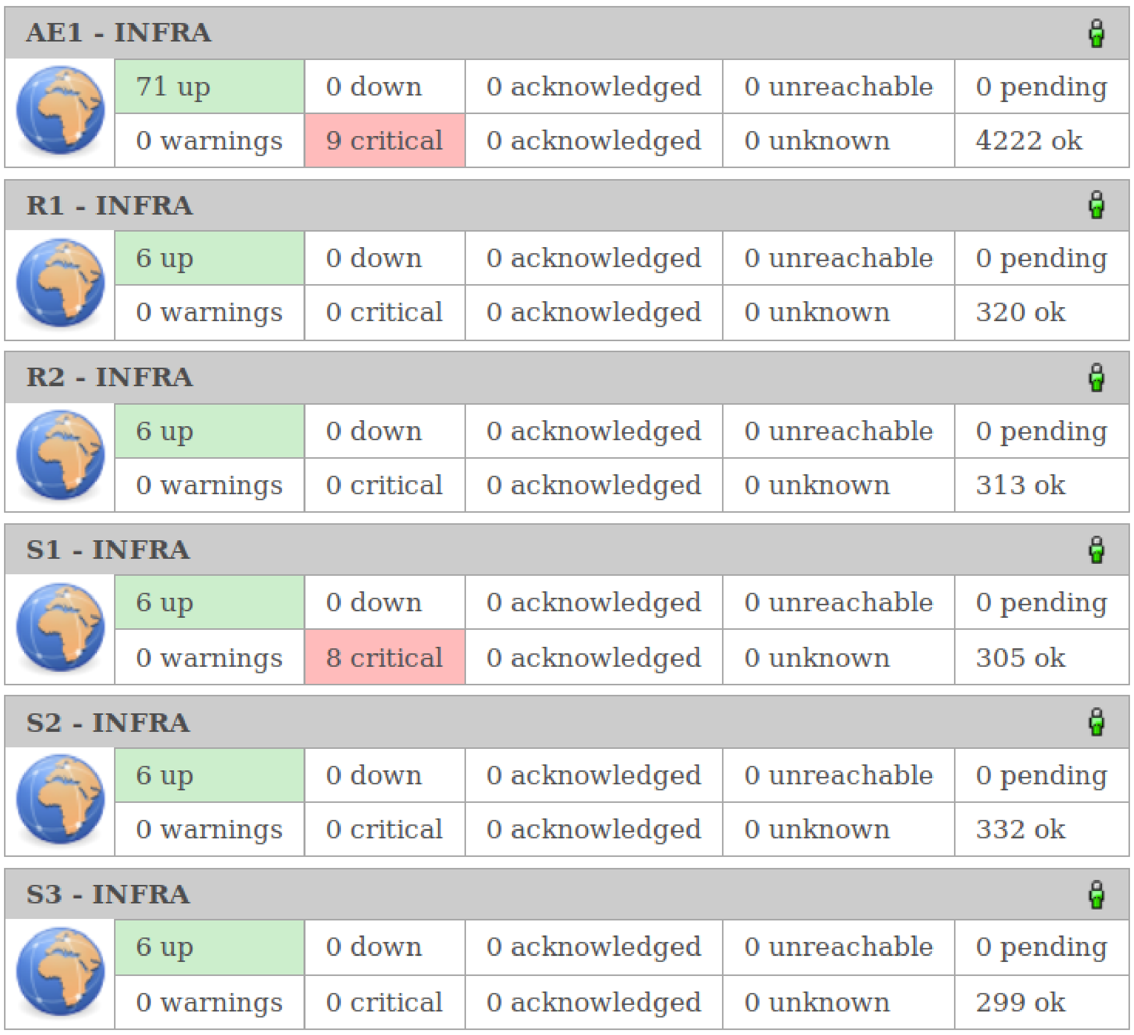
Figure 10. Part of the enterprise network weathermap.

The Weathermap back-end is a PHP script which looks into the RRD files as per configuration and generates a PNG (Portable Network Graphics) image accordingly. The script is invoked by cron every minute, which also defines the map refresh rate for the user.

4.4.5 MNTOS

MNTOS was downloaded and configured to work with Apache web server. As with PHP Weathermap, the configuration of MNTOS is static, defining the URL (Uniform Resource Locator) and access credentials for each Nagios server it is intended to track and aggregate.

Figure 11 shows the completed MNTOS dashboard. Each section of the dashboard shows the summary of one Nagios server. In this example, AE1 is the main/global Nagios and R1 through S3 are site-local Nagios instances.



AE1 - INFRA

71 up	0 down	0 acknowledged	0 unreachable	0 pending
0 warnings	9 critical	0 acknowledged	0 unknown	4222 ok

R1 - INFRA

6 up	0 down	0 acknowledged	0 unreachable	0 pending
0 warnings	0 critical	0 acknowledged	0 unknown	320 ok

R2 - INFRA

6 up	0 down	0 acknowledged	0 unreachable	0 pending
0 warnings	0 critical	0 acknowledged	0 unknown	313 ok

S1 - INFRA

6 up	0 down	0 acknowledged	0 unreachable	0 pending
0 warnings	8 critical	0 acknowledged	0 unknown	305 ok

S2 - INFRA

6 up	0 down	0 acknowledged	0 unreachable	0 pending
0 warnings	0 critical	0 acknowledged	0 unknown	332 ok

S3 - INFRA

6 up	0 down	0 acknowledged	0 unreachable	0 pending
0 warnings	0 critical	0 acknowledged	0 unknown	299 ok

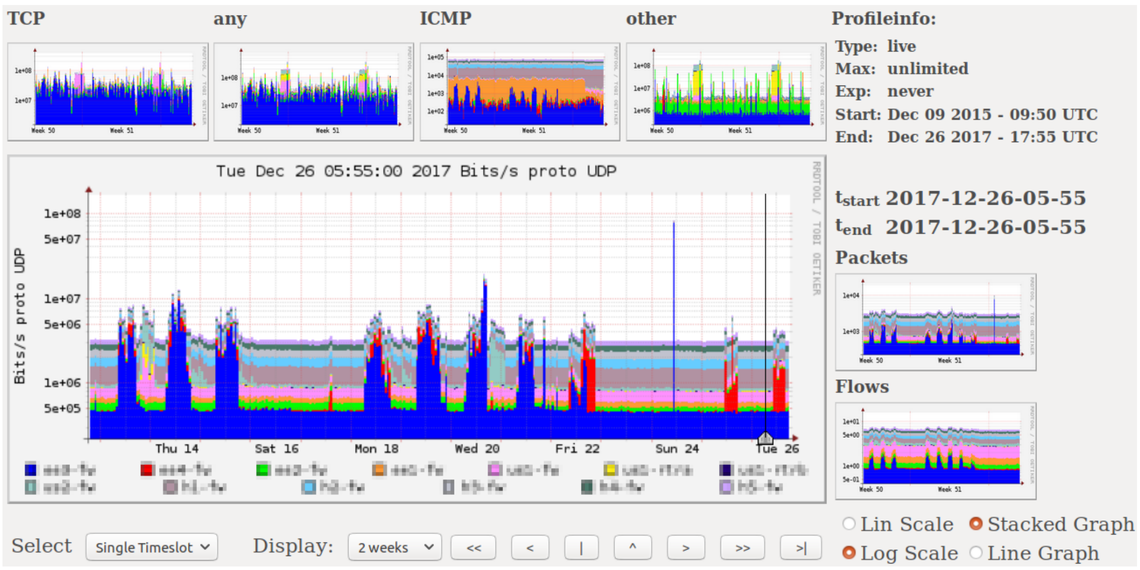
Figure 11. MNTOS provides an overview of all Nagios instances.

The dashboard provides active links to the actual Nagios web interfaces (by clicking on the blue globe image) and to the contact information to each individual operator (by clicking on the green figurine image; if configured).

4.4.6 NFSen

As NFSen and its main dependency, nfdump tools, were not available as Debian packages, they were compiled and installed from source code. NFSen uses the nfcapd daemon from the nfdump tools package to listen to and record NetFlow data. An instance of nfcapd is invoked for each device that exports NetFlow. As each instance uses different UDP (User Datagram Protocol) port, there must be matching entries in the device’s configuration and in NFSen configuration. Automation script was written to generate NFSen configuration. The script will scan device configurations and discover any configured NetFlow export targets (flow-server in Juniper terms). Having this information, the script re-writes the “sources” part of NFSen configuration file and restarts NFSen. This configuration update is invoked by cron every hour, making NFSen automatically follow relevant adds, moves and changes in the network.

NFSen web frontend provides the operator with graphing, searching and analyzing features. Custom graphs can be created based on filters and time ranges. Top talkers by parameters like number of flows, packets per second or bytes per second can be identified by searching and filtering. Figure 12[1] shows a product of the completed NFSen installation, the web interface displaying stacked graph of UDP traffic flows from all NetFlow-enabled devices for the past two weeks.



Profileinfo:
Type: live
Max: unlimited
Exp: never
Start: Dec 09 2015 - 09:50 UTC
End: Dec 26 2017 - 17:55 UTC

t_{start} 2017-12-26-05-55
t_{end} 2017-12-26-05-55

○ Lin Scale [✓ ● Stacked Graph]
[✓ ● Log Scale] ○ Line Graph

Figure 12. NFSen web interface displaying a stacked logarithmic scale graph of all sampled UDP traffic.

[1] Host names of the network devices have been intentionally redacted to protect confidential details.

4.4.7 Programming custom utilities

In addition to the main components of the network management toolkit discussed above, there was need for some custom utilities and tools for the network operators. Most of such utilities fetch data from the network devices and generate HTML (Hypertext Markup Language) views or periodically perform some repeated task, notifying by e-mail if there was a failure. The custom utilities were realized in Bash and Ruby. Some examples are given below.

L2MAP (Layer 2 map) is a utility that scans all network switches and stores information about port names, port descriptions, port VLAN (Virtual Local Area Network) configuration and MAC addresses seen on ports. It then fetches ARP (Address Resolution Protocol) tables from the firewalls and matches MAC addresses to IP addresses. As a result, it generates a HTML table of the whole network, showing the switches, ports, descriptions and MAC and IP addresses of hosts connected to each port. Such utility is very helpful for the system administrators who want to track a server or a virtual machine without having to ask help from the network team.

VerWatch (Version Watch) is an utility that keeps track of software versions installed on network devices. The script uses a configured list of device models and the software versions that are required on them. It then uses the device database to connect to every device and check the version with SNMP. As a result, it generates an HTML report, highlighting any device with wrong software version. If any wrong versions are found, an e-mail notification is also sent. This utility is helpful in managing the software versions of large network installations, making sure no single device is left out when critical software updates are rolled out or when device replacement or recovery occurs.

The network management server periodically runs tens of different custom scripts and utilities, helping to manage the network, detect and avoid human errors, bridge the gap between the network and system administration teams and lower workforce demand by automating repeated tasks. New orders from the operators and system administrators can be taken and custom utilities can be programmed by using rapid prototyping languages like Ruby. The scripts can make use of the data already existing on the network management server, like device configuration files (collected by Rancid), interface monitoring data in RRD files (collected by Cricket), etc.

5 Results and conclusions

As a result of this thesis work, an integrated network monitoring and management system was built, tailored to enterprise requirements. A diverse toolkit consisting of different freeware tools covers the areas of monitoring, configuration management, log management, traffic analysis and network visibility. The automation scripts follow changes in the network so that typical moves, adds and changes do not create significant reconfiguration burden. Previously, the enterprise was using commercial network monitoring and management systems, which did not cover all aspects. The old solutions also required per-device licenses and were not open for customizations and modifications. As a result of this work, all such software products were decommissioned and significant resources vacated.

The ultimate goal of any network monitoring and management system is not the existence that system itself, but the good health of the network it targets. By operating the new solution, a better visibility into the network became obvious. Small problems and opportunities for improvement surfaced in numbers. Unused network interfaces not being disabled, links flapping, gigabit Ethernet links running at speeds 100Mbps or lower due to poor cabling, error or discard counters increasing, a device accidentally skipped during a software upgrade. These are some examples of conditions and events in the network that can now be managed in a more systematic and proactive manner.

The author concludes that tried and proven, simple, mature lightweight tools can be used to replace commercial network management software. By default, such tools often lack the automation and integration features. They would have to be manually configured, replicating and repeating the basic information about managed devices for each and every tool. On the other hand, this issue can be solved quite easily by scripts that automate configuration updates, as the configuration itself is usually in easily manageable format. The main resource consumed throughout the implementation of such a solution is the quite high number of man-hours of work. Once this resource has been deployed sufficiently, the resulting system appears to be adequately reliable, without performance and scaling problems and mostly maintenance free.

It can be also concluded that setting up a system of many disperse freeware components tends to end up being more elaborate task than initially imagined. More detail-level issues need to be solved, more tests run, more error-handling scenarios considered, than initially planned. During the time it took to complete this work, new versions of some components matured to the point that an upgrade should be considered. This brings the author to think about future plans and developments.

A list of future developments and upgrades was identified as the project draw to a close. The underlaying operating system used for the network management servers will soon need to be upgraded to Debian 9. As Nagios 4 matures and especially when Debian packages become available, a Nagios upgrade will be considered. Nagios 4 provides better performance, further conserving the CPU and memory resources. It maintains most of the Nagios 3 configuration syntax, so the automation scripts would not need significant modification. Currently, all network management communication in the enterprise goes through VPN tunnels, so the use of protocols like SNMPv2 (Simple Network Management Protocol version 2) is still possible. It is desirable, however, to upgrade to SNMPv3, a support for which requires some further development and testing. As discussed in the initial parts of this document, a door was left open for adding redundancy to the global network management server, but full 1+1 redundancy was not yet implemented during this project. Redundancy will be added in future developments, with the monitoring and management processes running in parallel and active/passive server status coordinated for alerts and notifications. Last but not least, a full IPv6 (Internet Protocol version 6) support will be implemented and tested. There are many components that claim IPv6 compatibility. However, making full use of it, including IPv6 support in every script that currently uses IPv4 (Internet Protocol version 4), is not trivial.

6 Summary

With this thesis project, a network monitoring and management system was built for an enterprise, using freely available software. The project started with discussing the enterprise needs and prerequisites. Next, the components chosen or required to be used for the project were reviewed, highlighting the features relevant for this implementation. Network monitoring and management areas covered by the project were event monitoring, tendency graphing, configuration management, traffic analysis, baseline data collection, channel visualization and remote logging. A basic design pattern was established throughout the system, a CMDB to which all the components plug in through automation and configuration scripts.

The practical part of the project took on the task of implementing the new system to the point where previously used commercial solutions could be decommissioned, vacating enterprise resources. First, the base infrastructure from the virtual machines and operating system up, along with the main components of the network management toolkit, was installed and configured.

The main goal of the project was to create an automated and unattended system that follows moves, adds and changes in the network. To meet this objective, scripts were written to regularly read the CMDB and generate configuration for the various monitoring and management tools. Custom scripts also take care of various household tasks like sending monitoring alerts via SMS.

The system was used and tested in live enterprise environment, observing how it helped to improve the network, detect hidden problems and react to critical incidents. Finally, some goals for future developments and upgrades were identified, including IPv6 support and addition of full redundancy.

References

[1] A. Nucci, K. Papagiannaki, “Design, Measurement and Management of Large-Scale IP Networks: Bridging the Gap between Theory and Practice”, USA, Cambridge University Press, 2008

[2] J. Tiso, “Designing Cisco Network Service Architectures (ARCH)”, USA, Cisco Press, 2011.

[3] K. Wallace, “CCNP TSHOOT 642-832 Official Certification Guide”, USA, Cisco Press, 2010.

[4] J. Välk, E. Proomann, A. Volkov, “*Example Corporation Inc. andmesidevõrgu projekt ja teostus*”, Tallinn, IT College, 2015.

[5] K. Jõgi, “*Monitooringusüsteemi valik ja rakendamine Spark Systems OÜ näitel*”, Tallinn, IT College, 2009.