TAMPERE UNIVERSITY OF TECHNOLOGY

JOONAS RUOHONEN

Evaluating the Network Management Capabilities of YANG and NETCONF

Master of Science Thesis

Examiner: Prof. Jarmo Harju
Examiner and topic approved by the Faculty Council of the Faculty of Computing and Electrical Engineering on 9th of March 2016

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Signal Processing and Communications
JOONAS RUOHONEN: Evaluating the Network Management Capabilities of YANG and NETCONF
Master of Science Thesis, 77 pages, 12 Appendix pages
May 2016
Major: Communication networks and protocols
Examiner: Prof. Jarmo Harju
Keywords: NETCONF, YANG, network management, NFV, SDN

The purpose of this thesis was to evaluate a new network management solution offered by the YANG data modelling language and the NETCONF network management protocol. The evaluation set out to answer the question if this network management solution is an efficient, secure and reliable way to dynamically manage networks. In addition, the new emerging network paradigms Software Defined Networking (SDN) and Network Function Virtualization (NFV) were examined to understand how this new network management solution relates to them.

The fundamental network management capabilities of network management solutions were examined for the purposes of the evaluation. The used source material included specifications of many network management protocols and research papers related to the field of network management. Moreover, a new YANG model was modelled and a prototype NETCONF server was implemented. The model was done to get a more in depth view of the data modelling process. Whilst, the server was examined to get an understanding of the server's internals and implementation details. The gathered information was used to evaluate the suitability of this new network management solution for the use of SDN and NFV. Furthermore, this new network management solution was analysed based on its strengths, weaknesses, opportunities and threats.

In the evaluation it was observed that this new network management solution is substantially better for network configuration compared to the other solutions. However, it was observed that for network monitoring this new network management solution does not offer the same level of efficiency as some of the other solutions. In addition, this new network management solution was found to offer a more sophisticated and versatile information security and user management controls than the other solutions. The analysis showed that the strengths of this new network management solution were open interface, extendibility, security, reliability and performance, whilst the weaknesses of it were resource consumption and low utilization.

The conclusion of this evaluation was that this new network management solution does offer an efficient, secure and reliable way to dynamically manage networks. However, many of the other network management solutions are still seen as useful solutions in their own limited areas.

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Signaalinkäsittelyn ja tietoliikennetekniikan koulutusohjelma
JOONAS RUOHONEN: YANG-tietomallinuskielen ja NETCONF-verkonhallintaprotokollan tarjoaman verkonhallintaratkaisun ominaisuuksien arviointi
Diplomityö, 77 sivua, 12 liitesivua
Toukokuu 2016
Pääaine: Tietoliikenneverkot ja protokollat
Tarkastaja: Prof. Jarmo Harju
Avainsanat: NETCONF, YANG, verkonhallinta, NFV, SDN

Tämän diplomityön tarkoituksena oli arvioida YANG-tietomallinuskielen ja NETCONF-verkonhallintaprotokollan tarjoamaa uutta verkonhallintaratkaisua. Arvioinnissa pyrittiin vastamaan siihen, onko tämä uusi verkonhallintaratkaisu tehokas, turvallinen ja luotettava tapa hallita tietoverkkoja dynaamisesti. Lisäksi työssä selvitettiin, miten tämä uusi verkonhallintaratkaisu suhtautuu uusiin tietoverkkojen toteutustapoihin Software Defined Networking (SDN) ja Network Function Virtualization (NFV).

Arviointia varten työssä tarkasteltiin verkonhallinan kannalta keskeisiä ominaisuuksia. Lähdeaineistoon kuului verkonhallintaprotokollien määrittelydokumentteja ja aiheeseen liittyviä artikkeleita. Kirjallisen selvityksen lisäksi arviointia varten mallinnettiin uusi YANG-tietomalli sekä toteutettiin prototyyppi NETCONF-palvelimesta. Tietomallia ja sen mallinnuksessa käytettyä prosessia käytiin läpi, jotta saatiin tarkempi näkemys tietomallinnuksen yksityiskohdista. Palvelinta ja sen sisäistä rakennetta avattiin, jotta pystyttiin tarkemmin selvittämään palvelintoteutuksen yksityiskohtia. Kerätyn aineiston pohjalta arvioitiin tämän uuden verkonhallintaratkaisun soveltuvuutta SDN:n ja NFV:n käyttöön. Lisäksi tätä uutta verkonhallintaratkaisua analysoitiin esittämällä kerätystä aineistosta sen käyttöönoton vahvuudet, heikkoudet, mahdollisuudet ja uhat.

Arvioitaessa tätä uutta verkonhallintaratkaisua havaittiin, että se tarjoaa huomattavasti muita käsiteltyjä ratkaisuja paremmat lähtökohdat tietoverkkojen asetusten dynaamiseen muokkaamiseen, mutta osittain heikommat lähtökohdat tietoverkkojen tilan valvontaan. Lisäksi tämän uuden verkonhallintaratkaisun tietoturva ja käyttäjähallinta nähtiin muita käsiteltyjä ratkaisuja kehittyneemmäksi ja monipuolisemmaksi. Tämän uuden verkonhallintaratkaisun vahvuuksiksi nähtiin sen avoin rajapinta, laajennettavuus, turvallisuus, luotettavuus ja suorityskyky. Heikkouksiksi nähtiin resurssien käyttö ja alhainen käyttöaste.

Arvioinnin tuloksena päädyttiin siihen, että tämä uusi verkonhallintaratkaisu tarjoaa tehokkaan, turvallisen ja luotettavan tavan hallita tietoverkkoja dynaamisesti. Muut verkonhallintaratkaisut nähtiin kuitenkin edelleen hyödyllisiksi omissa rajatummissa käyttötarkoituksissaan.

PREFACE

Finally... The time since I finished the last actual course at the university seems like ages ago, yet the final task of writing this thesis lingered on for quite a while longer than it should have. Although I finally picked up the pen and started writing, it should have happened two years earlier. This is not to say that I did not enjoy researching this topic, on the contrary, I found the subject to be quite intriguing, timely and important, although time-consuming. Yet, this thesis would not have been what it is without the help and support from other people.

First of all I would like to thank my lovely wife Satu for supporting me all these years and for *constantly* kicking me forward to finish this thesis. We agreed in Rome in autumn 2014 whilst sharing a bitter glass of grappa that I would start and finish this thesis before summer, but I do not think we explicitly specified a year, so now I can gloat and say that I actually finished it before summer. (*wink*)

I would also like to thank Jarmo Harju and Mika Joutsenvirta for taking their time to read this thesis and give valuable feedback to make it better. Furthermore, I would like to thank Insta DefSec Oy for providing me a great place to work in and spend time on this thesis. In addition, I would like to thank all of the great team at work for providing an environment to share and develop ideas for this thesis. Finally, Tampere University of Technology deserves a word of thanks for all these past 7 years of exquisite studies.

Sincerely,
Joonas Ruohonen

CONTENTS

1 Introduction 1
2 Network Management 2
2.1 Network Function (NF) 2
2.2 Network Configuration 3
2.3 Network Monitoring 6
2.4 Software Defined Networking (SDN) 9
2.5 Network Function Virtualization (NFV) 14
3 YANG - Data Modelling Language 16
3.1 Language Concepts 16
3.2 Language Features 21
3.3 Existing Models 25
4 NETCONF - Network Configuration Protocol 27
4.1 Protocol Concepts 28
4.2 Network Management Features 34
4.3 Security Measures 36
4.4 Existing Implementations 39
5 YANG Model Development 41
5.1 Network Function Instrumentation 41
5.2 Instrumentation Development 43
5.3 Modelling Network Functions 48
6 Prototype NETCONF Server 52
6.1 Software Architecture 52
6.2 Modular Design 55
6.3 Security Measures 60
7 Analysis of YANG and NETCONF 64
7.1 Comparing Network Management Solutions 64
7.2 SWOT Analysis 67
8 Conclusions 72
References 74
Appendix A 78

LIST OF SYMBOLS AND ABBREVIATIONS

API *Application Programming Interface* is a service description that defines how to call a set of operations and how those operations respond. APIs are used to abstract implementation details and thus allow different implementations be accessed in a unified way.

CLI *Command Line Interface* is a primitive user interface used to provide a set of commands to manage a device.

DSL *Domain Specific Language* is a language designed for a specific application domain to express it more clearly than any existing language.

ETSI *European Telecommunications Standards Institute* is an European standardation organization focused on telecommunications field.

IETF *Internet Engineering Task Force* is an open standardation organization that develops Internet Standards.

IKEv2 *Internet Key Exchange version 2* is a reimplementation of the IKE protocol. IKEv2 brought a number of changes to the IKE specification and unified the wide spread of RFCs under a single more consistent document.

IPC *Inter-Process Communication* is a communication mechanism for two or more processes to share data with each other.

IPsec *Internet Protocol security* defines two protocols, Authentication Header (AH) and Encapsulating Security Payload (ESP), which are used respectively to provide integrity and authenticity or integrity, authenticity and confidentiality for internet traffic. IKE is used for automatic keying of IPsec, but it can also be managed manually.

NACM *NETCONF Access Control Model* is an access list based mechanism used to limit the allowed actions of the NETCONF protocol users.

NF *Network Function* can describe any network element like a switch, a router or a middlebox that performs some network operations.

NFV *Network Function Virtualization* focuses on virtualizing network resources, referred as functions, and creating network services by connecting and chaining these functions together. Reducing network maintenance costs is a major driver for NFV.

NETCONF *Network Configuration Protocol* is an IETF stantardized protocol used to manage network devices.

RFC *Request for Comments* is a technical document published by the Internet Engineering Task Force. RFCs specify many of the protocols and practices used in the Internet.

RPC *Remote Procedure Call* is an invocation of operation that is executed in an external process, usually on another machine. RPCs are a form of IPC.

SNMP *Simple Network Management Protocol* is an IETF stantardized protocol used mainly to monitor network devices, although the protocol also supports management features.

SDN *Software Defined Networking* focuses on allowing a programmable access to network nodes and and delegating the decision making to an external entity, thus allowing to separate the data plane (packet forwarding) from the control plane (routing logic).

SMIv2 *Structure of Management Information Version 2* is a modelling language defined for the SNMP protocol and it closely relates to the YANG modelling language. SMIv2 is used to define Management Information Base (MIB) modules.

SSH *Secure Shell* is a cryptographic protocol designed to secure network communications. SSH is mainly used to provide secure remote access.

TLS *Transport Layer Security* is a cryptographic protocol designed to secure network communications. TLS is mostly used to provide a secure channel between two applications.

VNF *Virtual Network Function* can describe any virtualized network element like a switch, a router or a middlebox that performs some network operations.

YANG *Yet Another Next Generation* is a data modelling language for the NETCONF protocol. YANG is used to provide a management API for network devices.

1 INTRODUCTION

Network management is relatively simple when the networks being managed are small, consisting of maybe five to ten different network functions (NF). However, the network management becomes increasingly complex when the NF counts go up. In 2012, Sherry et al. conducted a survey on network deployments across enterprises of varying size and found out that the networks contained substantial amounts of middleboxes and Layer 3 (L3) routers, with the average network ranging from 17 on small to 4800 on large enterprises [1]. A typical L3 router or a middlebox contains multiple NFs, for example a router could contain NFs such as NTP and DHCP servers in addition to the routing functionality. The survey also found out that network management personnel required to manage these networks ranged from few on small enterprises to hundreds on large enterprises [1]. Furthermore, the network management complexities got underlined when most of the administrators in the survey (˜62%) estimated misconfiguration to be the most common cause for network failures on middlebox deployments [1]. Two other studies conducted in 2003 by Oppenheimer et al. [2] and 2004 by Kerravala [3] also conclude that the number one reason for network outages is human error. In addition, Kerravala's study notes that 80% of enterprise IT budgets is spent on maintaining the status quo, leaving only scraps for new developments [3].

This thesis explores how network management could be made less error-prone by using the network management protocol NETCONF and its data modelling language YANG. The NETCONF protocol offers programmatic access to the managed NFs that can be used to achieve automated network management solutions that control network dynamically and efficiently. Furthermore, the thesis focuses on evaluating this approach compared to the existing network management solutions.

The thesis starts by exploring and explaining the different aspects of network management which involve mainly configuration and monitoring. These aspects are sometimes combined in programs called network management systems. In addition, the new emerging network paradigms Software Defined Networking (SDN) and Network Function Virtualization (NFV) are briefly explored since the YANG modelling language and the NETCONF protocol both serve as enablers for both technologies in some degree. After the introduction to network management, the concepts of YANG and NETCONF are discussed in their own chapters. With the basis laid out, the thesis moves on to give some insight on the how to use YANG and NETCONF to actually integrate into existing systems. In addition, a prototype NETCONF server implementation made for the purposes of this thesis is presented and its details are discussed. Finally, the theory and prototypes are used to evaluate the network management capabilities of YANG and NETCONF. In the evaluation the focus is on what are the strengths and weaknesses of using YANG and NETCONF and what are the offered opportunities and foreseeable threats.

2 NETWORK MANAGEMENT

There are plethora of different kinds of networks, ranging from small home networks through enterprise intranets and extranets to wide area networks operated by service providers. These networks have very different needs for their network management requirements. Small home networks consisting of few devices require low management overhead and can often be most efficiently controlled manually, whereas larger networks from ten to few hundred devices require already quite systematic approach to cope with the complexities that rise from the sheer size of such networks. As the network grows larger it becomes apparent that a management solution is required to help administrators to cope with the complex structure.

Virtualization has provided network administrators cost effective solution to segment the network in different layers of the OSI model. On data link layer came the first network virtualization solutions in the form of virtual local area networks (VLANs). With VLANs administrators could get the same functionality of network segmentation with a fraction of the devices earlier required. On network layer then came virtual routing and forwarding (VRF) that allowed administrators to segment the IP networks. With VRFs administrators could serve multiple networks again with a fraction of the devices earlier required. And now we are in the advent of full blown virtualization where almost all of the elements in the network support virtualization as virtual machines that virtualize all of the layers in the OSI model. The increased use of virtualization has cumulated into huge amount of functionality in the network that administrators have to cope with. Whilst the amount of physical devices has diminished, the network segmentation and complexity has increased.

In this chapter a thorough look into the network management is given. The network management consists of fault, configuration, accounting, performance and security (FCAPS) management as standardized by the International Organization for Standardization (ISO) [4], also referred to as *OSI network management model*, which in this thesis are roughly merged into two categories: configuration and monitoring. Furthermore, the concepts of SDN and NFV are discussed.

2.1 Network Function (NF)

Network Function (NF) is a term used a lot in marketing materials and white papers dealing with the newest network management paradigms. However, NF is a very poorly defined term. Most of the papers pass on the definition altogether and others give vague descriptions such as "Network Function (NF): functional block within a network infrastructure that has well-defined external interfaces and well-defined functional behaviour" [5], which is understandable in that NF could potentially mean any kind of functionality in network, but for the purposes of this thesis a more concrete and detailed definition, specific to IP-based networks, is given:

> ***Network Function (NF)*** *is a hardware or software component, part of a network device, that implements one specific functionality. One network device can hold multiple NFs within. A typical NF requires at least connectivity at physical and data link layers in order to function properly. Examples of a NFs are MAC filters, OSPF instances, IPsec gateways and HTTP proxies.*

Even the simplest NFs have some basic requirements for their execution environment. NFs functioning on data link layer usually do not require anything but a platform capable of sending and receiving messages from a given medium. NFs at network layer usually require IP connectivity and possibility to configure the IP-layer, thus they at minimum require addressing and static routing functionalities. In addition, common functionalities that can be taken as granted in NFs include network services such as Network Time Protocol (NTP) and Domain Name System (DNS) clients. These common functionalities are not considered in this thesis as individual NFs, but rather as a single underlying NF referred to as *system*.

In some cases multiple NFs must be chained inside a single execution environment to fulfil required functionalities. Border gateway is an example of such functionality because it might be required to speak Interior Gateway Protocol (IGP) to some links and Exterior Gateway Protocol (EGP) to some other links and these functionalities can not be separated to different machines. Another example is Protocol Independent Multicast Sparse Mode (PIM-SM) router that might require some IGP protocol to figure out the network layout. However, even if two or more NFs must be bound together, they still count as multiple NFs rather than one. Furthermore, it is very common in network devices to bundle tens of NFs into one platform, but this could change in the future with the advent of NFV.

2.2 Network Configuration

The OSI network management model splits network management into five distinct functional areas, namely into fault, configuration, accounting, performance and security management [4]. This thesis bundles these into two categories of which the first is network configuration. Network configuration in this thesis is considered as the process of altering the NFs' internal configurations. Network configuration is related to the five areas defined in OSI network management model as follows:

Fault management requires network configuration in order to correct the faults in the network.

Configuration management as a whole is part of the network configuration from the initialization through the operation until the termination of the NFs.

Accounting management requires network configuration in order to set the accounting limits.

Performance management requires network configuration in order to alter the network to offer better resource utilization based on the performance management activities.

Security management can be considered as a sensitive configuration management and thus relates to network configuration the same way as configuration management.

Network configuration is an error-prone task as found out by Sherry et al. [1] and it becomes increasingly more difficult as network sizes grow. Automated configuration generation could potentially help on this matter as the chance of human errors is reduced, granted that the automation works correctly. Another problem faced in network configuration is scalability as the amount of management data and NFs grows [6].

There are multiple ways to configure network elements available to administrators, three of which are configuration files, Command Line Interface (CLI) and Simple Network Management Protocol (SNMP). In this section we explore the characteristics of each one individually. In addition, a couple of different proprietary solutions for network configuration are briefly discussed at the end of this section.

2.2.1 Configuration Files

The most prevalent way on configuring NFs running on open source operating systems is configuration files. Configuration files usually employ a domain specific language (DSL) that is used to describe the configuration in format that is close to the implementation details of the given program that is being configured. Examples of these DSLs include custom syntaxes such in Figure 2.1 and more generic solutions such as JSON or INI files. The program that is configured using configuration files reads the configuration at start up and sometimes has a command or signal to reload the configuration as necessary at runtime.

```
listen on 192.168.0.1
servers fi.pool.ntp.org weight 1
```

Figure 2.1: *Example of NTP server configuration file*

Usually open source network elements deploy from few to dozens of different programs that all require their specific kind of configuration files and it is rare that any two programs would share common configuration model or syntax. For administrators this approach weighs quite a burden because each element in the network requires its own local configuration files in their own syntax. The problem is further amplified with virtualization where each virtual instance requires its own set of configuration files.

The benefit of configuration files is that they usually very closely resemble the internal information structure of the programs and they are easy for the developers to understand since they themselves create the syntax. Because the configuration files are located in local filesystem of the network elements, modifications to them usually require remote filesystem access. Thus, configuration files are very static in nature and incapable of dynamic changes if the network around them is modified.

2.2.2 Command Line Interface

CLIs are the most common approach to allow network configuration in enterprise grade networking devices. This approach is taken by many network equipment companies, although these companies also usually allow some other configuration interfaces for their devices. CLIs are just frontends for the administrators and their internal implementation could be very different between different vendors and even between a vendor's own devices [6]. The CLI can use an internal API to dynamically change logic of the network functions or it could transform the CLI commands to an internal configuration file and then reload the configuration when the administrator is done editing through the CLI. An example of a CLI transaction is shown in Figure 2.2.

```
 > configure terminal
#> ntp server fi.pool.ntp.org
#> access-list ntp-accesslist permit ip 192.168.0.1 0.0.0.0
#> access-list ntp-accesslist deny ip any any
#> ntp access-group query-only ntp-accesslist
#> !
```

Figure 2.2: *Example of NTP server CLI transaction*

CLIs impose the same burden as configuration files to the administrators since there is no unified CLI that every device supports and the CLI commands can vary drastically even between vendor's own CLI implementations. This leads to administrators needing to learn a number of different CLIs to configure their whole network. However, CLIs can offer somewhat more dynamic modifications compared to configuration files and thus can be programmed, although with difficulty, to respond to dynamic modifications of network around them. [6]

2.2.3 Simple Network Management Protocol (SNMP)

SNMP is a network configuration and monitoring protocol standardized first time in 1988 by IETF. SNMP is a UDP based protocol client-server protocol where the device in client role that is managing the NFs is called *manager* and the NFs being managed are called *agents*. SNMP's network management capabilities are represented in Management Information Base (MIB) modules that are used to expose *agents*' management data to the *manager*. By knowing MIB modules that expose the *agent*'s configuration data via SNMP to the *manager*, the *manager* could dynamically alter the *agent*'s configuration. [7][8]

Using SNMP to manage network configuration has mostly been an academic endeavour [9] rather than one of practical deployments [6]. Reasons that have discouraged the use of SNMP are the fact that SNMP did not support authentication of the *manager* until the release of version 3 [10] in 1999 and that SNMP lacks the support for more complex configuration manipulation capabilities such as easy retrieval and playback of configurations [6]. But even more prevalent reason why

SNMP is not used for configuration is the lack of standardized MIB modules that could be used to configure NFs [6].

2.2.4 Other Approaches

Other approaches for the network management include web page based solutions that enable the administrator to modify the device with a graphical presentation of the device's capabilities, which is considered more user friendly but more limiting and management application hostile than CLIs [6]. Also some devices offer a proprietary protocol to configure the devices configuration. Usually these solutions employ some sort of transformation from the frontend presentation to the backend configuration, which is usually configuration files on open source solutions.

These solutions are all very specific to a single device or device family and usually do not even try to be compatible with each other. These solutions also weigh burden for the administrators since each solution requires them to learn yet another way to configure the NFs. [6]

2.3 Network Monitoring

As noted earlier, the OSI network management model splits network management into FCAPS areas and this thesis bundles these into two categories of which the second is network monitoring [4]. Network monitoring in this thesis is considered as the process of supervising the NFs' internal state. Network monitoring is related to the FCAPS areas as follows:

Fault management requires active fault detection capabilities that can be achieved with network monitoring.

Configuration management requires network monitoring to collect information of the current network state and to receive notifications of significant changes in the network.

Accounting management requires network monitoring in order to keep track of resource consumption.

Performance management requires the gathering of statistical data and history of the network utilization via network monitoring.

Security management requires network monitoring to receive reports of security related events.

Therefore, network monitoring includes keeping a near real-time view of the state of the network so that administrators can react to problems and see the utilization of the network. There are multiple solutions available for administrators to monitor network elements, three of which are logging, SNMP and IP Flow Information Export (IPFIX). Each of these solutions offer different set of functionalities to expose the network state. In this section we explore the characteristics of each one individually. In addition, a couple of different proprietary solutions for network monitoring are briefly discussed at the end of this section.

2.3.1 Logging

A very common way for NFs to communicate their state changes and other important events such as security incidents is logging. A widely used protocol for sending log messages to remote systems is the syslog protocol which is an IETF standardized protocol that defines the message header and some of the formatting rules when serializing messages [11]. However, the syslog protocol does not force any format which the syslog messages must use, although there has been an attempt to define common format for log messages using structured data in the standard [11].

The syslog protocol can be enhanced with TLS and RELP[1] protocols to offer security and reliability. However, only some implementations and deployments support these advanced protocols leaving the UDP based syslog protocol as the most commonly used way of delivering log messages to remote servers. The problem with UDP, and to some extent TCP[2], in conjunction with network monitoring is unreliability which might cause the system monitoring the network to miss important events [11].

Logging free form messages using protocols such as syslog to remote servers for persistent storing is a common practise, but sometimes these messages are also parsed to gather more detailed information of the network state [12]. An example of a syslog message can be seen in Figure 2.3. The upside of this approach is that, as logging is de-facto way of noting state changes for many programs, most of the state changes can be monitored. However, the downside of this approach is the fact that the messages are usually free form and thus highly implementation specific which makes them hard to parse and prone to constant changes as the NFs are developed further. Furthermore, it could be argued that logging was not designed for active network monitoring, but rather for storing records of past events.

```
<14>Jan  4 13:30:20 bob ntpd[7714]: adjusting local clock by 8.044731s
```

Figure 2.3: *Example of log message sent by NTP server*

2.3.2 Simple Network Management Protocol (SNMP)

SNMP was presented earlier in the network configuration section, but it also offers network monitoring capabilities. As noted earlier, using SNMP for configuring devices has not been a common practice, however monitoring devices with SNMP has been the de-facto standard almost since its publication [9]. The vast majority of MIB modules standardized for SNMP deal with NFs' state information and even custom vendor specific MIBs have been created [6].

In addition to the *manager-agent* type of communication, SNMP offers network monitoring mechanisms called traps and notifications that can be used to push lightweight event notifications to the *manager* asynchronously. SNMP traps are

[1]The Reliable Event Logging Protocol. Available at: `http://www.rsyslog.com/doc/relp.html`. Accessed 10.02.2016

[2]On the (un)reliability of plain tcp syslog... Available at: `http://blog.gerhards.net/2008/04/on-unreliability-of-plain-tcp-syslog.html`. Accessed 10.02.2016

used in network monitoring systems to receive error and failure notifications from the NFs [12]. Open source examples of network monitoring systems include Nagios[3] and Zabbix[4] which use extensively SNMP to query state information and to receive notifications. An example of SNMP state query on Linux machine is shown in Figure 2.4.

```
#> snmpwalk -c public -v 2c localhost ifName
IF-MIB::ifName.1 = STRING: lo
IF-MIB::ifName.2 = STRING: eth0
IF-MIB::ifName.3 = STRING: eth1
IF-MIB::ifName.4 = STRING: eth2
```

Figure 2.4: *Example of SNMP query of local interface names*

Since SNMP can access quite detailed information of the NF's internal state, depending on the MIBs installed to the device, it was eventually standardized to include authentication with the third revision of the protocol (SNMPv3) which, although being much more secure protocol, has not completely replaced its earlier unauthenticated version (SNMPv2c).

2.3.3 IP Flow Information Export (IPFIX)

IPFIX is an IETF standardized network monitoring protocol used to export IP flow information from NFs [13]. The protocol defines the format which is to be used when transmitting the flow information and it also defines the method of transportation. The protocol defines *exporters* and *collectors* where the *exporters* gather flow information from the network and send it to one or more *collectors*. An example of simplified message contents sent in IPFIX messages can be seen in Table 2.1, which shows two sets of messages received one minute apart reporting the amount of packets that have been received since the last report. In addition, the IPFIX standard specifies a method to implement custom data types and ordering in the protocol messages thus allowing customization of the protocol to fit for different network monitoring needs [13].

Table 2.1: *Example of IPFIX messages containing flow information*

Timestamp	Source	Target	Packet Delta
2016-01-02 12:30:12	172.16.1.1	192.168.1.1	30
2016-01-02 12:30:12	172.16.1.1	192.168.2.1	245
2016-01-02 12:30:12	172.16.1.1	192.168.3.1	3
2016-01-02 12:31:12	172.16.1.1	192.168.1.1	700
2016-01-02 12:31:12	172.16.1.1	192.168.2.1	32
2016-01-02 12:31:12	172.16.1.1	192.168.3.1	1

[3] A monitoring system. Available at: `https://www.nagios.org/`. Accessed 01.04.2016
[4] A monitoring solution. Available at: `http://www.zabbix.com/`. Accessed 01.04.2016

IPFIX is a binary protocol designed to conserve bytes on the wire and can be considered a very lightweight protocol. In addition of being lightweight, IPFIX also employs so called *push* mechanism so that *collector* does not need to actively *poll* the *exporter* for information, but rather it receives updates as packets flow in the network. IPFIX supports SCTP, TCP and UDP as transport protocols which can be secured by using TLS protocol with mutual authentication between the *exporter* and the *collector*. [13]

2.3.4 Other Approaches

The other approaches at network monitoring include utilities such as small programs and scripts that are run on the NFs to gather information of the NF's state. These utilities then export the gathered information using either output files with specialized file formats or custom communication protocols. Monitoring utilities such as these usually offer very detailed view of the monitored NF, but do not output any easily processable data and are usually rather slow and thus not applicable for continuous real-time monitoring of the network [6].

Also different kinds of packet sniffers and *bump-on-the-wire* devices are used to gather performance statistics from the network. However, such solutions could efficiently be replaced by use of advanced standardized protocols such as IPFIX without introducing new network elements. There are also some custom network monitoring solutions specialized in security monitoring, which actively scan the network for open ports and services.

2.4 Software Defined Networking (SDN)

SDN proposes a network management architecture that decouples the control plane containing the network logic from the data plane that forwards the packets in the network. This approach allows the network control logic to evolve as independent software that is not bound to specific hardware platform. The decoupled control approach allows programmability, automation and network control in greater extent than before. SDN has become a widely used term in recent years especially after publication of Open Networking Foundation's (ONF) SDN white paper in 2012. However, the origins of SDN and programmable network concepts date back to 1996 to the specification of General Switch Management Protocol (GSMP) [14] that first proposed the separation of control and data plane [15]. [16]

The steady evolution of network management paradigms and their implications to control and data planes is visualized in Figure 2.5. The figure shows some of the protocols that contributed to the evolution of both planes along the years. The data plane was revolutionized in the advent of network routing protocols such as the early Gateway-to-Gateway Protocol (GGP) and the Router Information Protocol (RIP) that enabled dynamic updates to the data plane and thus allowed the network to react on changes on the network [17]. These early routing protocols were later replaced by protocols such as the Open Shortest Path First (OSPF) protocol and the Border Gateway Protocol (BGP). The control plane remained static quite a bit longer than data plane and dynamic operation was enabled, but rarely utilized [6], as SNMP was standardized. The dynamic control plane allowed the network to react to a wider range of changes in the network automatically. It was much later in form

of the Network Configuration (NETCONF) protocol that this dynamic control plane was truly enabled. Eight years after SNMP was proposed, a new idea of decoupling the control plane from NFs altogether was proposed by the GSMP. However, the idea proposed by GSMP did not become widely used until it was proposed again by the OpenFlow protocol and the SDN paradigm.

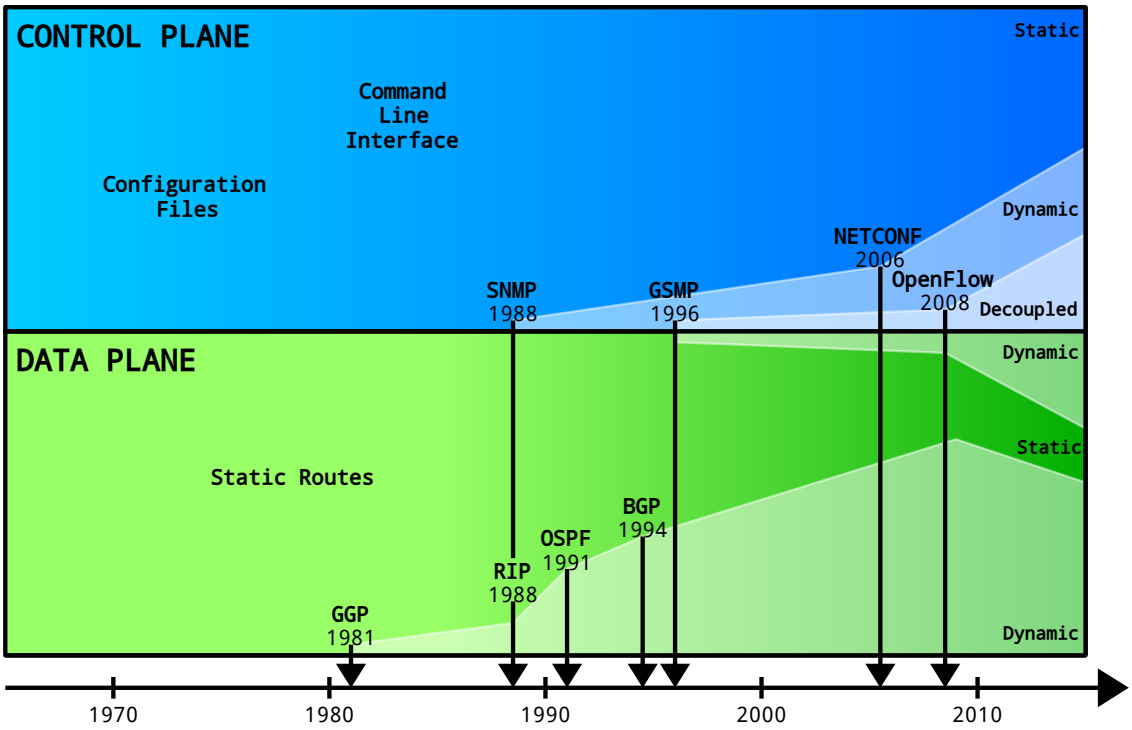

Figure 2.5: *Evolution of control- and data planes in IP networks.*

This section attempts to explain the main principles and features offered by the SDN paradigm and why they potentially can change how computer networks are perceived. Additionally, the main driver for full-blown SDN deployments, the OpenFlow protocol, is presented.

2.4.1 Principles and Interfaces

SDN enables new ways to control the network and can solve many use cases that are hard to solve without it. In order to define SDN more specifically Jarschel et al. [18] describe in their paper four principles of SDN. These principles are:

Separation of control- and data plane which means the ability for an external controller to change the forwarding behaviour directly. This allows the control- and data plane to evolve separately.

Logically centralized control which means that the controller needs to behave as centralized entity to utilize network wide information to make, for example, optimized forwarding decisions. This does not mean that the controller would have to be a single instance but it most likely should be a distributed system to tackle scalability.

Open interfaces which means that fundamental interfaces needed to control the NFs must be and remain open because otherwise there could not exist interoperability between implementations. This principle affects also controller to controller and application to controller APIs in addition to the controller to NF APIs.

Programmability which means the ability to treat the network as a single programmable entity. This principle is perhaps best described by the recent attempts to treat network as a network operating system[5] in which NFs are treated similarly like processors are treated in ordinary operating systems. Here the challenge is to find the right abstraction levels.

These principles must be followed in order to classify a solution as SDN. [18]

SDN defines four interfaces between network devices, namely north-, south-, east- and westbound APIs. One principle of SDN was the openness of these APIs so that components using any of them would be easy to replace with another implementation. The SDN interfaces can be seen in Figure 2.6. The southbound API is needed for the separation of control and data plane and thus defines the API between the controller and the NFs. The northbound API is needed to enable applications running on top of SDN to communicate with the controller. The westbound API describes controller to controller communication that is needed to multiple SDN control domains communicate with each other. The eastbound API is required to communicate with the legacy systems not operating under the SDN paradigm using protocols such as BGP. [18]

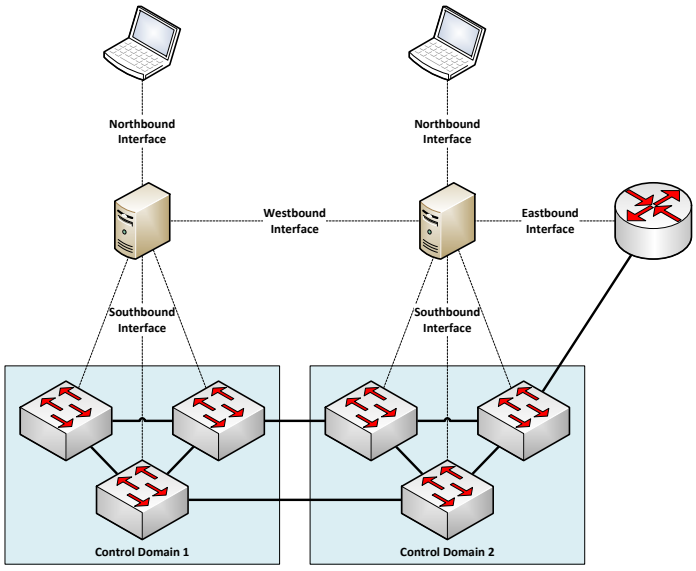

Figure 2.6: *SDN interfaces.*

[5]Open Network Operating System (ONOS). Available at: `http://onosproject.org/`. Accessed 16.03.2016

2.4.2 Benefits and Challenges

The SDN white paper [16] claims multiple benefits of SDN paradigm. These benefits can be summarised to the following seven items that potentially offer value when using SDN:

Vendor independence by using open interfaces the SDN solutions are able to control NFs from multiple vendors with one management solution.

Improved network utilization by using a centralized controller that has a complete view of the each network paths' utilization and has the ability to direct flows to the underutilized paths.

Reduced complexity through automation by using open interfaces to create tools that reduce operational overhead and reduce network instability.

Higher rate of innovation by allowing the network to be programmed in real-time to meet business requirements.

Increased network reliability and security by allowing consistent management and enforcement of configurations and policies from a centralized location.

More granular network control by allowing network operators to apply policies in different network layers easily.

Better user experience by presenting the network as a whole in a higher abstraction level to the SDN applications and users.

With these benefits, the SDN paradigm attempts to propose a future of open programmable networks that allow a rapid pace of innovation as the new norm for the currently rather static networks. [16]

While SDN offers many benefits, it does not come without problems that need to be addressed. Sezer et al. [15] identified four yet unsolved challenge areas that are faced with the rise of the SDN paradigm. Those identified challenge areas were:

Performance becomes challenging to maintain while trying to keep the flexibility to operate in a generic programmable way.

Scalability becomes a problem in controller design as the controllers need to be able to scale in all SDN interfaces efficiently without introducing high latencies to the network.

Security problems arise from the southbound protocols, if not mitigated with secure transport, and controller design that needs to be secured to allow shared network and infrastructure between various stakeholders.

Interoperability needs to be addressed to accommodate the transition from traditional networking to SDN since the transformation can not happen in one giant leap. This requires a way to operate parts of the network in the traditional way and part of it with SDN solutions.

However, these challenges are being addressed by research community and industry and it is likely that they will be solved in the coming years. [15]

2.4.3 OpenFlow

The OpenFlow protocol is a network management protocol designed especially for SDN. The OpenFlow standard is maintained by Open Networking Foundation (ONF). There are two important terms to understand in OpenFlow:

Flow is a one-way communication defined by layer 1-4 header information and metadata. The header information consists of physical layer interfaces, data link layer addresses, network layer addresses, transport layer ports and other header information.

Flow Table is a forwarding table used to define the actions to be taken for each specific flow when encountered.

The OpenFlow protocol allows control and data plane separation by creating mechanisms for an external controller to modify NFs' data plane. The version 1.0 of the OpenFlow protocol was specified in 2009 and it has been since actively developed further. [19]

The OpenFlow protocol allows the external controllers to install, modify and remove flows from the OpenFlow enabled NFs' flow tables. In addition, the OpenFlow protocol specifies mechanisms with which the devices can query the controller for decision for flows which do not have any rules in the flow table. In Table 2.2 can be seen a hypotetical example of a NF implementing the OpenFlow protocol with four installed flow rules in place. [19]

Table 2.2: *Example of OpenFlow flow table*

Src MAC	Dst MAC	Src IP	Dst IP	Src Port	Dst Port	...	Action	Count
00:...	ef:...	192....	172....	44677	80	...	→ *eth0*	13
ef:...	00:...	172....	192....	80	44677	...	→ *eth1*	755
00:...	ef:...	192....	172....	44678	443	...	*drop*	25
00:...	ef:...	192....	192....	*	*	...	*drop*	88
*	*	*	*	*	*	*	→ *controller*	3

The control offered by OpenFlow's flow tables is much more detailed than is possible to achieve with the traditional IP based routing. In IP based routing the flows between two hosts must use the same path in the network, but with OpenFlow the flows can each be assigned with unique path depending on the quality of service requirements of that particular flow. [16]

The OpenFlow protocol offers a low level network configuration solution that can be used to configure highly detailed forwarding actions on the network. NFs common to middleboxes that require more complicated logic such as traffic analysis must be implemented in the controller or in other non-OpenFlow devices since OpenFlow by design delegates such logic to an external entity. To network monitoring the OpenFlow protocol offers the controller a detailed view of the network state and statistics through built-in query mechanism. However, the more complicated monitoring logic must be implemented in the controller or in other non-OpenFlow devices.

2.5 Network Function Virtualization (NFV)

NFV proposes an idea to transform NFs into virtual NFs (VNFs) that utilize virtualization technologies to separate software from hardware. Virtualization helps to decouple NFs from physical location and share infrastructure resources more efficiently. Decoupling from hardware and moving to use virtualization on generic hardware enables VNFs to scale better and be more flexible. [20]

Current networks are usually comprised of NFs chained together in a certain way to achieve the desired network service. The chaining of NFs is usually static. NFV allows these NF chains to be reordered and moved around dynamically. This section covers the objectives of NFV and how they can be achieved. In addition, the requirements and challenges posed by NFV are explored. [20]

2.5.1 Objectives

The main objectives of NFV include cost-effectiveness, reduced power consumption, vendor-independence, flexible networks and rapid innovation. The objectives of NFV are achieved by virtualization of NFs to commercial-off-the-shelf (COTS) hardware. The cost-effectiveness and reduced power consumption stem from the fact that this COTS hardware can be shared efficiently between the NFs for example by using only one server during low peak hours and increasing the amount of active servers when the high peak hours begin as illustrated in Figure 2.7 and Figure 2.8. Vendor-dependence moves from hardware vendors to virtualization vendors such as companies developing hypervisors, but this does not the pose same level of dependence since software is usually easier to replace than hardware. Flexible networks realize when there are mechanism to configure the network dynamically so that NFs can be dynamically combined, chained and moved around. This is one of the biggest points of why SDN and NFV support each other rather than being competing technologies. Rapid innovation is enabled because the VNFs are all software so they can be more easily be modified and experimented with. [20]

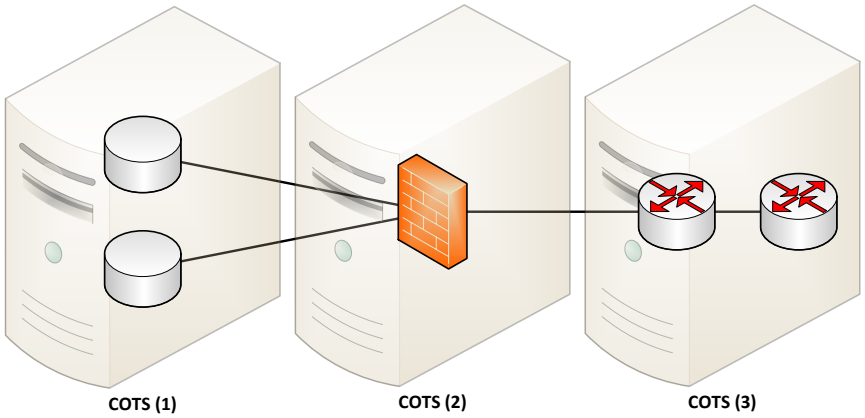

Figure 2.7: *NFV enabled service running in high peak hours.*

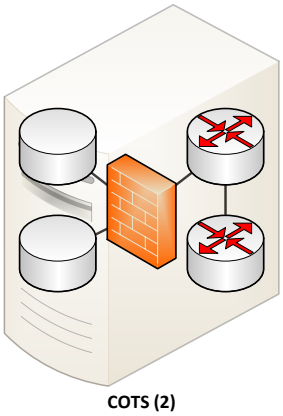

Figure 2.8: *NFV enabled service running in low peak hours.*

2.5.2 Requirements and Challenges

In recent article [21] Han et al. discuss the challenges and opportunities involved with NFV. In the article the authors categorise the non-functional requirements of NFV deployments to four topics.

Performance, which includes ensuring that throughput and latency requirements are met despite the virtualization.

Manageability, which is required as dynamic nature of NFV further complicates network management.

Reliability and Stability, where by using COTS hardware traditional SLA requirements can no longer be provided by special hardware but need to be handled by the NFV deployments to ensure the five-nines or better reliability.

Security, for which NFV brings additional security concerns due resources being in same hardware and increased system integration complexity, which need to be handled by the deployments.

Han et al. see the same benefits of NFV as they appear in ETSI White Paper [20], but they bring up few challenges to be solved before these benefits can fully realised. First challenge is to solve how network traffic can be dynamically redirected without service degregation for which SDN could provide solutions. Second challenge raised by the article is of how to provide the same level of service and performance as today with less reliable and less stable virtual machines. Third challenge raised is of how to manage and orchestrate VNFs so that the problems NFV is set to solve will not reoccur in NFV environments. [21]

3 YANG - DATA MODELLING LANGUAGE

Yet Another Next Generation (YANG) is a data modelling language developed and standardized by the IETF data modelling language workgroup (NETMOD) in year 2010 [22]. The current version of the language specification is 1.0, but there is an IETF draft being worked on for the version 1.1. YANG was initially developed for the needs of the NETCONF protocol to model its configuration and state data, remote procedure calls (RPCs) and notifications in an attempt to strike balance between high and low representations of the management data [22].

YANG closely relates to the data modelling language of NETCONF's predecessor SNMP, called SMIv2 (Structure of Management Information version 2). SMIv2 is used to define Management Information Base (MIB) modules which can in fact be translated to read-only YANG modules automatically, but will without further modifications miss some of the more advanced features offered by YANG. [22]

YANG is used to model NFs in a standardized way that can easily be converted to Extensible Markup Language (XML). Hence, YANG models serve as the Application Programming Interface (API) between the server implementing data access based on the YANG models and the client accessing this data.

This chapter deals with the concepts and features of YANG. The most common language constructs are presented and the benefits of the features are explored. Finally, the existing YANG models are discussed in detail.

3.1 Language Concepts

YANG defines the NF's configuration and state data in a hierarchical tree structure. YANG consists of a set of built-in keywords usually referred to as statements and types, which can be used to construct rich definitions of the data being modelled. In addition as YANG is a hierarchical language, it offers a similar hierarchical syntax as JavaScript Object Notation (JSON).

A YANG model contains a module that defines the hierarchical tree structure of the configuration and state data, the remote procedure calls (RPCs) arguments and the notification contents. The data described in YANG model can be identified with unique instance identifiers. In addition, all data that can contain a value is typed and describes its allowed values. This section aims to describe the basics of these different kind of concepts that appear in YANG models.

3.1.1 Module

YANG models define *modules* that have a predefined layout that they should follow. The layout starts with the module header that names the module and defines its *namespace*, *prefix*, dependencies, contact information, description and revisions.

After the header follow the type, data, RPC and notification definitions. An illustration of a generic YANG model's module layout is shown in Figure 3.1.

```
module example {
  namespace "urn:ietf:params:xml:ns:yang:example";
  prefix ex;

  /* ... imports, includes ... */

  /* ... organization, contact, description ... */

  /* ... revisions ... */

  /* ... type definitions ... */

  /* ... configuration data definitions ... */

  /* ... state data definitions ... */

  /* ... rpc definitions ... */

  /* ... notification definitions ... */
}
```

Figure 3.1: *The YANG model's module layout.*

YANG models are used to define one specific NF in a general way that includes the essential parts of the NF functionality so that basic deployment scenarios can work. Additional NF features are either tagged under some specific feature using *if-feature* statement that allows conditional data definitions or augmented to the YANG model from an another YANG model. Basic NF functionality could be all the MUST features defined in an RFC defining the NF or the smallest common denominator between multiple NF implementations. YANG models can be imported by other YANG models allowing reuse of existing data definitions. [22]

Namespaces are used to differentiate modules from each other so that naming collisions can be avoided even if parties developing different modules have no knowledge of each other. Closely related to namespaces are the prefixes that are used to simplify modellers' and reviewers' burden by shortening the namespace so it can be quickly and compactly referenced. In published RFCs the namespaces are always assigned by IANA. [22]

Revision tells the current version of the module so if the module changes for any reason it can be noticed by implementations using the YANG model and so be coped by changing the used module revision to a version that both parties support. Revisions enable backwards compatibility in implementations using YANG models. [22]

After the module's header follows the actual data definitions that is the core

of all YANG models. The common order observable in most standardized YANG models is to put first the type definitions then the configuration data definitions first, followed by the state data definitions after which come the RPCs and notifications. Usually there is only one or two top-level data definition *containers* per module, which contain the rest of the data definitions inside of them.

3.1.2 Data Definition

Data definition statements either contain other data definition statements or they contain at least a type definition statement and can thus hold a data value. Data definition statements always represent either configuration or state data, in YANG models the distinction is made with special statement called *config* within the data definition statement. Data definition statements can be one of the following:

leaf is a simplest data definition that is at most one instance and has no child data definition statements. *Leaves* contains data values.

leaf-list is otherwise the same as *leaf*, but can contain a list of unique *leaves*. In context of *leaf-list* a key refers to its value.

container is a data definition that is at most one instance and can hold no value, but can have one or more child data definition statements such as *leaves* or *containers*.

list is otherwise the same as *container*, but can contain a list of unique *containers* identified with one or more key *leaves*.

choice and cases are data definition statements that do not describe data but offer a conditional branches in the data structure. A *choice* is a condition that ensures that at most only one of *choice*'s *cases* can exist at any given time in the model. One of the most common uses of choice in current models is IPv4 and IPv6 separation.

As noted, these data definition statements can together be used to form complex tree-like data structures that closely resemble the data required to configure and monitor NFs. Each YANG model usually defines at least one top-level data definition that is located at the root of the data structure, however if desired they can define more of these top-level definitions. An example of *leaf* and *container* data definition statements can be seen in Figure 3.2, describing clock's state as its standardized in the YANG data model for system management [25]. [22]

```
container clock {
  description
    "Monitoring of the system date and time properties.";

  leaf current-datetime {
    type yang:date-and-time;
    description
      "The current system date and time.";
  }
...
```

Figure 3.2: *Data definition statements for clock state.*

Data definition statements can be used in multiple locations in the YANG model if they are grouped together with a *grouping* statement. This statement allows efficient reuse of definitions and thus decreases the modeller's burden and makes error more unlikely since a part of a tree used in multiple locations can be changed at once and the change affects all parts that used the structure. Inside the data structure a grouping can be referenced using the *uses* statement. [22]

3.1.3 Instance Identifier

Each data definition in YANG models has a unique instance identifier that can be used to refer to it. The identifier is constructed by reading the models from top-level containers towards the definition in question. The identifiers are YANG namespace aware and thus the namespaces must be included in the identifiers, which is usually done by using the prefixes to shorten the identifier. The data definitions statements *leaf-list* and *list* must be accessed using their keys or with positional index number if no keys are specified because otherwise there is no way to make the distinction between multiple values. [22]

For example Björklund [23] has defined a YANG data model for interface management that defines a three level data structure for basic interface configuration. In the model the top-level is **interfaces** *container* followed by **interface** *list* which contains multiple *leaves* of which the key is called **name** and the *leaf* describing the interface's power status is called **enabled**. An example instance identifier for interface's, named *eth0*, administrative status is shown in Figure 3.3.

```
/if:interface/if:interface['eth0']/if:enabled
```

Figure 3.3: *Instance identifier of an interface's administrative status.*

The instance identifier syntax closely resembles XML Path Language (XPath) expressions, which is intentional since many advanced features of YANG are enabled by XPath expressions. In fact instance identifiers are required to be valid XPath expressions. [22]

3.1.4 Type Definition

Each *leaf* and *leaf-list* data definition statement holds a mandatory type definition statement that defines the format that the data must be presented in to be considered valid. YANG offers some basic built-in types such as *string*, *enumeration* and *uint64* which can also be found in most generic programming languages. In addition, YANG also enables modellers to derive their own data types with type definition statements called *typedefs*. An example of a type definition statement defined in common YANG data types by Schönwälder [24] defines system's date and time as presented in Figure 3.4. The example shows how an existing type definition *string* can be extended to a new type that limits its allowed values with the *pattern* statement to a format such as *"2016-02-02T10:00:00.12345Z"*. [22]

```
typedef date-and-time {
  type string {
    pattern '\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}(\.\d+)?'
          + '(Z|[\+\-]\d{2}:\d{2})';
  }
...
```

Figure 3.4: *Type definition statement of date and time.*

There exists also special types other than those derived from the basic types, namely *leafref* and *identity*. A *leafref* type indicates that this *leaf* contains an instance identifier pointing to some other data node in the data structure. In turn an *identity* type defines a globally unique identity in the system that can be referenced with the *identityref* statement. These identities can be used to for example separate protocols such as UDP and TCP from each other. An *identity* definition can be used as a base identity for derived identities. Example of a base and a derived identity shown in Figure 3.5 was defined by Bierman and Björklund [25] in a YANG model for system management. [22]

```
identity authentication-method {
  description
    "Base identity for user authentication methods.";
}

identity local-users {
  base authentication-method;
  description
    "Indicates password-based authentication of locally
     configured users.";
}
```

Figure 3.5: *Identity statements defining user authentication methods.*

3.1.5 Remote Procedure Call (RPC) and Notification

While most of the YANG specification deals with the data and type definitions, there are also a way in YANG models to define RPCs and notifications. Both of these definitions closely relate to data definitions as they define the top-level objects that are used to contain related data definitions. The RPC definitions contain definitions for input and output and the notification definitions contain definitions for information content. [22]

RPC definitions allow the modeller to define procedures with specified input arguments and output results. RPCs defined by YANG models can thus be used to define a full application programming interface (API) that can be used easily over the network. In fact, all the operations of the NETCONF protocol are defined as RPC definitions in YANG models. [22]

Notification definitions can be used to define set of important events that are emitted by the NF. Notifications can be defined to hold even very complex information contents using the same rules as used with data definitions. Therefore, notifications can provide a valuable event-based interface to the NF's state.

3.2 Language Features

YANG offers few special features that distinguish it from a mere JSON document and enable it to describe efficiently network configurations. These features include extensive model validation information, standardized way to extend and modify the models and enforced backwards compatibility between model revisions.

In this section the main features offered by YANG models are presented. The first and most appealing feature is the possibility of automatic validation of all the data conforming to a YANG model. Other useful features being explored in this section include augmentation and deviation of YANG models and backwards compatibility offered by YANG models defined by following the rules specified in the YANG specification.

3.2.1 Validation

One of the most important features enabled automatically by YANG models is the possibility of extensive automatic validation. This is important because validating data content has proven to be a hard task. This claim is supported by the fact that data injection, due to insufficient validation, has been listed as the most common security vulnerability in OWASP's Top Ten project[1] listing web application security threats for almost a decade. The messages conforming to a specific YANG model can be validated syntactically and semantically to a great extent assuming these capabilities are used.

However, automatic validation offered by YANG is no silver bullet to all validation problems as it provides the essential validations from the NF's point of view but leaves the implementation specific restrictions open. For example if there is a string value for some NF attribute and no further restrictions required by the NF, there

[1]OWASP Top Ten Cheat Sheet. Available at: `https://www.owasp.org/index.php/OWASP_Top_Ten_Cheat_Sheet`. Accessed 13.02.2016

might be an implementation need to restrict that value further. This need might arise if the API to the NF is implemented using mechanisms that forbid some special characters appearing in the data thus requiring the characters to be escaped or removed from the data. In addition, there might be some operating system limits to some values like username length and other variables that are not NF's restrictions but are implementation's restrictions.

There are two types of validation offered by YANG models: syntax and semantic validation. Syntax validation is the monotonous part which ensures that only allowed byte sequences are found in the data, which is usually the action that is associated with the word *validation*. YANG models contain inherited syntax validation capabilities since all the nodes on the models are typed and structured. The structure and types allow for creation of validation code or schemas that can be used to automatically validate the messages conforming to YANG models. Semantic validation is a more complex validation type that can be used to describe dependencies between data definitions. YANG models can be added with statements *when* and *must* that can be used to describe conditional expressions that need to be evaluated to determine the semantic validity of the data. For example the *when* statement could be used to allow transport layer port definition to show only on protocols that support ports, such as TCP or UDP. The *must* statement could be used ensure that at least two interfaces exist when creating a router instance. An example of the use of semantic validation in YANG models can be seen in Figure 3.6 presented as an example by Björklund [23] in the RFC defining a YANG model for interface management.

One of the most widespread types of XML document validation is the use of XML Schema Definitions (XSD) that can be used by XML libraries to automatically validate data contents [26]. The other more advanced XML based approach is the use of Document Schema Definition Languages (DSDL) that includes standardized mechanisms for syntax and semantic validation [26]. However, YANG models are not constrained to just XML based validation and can be automatically converted to other schemas or to generic code that do the validation.

3.2.2 Augmentation

Standard YANG models should cover only the very basic NF capabilities, with some advanced features included with the *if-feature* statement. Augmentation is a YANG feature that can be used to extend existing YANG models with new data definitions. There are few reasons why augmentation might be used such as building another standard YANG model on top of an existing standard YANG model, adding optional and experimental NF functionalities or adding implementation and vendor specific functionalities not found in NF specification. [22]

Augmenting YANG models works by including the standard YANG model being augmented to a new YANG model that then uses the *augment* statement to add new data definitions to the standard YANG model. An example augmentation of the YANG model for interface management defined by Björklund [23] can be seen in Figure 3.6. The new YANG model formed this way can offer advanced functionalities if the configuring party knows how to make use of them, but in the same time can remain backwards compatible with the default model. Augmentation

has some limitations that require the YANG models to be augmented in a way that does not break the general YANG modelling rules. In addition, no mandatory data definitions can be augmented to existing data defintions, which also ensures the backwards compatibility. [22]

```
...
  augment "/if:interfaces/if:interface" {
    when "if:type = 'ianaift:l2vlan'";

    leaf base-interface {
      type if:interface-ref;
      must "/if:interfaces/if:interface[if:name = current()]"
         + "/vlan:vlan-tagging = 'true'" {
        description
          "The base interface must have VLAN tagging enabled.";
      }
    }
    leaf vlan-id {
      type uint16 {
        range "1..4094";
      }
      must "../base-interface" {
        description
          "If a vlan-id is defined, a base-interface must
           be specified.";
      }
    }
  }
...
```

Figure 3.6: *Augmenting VLAN model on top of interface model.*

3.2.3 Deviation

While default YANG models strive for the very basic protocol functionalities, sometimes there might be NF implementations that do not implement even all of these. And sometimes there might have been some lapse in the review of the model and some special optional functionalities have slipped into the model. In these cases existing models can be deviated from the published standard by, for example, forbidding some of the fields in the existing model.

Deviations are mostly used to document that this particular implementation misses some vital piece of protocol specification and thus functions only partially. However, deviations are strongly discouraged and should be used only as the last resort. Deviation statement can be one of *not-supported*, *add*, *replace* or *delete*. The *not-supported* statement just indicates that the given node that the deviation targets can not be configured on this NF. On the other hand, the *add* statement

adds new substatements as shown in Figure 3.7, *replace* statement modifies existing substatements and *delete* statement removes existing substatements. [22]

```
...
deviation /if:interfaces/if:interface {
    deviate add {
        max-elements 32;
    }
}
...
```

Figure 3.7: *Deviating interface model by limiting list's length.*

Despite that deviations are discouraged, they are the preferable way to communicate limitations to client implementations. In addition, if there is an operating system that would limit the available interface count up to 32, then the deviation shown in Figure 3.7 would be the best way to deal with the situation.

3.2.4 Backwards Compatibility

With revision numbers implementations interpreting YANG modelled messages can distinguish the supported revisions and adapt to the situation when they are not. YANG specification also describes update rules that must not be broken in order to ensure that the model remains backwards compatible with the old revisions. The rules defined by YANG specification dictate that any changes to published module should create new revision of the module, the module's name and namespace must not be changed, obsolete data definitions must not be removed and all the changes to the definitions must be backwards compatible. [22]

The backwards compatible changes to definitions include, but are not limited to:

- statements limiting the value space can be expanded but not reduced.
- *must* and *mandatory* statements can be removed.
- *default* statements can be added.
- new type definitions, RPCs and notifications can be added.
- new data definitions can be added as long as they do not add *mandatory* statements to existing data definitions unless they are made conditional with the *if-feature* statement.

With these rules it is ensured that YANG models can evolve over time without breaking existing applications. Naturally, there could be changes that can not follow these rules but those would require completely new YANG model and thus are not recommended. [22]

3.3 Existing Models

Currently there are only a few standardized YANG models. The standardized YANG models define the very basic *system* NF functionality, mentioned earlier as the functionalities that are taken for granted in any NF, and some of them define the NETCONF protocol. However, more YANG models are under development, some even in the late draft stages and should become standards during 2016.

The *system* functionality is not by itself enough to define any meaningful NF. Thus, the full potential of YANG as modelling language will not be realized until most of the common NFs currently found on the networks are standardized. Full potential can not be reached because different implementations do not yet share a common API for NF functionality and thus the workload to support different vendors remains high. Yet, even providing a formal API specification with YANG is a step forward compared to the proprietary single purpose APIs, since the implementations would at least get to utilize the common features offered by YANG.

In this section a small survey is made to the existing YANG models both standardized and ones in different draft stages. Since the standardized models are already complete they are first discussed with more detail, followed by a glimpse to the draft models.

3.3.1 Standard Models

The first standard YANG model was made to describe the common YANG data types in 2010 right after YANG specification [22] was standardized. That first YANG model was shortly followed by standardized YANG models for the NETCONF protocol. The first YANG model to describe actual NF functionality was not standardized until 2014 when the YANG model for interface management by Björklund [23] was standardized. Thereafter YANG models describing NF functionality have been standardized in a rather slow pace amounting to only four models.

Currently standardized YANG models describing NF functionality include:

A YANG Data Model for Interface Management which defines configuration definitions that can be used to modify interfaces and state definitions that can be used to access interfaces' status and statistics such as packet counts. [23]

A YANG Data Model for IP Management which defines configuration definitions that can be used modify interfaces' IPv4 and IPv6 parameters such as addresses and state definitions can be used to access interfaces' operational IP status. [27]

A YANG Data Model for System Management which defines configuration definitions for system information and users, NTP, DNS and Radius and state definitions for retrieving system information and time. In addition, this YANG model is the first YANG model not specific to the NETCONF protocol to include RPC definitions. [25]

A YANG Data Model for SNMP Configuration which defines extensive configuration definitions to configure SNMP parameters, but does not define any state definitions since those are expected to be converted from MIB modules.

In addition, this YANG model is the first YANG model to extensively utilize the YANG *submodules* that can be used to split a YANG module into multiple smaller modules. [28]

With these YANG models it is already possible to perform basic system configuration that could be used to configure systems with simple network configuration such as desktop computers. The standardized models still lack much of the models needed for middleboxes and routers that might contain tens or even hundreds of NFs.

3.3.2 Draft Models

The YANG models need to go through a review process that can take a long while before they become published standards. These YANG models that are being reviewed are called drafts and they can and do change based on the input of the YANG model reviewers. There exists three draft models that are very near of becoming standards:

A YANG Data Model for Network Access Control List (ACL) which will standardize the way how packet filters are configured. [29]

A YANG Data Model for Routing Management which will form a base configuration for routing. This base YANG model describes only static routing, but is expected to be extended by various routing protocols such as OSPF and BGP. [30]

A YANG Data Model for Syslog which will standardize the way how logging is configured. [31]

Other interesting YANG model drafts still in earlier stages of development include definitions for following network protocols: OSPF, BGP, IS-IS, IPsec, BFD, VRRP, DHCP, VLAN, IGMP and PIM. The mentioned YANG model drafts are modelled outside of the NETMOD workgroup in their own protocols' IETF workgroups. Currently there is no standardized YANG models made by the other IETF workgroups so it remains unclear how long it takes for a draft to reach standard status in those, but it could be predicted that at least some of the drafts will get standardized in 2017.

4 NETCONF - NETWORK CONFIGURATION PROTOCOL

The NETCONF protocol allows the NF to implement a formal API using YANG models that can be used to manage the NF's configuration data and to monitor NF's operational state. The NETCONF protocol uses a remote procedure call (RPC) paradigm built on top of an XML encoded message stream. The NETCONF protocol can be used to enable dynamic management of the network and it plays along with the new rising network paradigms such as SDN and NFV. Since the NETCONF protocol is XML based it can also be used to implement complex controller operations with relative ease by using XML transformation techniques like EXtensible Stylesheet Language Transformations (XSLT). [32]

NETCONF is often described as the successor of SNMP. This is because both protocols describe a holistic view of network management, where one can define standardized interfaces to configure and monitor network devices using a single protocol. Figure 4.1 shows a side-by-side comparison of SNMP and NETCONF in different levels. As definition language SNMP uses MIB which at high level is very similar to YANG used in NETCONF, both can be used to model data and its restrictions in detail. Both protocols also communicate with custom NF instrumentation that does the integration with the NF APIs and the actual NF implementations. For communicating with the NF instrumentation SNMP uses ASN.1 encoded values opposite to the UTF-8 encoded values used by NETCONF. Another difference is that NETCONF separates operations from messages whereas SNMP messages contain the requests as a value in the message header. NETCONF approach allows arbitrary operations defined by the definition language. Another difference in message level is that in SNMP requests and responses are unidirectional since SNMP usually utilizes the UDP transport protocol whereas NETCONF is connection oriented protocol allowing RPCs and their replies to function in bidirectional channel. As message encoding SNMP uses binary format and NETCONF uses XML. Monitoring events are semantically almost the same in SNMP Informs and NETCONF notifications. SNMP is usually configured to use UDP as transport mechanism but it can be configured to use TCP also. In addition, SNMP can have optional security added at the message layer using methods specified in the SNMPv3 standard. Whereas NETCONF can use SSH or TLS on top of a TCP connection to offer the transport layer protection. [32][10]

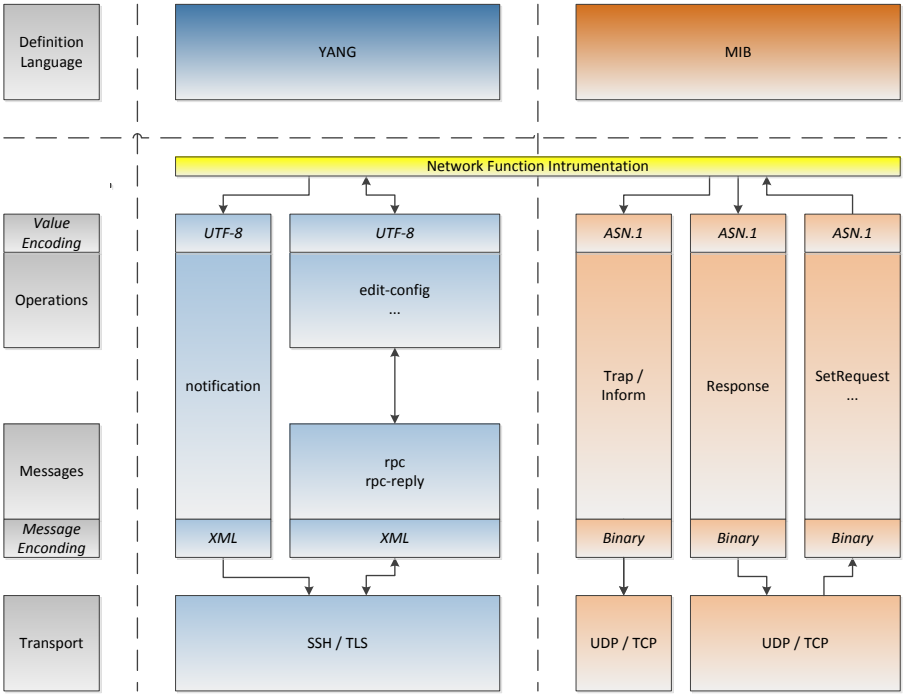

Figure 4.1: *Comparison of NETCONF and SNMP layers.*

This chapter goes briefly through the main concepts and features offered by the NETCONF protocol and then explains how the protocol can be used to configure and monitor NFs. In addition, the security aspects, transport and access control of the protocol are discussed in detail. Finally, a quick look is taken on the existing protocol implementations, which include both client and server implementations.

4.1 Protocol Concepts

The NETCONF protocol defines a network management protocol between a client, which in this case is the network management system or administrator of the system, and a device containing one or more NFs. These client and server start the protocol session by exchanging capabilities, which define what operations are available for use, in a *HELLO* message. The *HELLO* is the first NETCONF specific message sent after the secure transport layer is established. [32]

The NETCONF protocol defines RPCs, sent after the initial capability exchange, as operations. Almost all of the predefined operations target some datastore, which is a storage where NFs' configurations are stored, but developers can implement new operations that might not have anything to do with datastores. [32]

This section explains the main concepts related to the NETCONF protocol, starting from the NETCONF capabilities and protocol sessions. Another important concept discussed is the datastore, which is the backend where NETCONF servers store all the configuration data. In addition, the implications of the operations and notifications in NETCONF are explored.

4.1.1 Capabilities

The NETCONF protocol is designed to be highly extensible and to this end it supports in the initial exchange of *HELLO* messages the capabilities offered by both communicating parties. This exchange of capabilities permits the client to adjust its behaviour based on the supported functionality on the server. A capability in NETCONF is a threefold concept:

Capabilities are special namespaces assigned by IANA to indicate capabilities that can not be fully described using YANG alone. Usually there is no need to add new capabilities to NETCONF.

Models supported by an NF are also exchanged in the capability exchange so that client can learn the YANG models and their reversions implemented on the server.

Features supported by an NF are exchanged as parameters inside the supported models. [32]

The actual capabilities use their assigned namespace but supported modules and features use a special parameter syntax to encode the capability information. In Figure 4.2 can be seen a simple NETCONF HELLO message, which indicates support for protocol version 1.1, *candidate* capability and *ietf-system* module at revision 2014-08-06 with *authentication*, *local-user* and *ntp* features enabled.

```
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capabilities>
    <capability>
      urn:ietf:params:netconf:base:1.1
    </capability>
    <capability>
      urn:ietf:params:netconf:capability:candidate:1.0
    </capability>
    <capability>
      urn:ietf:params:xml:ns:yang:ietf-system?
      module=ietf-system&
      revision=2014-08-06&
      features=authentication,local-users,ntp
    </capability>
  </capabilities>
  <session-id>2</session-id>
</hello>
```

Figure 4.2: *Example of a HELLO message describing NFs' capabilities.*

4.1.2 Sessions

The NETCONF protocol is session-oriented and uses client-server architecture in which the server waits listening on an assigned port for clients to connect to it. When a client connects to the server, the server authenticates the client and the client authenticates the server thus creating a scheme called *mutual authentication*. Once the authentications are done a NETCONF session is started with the exchange of HELLO messages, the session receives its own unique session identification assigned by the server. The same client can connect to one server with multiple connections simultaneously and other clients can also connect to the server while there are active sessions already going on. [32]

The NETCONF specification is not bound to any specific transport protocol and can be used with multiple existing transport protocols [32]. The specification mandates that all NETCONF implementations must at least support the SSH protocol [33] as the transport protocol [32]. The NETCONF protocol utilizes the underlying transport protocol to a great extent and does not try to reinvent the wheel on that regard. In Figure 4.3 a typical message exchange leading to an established NETCONF session layered on top of SSH transport protocol is shown. As it can be seen from the figure the almost all of the messages are SSH messages in the protocol's different phases [33] (key exchange, service request, user authentication and channel creation) leaving the NETCONF protocol session creation to only a really simple XML encoded HELLO exchange.

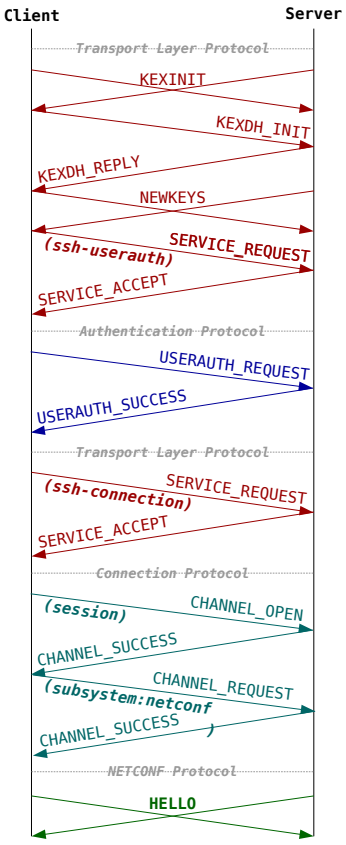

Figure 4.3: *Sequence diagram of SSH/NETCONF session initiation.*

Since there can be multiple open sessions to a server, but there is only one active configuration on the server at any given time, the sessions changing the configuration need to lock and unlock it in a way that the sessions' changes do not clash with each other. Sometimes it is also necessary to kill misbehaving sessions for which the NETCONF protocol offers a mechanism. If a session dies or is closed during a configuration operation the specification mandates that the changes are reverted so that the server is not left in unspecified state. [32]

4.1.3 Datastores

The NETCONF standard specifies datastore as a conceptual place that can be used to store and access the configuration and state data. The way how datastores are implemented is up to the implementation but they could for example be implemented using simple files or complex distributed databases. There are three types of datastores defined by the NETCONF protocol specification: *startup*, *running* and *candidate*, of which only *running* is mandatory to implement. The *startup* datastore is used to persistently store NFs' configuration data and it is copied as *running* datastore at startup. The *running* datastore reflects the configuration data currently in use. And the *candidate* datastore can be used as staging area for changes to be later pushed into the *running* datastore. [32]

The datastores hold the configuration data defined by YANG models, which serve as the specification of what kind of configuration is allowed to be stored in the datastores [32]. By supporting only the *running* datastore the NETCONF implementation will lack some of the more advanced features such as *validate* and *confirmed-commit* operations. Hence, the more advanced NETCONF implementations usually implement also the *startup* and *candidate* datastores.

The state data is held only in the *running* datastore since operational state indicates only the present and thus can not be stored the same way as configuration can. For that same reason, the state data can not be modified either. A hypothetical NETCONF implementation only having a *running* datastore with only state data accessible would be very close to the functionality SNMP is used for currently.

4.1.4 Operations

Operations in the NETCONF protocol are all defined as RPCs in the relevant YANG models. The YANG models also define the input arguments and output contents for each operation. All operations are XML encoded within *rpc* messages which are in fact the only messages allowed to be sent by clients in NETCONF sessions after the initial *HELLO* exchange. Since operations are RPCs they also have a result that the applications calling the RPC expect to receive. This result is sent back for each *rpc* message in a *rpc-reply* message usually containing an *ok* tag if everything went well or an *error* tag detailing the reason for the error if something went wrong. Figure 4.4 shows the basic message flow in NETCONF sessions, which is simply just sending RPCs and receiving replies to them. [34]

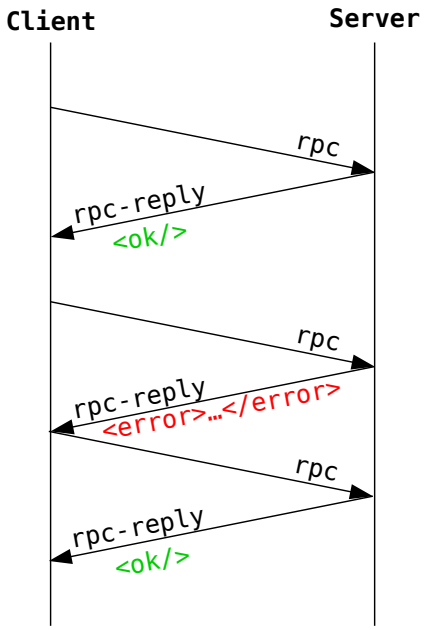

Figure 4.4: *Sequence diagram of typical client server interaction.*

By default the NETCONF protocol defines nine different operations, but few more are supported in implementations that implement some of the more advanced capabilities. The basic operations are:

get-config operation is used to query configuration data from NETCONF server. The query results can be filtered to include only essential parts of the configuration.

edit-config operation is used to create, update and delete NETCONF server's configuration data. The operation works with *running* and *candidate* datastores.

copy-config operation is used to copy configuration data from a datastore to another. The most common use case for this operation is to store *running* datastore to *startup* datastore.

delete-config operation is used to completely remove all configuration from a datastore. The operation works with *running* and *candidate* datastores.

lock operation is used to protect a datastore from other sessions' changes and allow contention free modifications to the configuration data.

unlock operation used to remove the acquired lock from a datastore.

get operation is used to query configuration and state data from NETCONF server. However, state data can be queried from *running* datastore.

close-session operation is used to end the current NETCONF session. This operation can be sent either by the NETCONF client or the NETCONF server.

kill-session operation is used to terminate targeted active NETCONF session. This can be used as administrative operation to clean idle sessions or to remove misbehaving sessions.

By supporting more advanced capabilities (*candidate*, *confirmed-commit* and *validate*) implementations can also support following operations:

commit operation is used to atomically commit *candidate* datastore's contents to *running* datastore making the candidate configuration active.

cancel-commit operation is used to revert pending confirmed-commit operation, which allows network wide commits to rollback if any errors occur on any of the NFs being configured.

validate operation is used to check configuration's validity without actually applying the configuration as the new operational configuration.

As noted, the YANG models, used to define NF functionalities, can also include RPC definitions that enable new operations on the NF. A NETCONF client can call these new operations just as it can call the default operations and expect the results in form specified by the YANG model. Examples of custom operations can be found in the YANG data model for system management [25], which defines three such operations called: *set-current-datetime*, *system-restart* and *system-shutdown*.

4.1.5 Notifications

Whilst the NETCONF protocol is mainly designed for configuration management, there is a powerful monitoring capability specified in the specification. This capability is called notifications which like operations are defined by the YANG models. Notifications allow NETCONF clients to subscribe to specified notifications and receive them when such event is generated. Using notifications allows a lightweight way to monitor changes on the NFs' state. An example of typical notification interaction between NETCONF client and server can be seen in Figure 4.5.

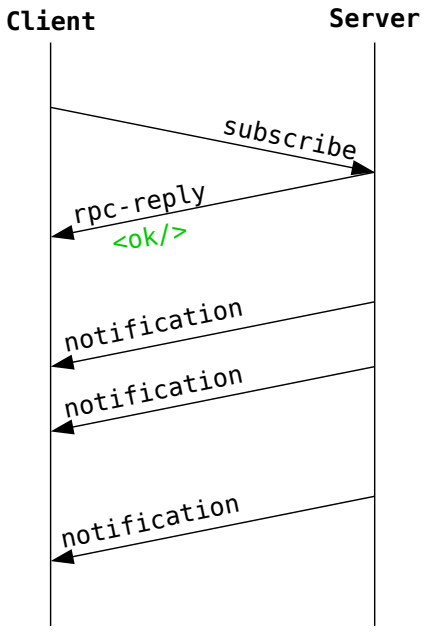

Figure 4.5: *Sequence diagram of typical notification interaction.*

Notifications can be received, depending on the server's and client's capabilities, interleaved with the other protocol operations or they need a dedicated session. A good analogy to NETCONF notifications is SNMP traps with the exception that a client does not receive all the notifications, but only those it subscribed for and is allowed (by the access control rules) to receive.

4.2 Network Management Features

The NETCONF operations and notifications are powerful tools for network configuration and monitoring. The operations open the NFs' active and pending configurations and operational state to be accessible through network on demand. The notifications enable a reactive management solutions that can react as NFs' operational state changes. Furthermore all this management information accessible as XML encoded content, thus allowing integration to a wide array of existing XML tools.

In this section a quick look is taken on how NETCONF can be used to manage NF configuration and how it can be used to monitor the NF state. When examining these features a comparison to the existing solutions is made to provide some insight why NETCONF was created.

4.2.1 Configuration Management

As the protocol's name indicates NETCONF was mainly designed for network configuration in mind. The protocol offers transactional way of handling changes in the network, for example administrator could configure ten NFs simultaneously and perform a delayed commit so that if even one of the commits fails or NF being configured loses the channel to controller administrator can rollback all the devices to the last working configuration. This allows so called atomic commits to the network.

Different ways to manage the NF's configuration are shown in Table 4.1. The simplest way is to use the *writable-running* capability which allows direct, instantaneous, editing of NF configuration much like is done usually when configuring NFs with CLI. Another option is to use the *candidate* capability which allows for changes to be made in a placeholder datastore called *candidate* and once all the changes are made, these changes can be committed to the *running* datastore much like is done when configuring NFs with configuration files. These two approaches are very much alike in achieved functionality and already with these capabilities NETCONF tells if the changes were applied successfully or offers a detailed error information if they were not. However, with *confirmed-commit* capability NETCONF offers a method to perform network wide transactions by conditionally committing changes to multiple NFs and only if all went through successfully acknowledging the commit and thus making the changes permanent, if a failure occurs, all the commits can be reverted and the network is left unchanged. The *confirmed-commit* capability also prevents sessions from locking themselves out due to a configuration error as the commit will not be acknowledged if the session can not connect to the NF, thus providing timed rollback on such occasions. [32]

Table 4.1: *Configuration Styles*

Capability	Operation	Affected Datastores
writable-running	lock	running
	edit-config	running
	unlock	running
(distinct-startup)	copy-config	running → startup
candidate	lock	candidate
	edit-config	candidate
(validate)	validate	candidate
	commit	candidate → running
	unlock	candidate
(distinct-startup)	copy-config	running → startup
confirmed-commit	lock	candidate
	edit-config	candidate
(validate)	validate	candidate
	*(confirmed-)*commit	candidate → running
	commit	candidate
	unlock	candidate
(distinct-startup)	copy-config	running → startup

Another feature of NETCONF found on almost all network management implementations is the ability to save the running configuration to remain across NF restarts. This ability is shown in Table 4.1 as the *distinct-startup* capability.

The NETCONF protocol differentiates configuration from state by design and this differentiation is modelled into the YANG models that form the NF's NETCONF API. The NETCONF server can be asked to hand in only configuration information with the *get-config* operation that returns only *config* nodes. And if both configuration and state is requested, the *get* operation can be used.

Although differentiating configuration and state allows already some reduction of the received data per request, the protocol also offers an extensive filtering mechanisms that can be used to further limit the received data per query. The two mechanisms are *subtree* and *xpath* filtering, although *xpath* filtering is enabled only if NF's NETCONF implementation supports the *xpath* capability. With these filtering mechanisms it is possible to limit the query results to only a subtrees that match the filter thus reducing the amount of transmitted data drastically.

4.2.2 State Monitoring

NETCONF can also be used to monitor the devices. This can be accomplished by polling the device state periodically or by subscribing to different notifications coming from the device. The polling of the device state with NETCONF is very much similar to SNMP queries. Consequently subscribing to notifications is very much similar to SNMP traps or IPFIX.

The polling of device state is done in NETCONF with the *get* operation that can be limited to specific data with the *subtree* or *xpath* filtering in the same way

as configuration can be filtered. Many YANG models describe data definition that contains the identifiers for NF's state data. An example of standardized state data can be found in a YANG data model for interface management [23] that defines interface's packet statistics that contain count for inbound and outbound octets, inbound and outbound unicast packets and so forth. [32]

The other method used to monitor NFs with NETCONF are notifications, which as said earlier resembles SNMP traps and IPFIX. Currently there are not many notification types defined, but like RPCs and data definitions, notifications can be added by any YANG model. Notifications are subject to the same access control rules as other NETCONF objects and they also contain a mechanism to filter the unwanted notifications and a mechanism to replay past notifications if the monitoring application was not connected when the notifications occurred. Notifications are considered much more resource efficient way of monitoring and polling the state using the *get* operation should be reserved for special cases if possible.

4.3 Security Measures

NF configuration usually holds information about sensitive things such as network topology and passwords. Since configuration information is very sensitive, it should be kept secure when transmitting it to the NFs. Hence, the NETCONF protocol's specification has large portions focusing on the security of the protocol and there are multiple RFCs discussing different security related aspects of the protocol. [32]

The YANG data model for system management [25] defined a standard way to handle users in NETCONF servers. With the module it is possible to modify NETCONF users and associate SSH keys and passwords with them. This is currently the most essential YANG model to implement in order to create interoperable and secure NETCONF servers.

In this section two security measures that are used to enhance the NETCONF protocol's security are presented. The first one is secure transport protocol which is mandatory in all NETCONF implementations. The second one is the NETCONF Access Control Model (NACM) that can be used to limit users' ability to read and modify the NF's configuration and state data.

4.3.1 Secure Transport

The NETCONF standard specifies a mandatory use of secure transportation mechanism that provides authentication, data integrity, confidentiality, and replay protection. There are currently two transport protocols with an active specification that can be used in NETCONF: SSH and TLS. Specifications for those protocols define that mutual authentication between the client and the server must be performed, which should thwart Man-in-the-Middle (MitM) attacks against the NETCONF sessions. [32]

The specification for SSH transport in NETCONF has an IANA assigned port 830 for the server to listen on for incoming connections. The incoming connections are processed as follows:

1. The client connects, using TCP, to the server's port 830.

2. The client invokes the SSH transport protocol to exchange the integrity and encryption keys for the connection.
3. The client invokes the SSH authentication protocol to authenticate the user.
4. The client invokes the SSH connection protocol to open an SSH session
5. The client starts the NETCONF subsystem.
6. The client and the server send the NETCONF HELLO messages.

An example of this connection sequence can be seen in Figure 4.3. In addition, it is also mandated that the client must verify the server's identity and vice-versa the server must verify the client's identity. [35]

The specification for TLS transport in NETCONF has an IANA assigned port 6513 for the server to listen on for incoming connections. The incoming connections are processed as follows:

1. The client connects, using TCP, to the server's port 6513.
2. The client initiates the TLS handshake protocol to exchange the integrity and encryption keys for the connection.
3. The client and the server authenticate each other's certificates and finish the TLS handshake protocol.
4. The client and the server send the NETCONF HELLO messages.

In addition, the NETCONF's TLS specification contains predefined rules that are used to derive the client's username from the certificate contents since the TLS protocol itself does not specify username concept. [36]

4.3.2 NETCONF Access Control Model (NACM)

In 2012 the NETCONF protocol gained an RFC describing a way for administrators to limit the data access on NFs, called NETCONF Access Control Model (NACM). The standard included guidelines for YANG model developers on how to embed the NACM controls into YANG models, description of the NACM functionality and a new YANG model describing how to modify the NACM access lists. NACM also specifies concept of groups to NETCONF user management, where user can belong to zero or more groups and these groups can be given access to different NETCONF resources. [34]

NACM can be used to limit access to NETCONF operations, datastores and notifications. The NACM limits NETCONF users' access using permissions. The limits on operations are enforced with the *execute* permission, the limits on datastores are enforced with the *create*, *read*, *update* and *delete* (CRUD) permissions and the limits on notifications are enforced with the *read* permission. NACM offers possibilities that are not available in any other network management protocol currently as there is no truly well defined way to limit authenticated users' rights in SNMP nor OpenFlow. [34]

As NACM's name indicates it operates on access lists by going through the lists from top to bottom trying to find matching rule that permits or denies the action being checked. In Figure 4.6 is shown a snippet of a possible access list that allows a user belonging to *observer* group to only retrieve interfaces' state information but deny all other access to the system.

```
<rule-list>
  <name>observer-acl</name>
  <group>observer</group>

  <rule>
    <name>permit-read-interface</name>
    <rule-type>data-node</rule-type>
    <module-name>ietf-interfaces</module-name>
    <path xmlns:if="urn:ietf:params:xml:ns:yang:ietf-interfaces">
      /if:interfaces-state/if:interface
    </path>
    <access-operations>read</access-operations>
    <action>permit</action>
    <comment>
        Allow read access to the interfaces' state.
    </comment>
  </rule>
  <rule>
    <name>permit-exec-get</name>
    <rule-type>protocol-operation</rule-type>
    <module-name>ietf-netconf</module-name>
    <rpc-name>get</rpc-name>
    <access-operations>exec</access-operations>
    <action>permit</action>
    <comment>
      Allow the statistics group to execute 'get' operation.
    </comment>
  </rule>
  <rule>
    <name>deny-all</name>
    <module-name>*</module-name>
    <access-operations>*</access-operations>
    <action>deny</action>
    <comment>
        Deny all other access.
    </comment>
  </rule>
</rule-list>
```

Figure 4.6: *Example of NACM rules defining statistics observer.*

4.4 Existing Implementations

Since the NETCONF protocol was first standardized in 2006 it is not surprising that there are already many implementations of the protocol. Since NETCONF is a client-server architecture protocol there are both client and server implementations.

In this section the existing client and server implementations are explored and some of their internal details explained. The popularity and maturity of existing implementations usually tells a tale of the acceptance a given protocol has received, for example if there was a great protocol with zero implementations then it probably is not very widely accepted and used protocol.

4.4.1 Client Implementations

NETCONF client implementations are required to handle the integration with the underlying transport protocol and provide a mechanism for the user of the implementation to input data to the implementation that is then packed into NETCONF *rpc* messages and to receive the result from the NETCONF *rpc-reply* messages as output. Use cases where NETCONF client implementation is needed are network management systems and testing applications.

There are various NETCONF client implementations varying from simple wrappers to a more complex validating implementations. Most notable standalone open source implementations are *ncclient*[1], *libnetconf*[2] and *JNC*[3]. These three implementations are examples of library implementations that provide an API for a network management system to use NETCONF, but do not do any content generation themselves. However, there exists also implementations inside SDN controllers that provide a southbound API implementation of NETCONF. The most prominent SDN controller implementations are *OpenDaylight*[4] and *ONOS*[5]. These two implementations are capable of, to some extent, automatically generating configuration based on the YANG models whilst the other implementations leave it up to the application developer to generate the contents of the NETCONF messages.

4.4.2 Server Implementations

NETCONF server implementations are required to listen on incoming connections, transform the received operations into actions against the internal datastores whilst validating and authorizing them and provide a way for the NF instrumentation to perform actions based on the received operations. Since NETCONF servers have to perform a lot more functionality than NETCONF clients, they tend to involve a lot more NETCONF specific code and are thus harder to implement.

From the currently available and active NETCONF server implementations a few are commercial and one is open source. All of the active implementations support all

[1]Python library for NETCONF clients. Available at: `http://ncclient.org/`. Accessed 16.03.2016

[2]C library for NETCONF. Available at: `https://github.com/CESNET/libnetconf`. Accessed 16.03.2016

[3]Java NETCONF Client. Available at: `https://github.com/tail-f-systems/JNC`. Accessed 16.03.2016

[4]SDN Platform. Available at: `https://www.opendaylight.org/`. Accessed 16.03.2016

[5]Network Operating System. Available at: `http://onosproject.org/`. Accessed 16.03.2016

the NETCONF operations defined in the protocol's RFC [32], but implement varying combinations of the extension RFCs defined for NETCONF. However, none of the servers come with large scale instrumentation for any existing operating systems, most likely due to the fact that current users of the NETCONF protocol do not lie in open source community, but rather in the operator and telecommunications sector. The most prominent of the commercial implementations are *confd*[6] and *netconfd-pro*[7]. In addition, the one active open source implementation is *netopeer*[8]. The integration with the NF instrumentation in *netconfd-pro* and *netopeer* relies on the generation of code stubs that the developer needs to fill in order to embedded his own logic into the server process. The integration in *confd* on the other hand provides a precompiled master agent to which application developers bind with custom TCP based protocol and subscribe to different events from the master agent. All of the servers are implemented using the C programming language.

[6]Management Agent for Networking Devices. Available at: `https://developer.cisco.com/site/confD/`. Accessed 16.03.2016

[7]Network Management Server Toolkit. Available at: `http://www.yumaworks.com/`. Accessed 16.03.2016

[8]NETCONF toolset. Available at: `https://github.com/cesnet/netopeer`. Accessed 16.03.2016

5 YANG MODEL DEVELOPMENT

YANG model development is a process that involves several different developer roles ranging from the people who create NF standards, through the people who utilize the standards to the people who create the NF instrumentation and implement the NF standards. These roles are namely modellers, reviewers, device developers and application developers [37].

Modellers are ideally the sort of people that understand the very internals of given NF such as protocol or service. Usually the same people who standardize protocols are in excellent position to define the YANG models when the standardization is complete as they possess in depth knowledge of the protocol and are not too focused on any specific implementation of that given protocol. Modeller should also take into account model extensibility and ensure that it can be adapted to possible future changes. [37]

Reviewers should also possess in-depth knowledge of the NF being modelled but do not require so much knowledge of YANG modelling itself, but rather evaluate the substance in the model and how well it fits its intended use case. Reviewers also assess the understandability of the model and thus ensure higher quality models. [37]

Device developers are the people who interpret the YANG models and implement the functionality described there into NF instrumentation. In other words, it is device developers' job to transform the YANG models into format understood by the NF implementations. [37]

Application developers are the people who gather information of existing YANG models in the network and build management applications utilizing these models to build business logic. A typical example of application developer could be a network management system developer that is implementing service provisioning and monitoring. [37]

In this chapter the focus is mainly on the modellers and the device developers. At first the concept of NF instrumentation is explained and the achievable integration levels with current NF implementations is explored. Then it is explored how a modeller's work flow goes and how that affects the device developers. And finally, a prototype YANG model for IPsec and IKEv2 configuration implemented for the purposes of this thesis is presented.

5.1 Network Function Instrumentation

New YANG models are defined and new NFs are implemented continuously. This leads to network devices that contain complex collections of NFs which might differ

from each other greatly. These YANG models and NF implementations need to be bound together by using something called *instrumentation*. [38]

In this section the concept of management daemon, that can be used to implement NF instrumentation, is explained. In addition, the levels that these management daemons can integrate the YANG models with the NF implementations are explored.

5.1.1 Management Daemon

A daemon is a system process that is started as background process to handle events occurring within the operating system. There are usually many daemons running simultaneously performing system tasks such as receiving and storing system logs and keeping system's time up to date. The origins of the word can be traced back to 1960 and the origin is explained by the following quote:

> *Our use of the word daemon was inspired by the Maxwell's daemon of physics and thermodynamics. (My background is Physics.) Maxwell's daemon was an imaginary agent which helped sort molecules of different speeds and worked tirelessly in the background. We fancifully began to use the word daemon to describe background processes which worked tirelessly to perform system chores.*[1]
>
> - Fernando J. Corbato

Therefore, in this thesis we fancifully use the term *management daemon* to describe a background process in the system that controls an NF and listens to configuration events from the management protocol implementation and converts those requests into actions targeting the running system. These configuration events could be requests to read the running system's state or requests to modify the configuration. In addition, management daemon can observe the NF that it controls for events that it can then notify to the management protocol implementation.

5.1.2 Levels of Integration

There are different ways for two programs to integrate together, be it through operating system signals and configuration files defined in some domain specific language (DSL) or some sort of an API. During the prototyping phase of this thesis it became apparent that there are at least three different levels of integration with configured programs that can be achieved with NETCONF and YANG based solution. These levels, ordered by efficiency from lowest to highest, are:

YANG-to-DSL in which the configuration data presented in YANG specified format is simply converted to the NF implementation's DSL by the management daemon reading the NETCONF datastore and then the NF implementation process is signaled to reread its configuration files.

YANG-to-API in which the management daemon reading the NETCONF transforms the YANG configuration data to dynamic API calls that provide an interface to the NF implementation being configured.

[1] The origin of the word daemon. Available at: `http://ei.cs.vt.edu/~history/Daemon.html`. Accessed 14.03.2016

YANG-as-DSL in which there is no management daemon in the middle, but rather the NF implementation itself subscribes to listen to configuration events from the NETCONF datastore and acts upon them directly.

Usually when developing NETCONF based management it is not possible to use the YANG-as-DSL approach. This is because the NF implementations being configured are usually external projects that have little or no care for dynamic network configuration and thus will not merge code for something they do not use themselves. This is especially problematic until a common API is found to interface the NETCONF daemon. One of the major findings of the implementation phase of this thesis was the applicability of AgentX protocol [38] for the use as inter-process communication (IPC) mechanism between the NETCONF server and management daemon. *Hence, slighly modified AgentX protocol could be a well suited candidate to offer a unified IPC API even between different NETCONF server implementations and management daemons.*

Therefore, for common development situations only the YANG-to-DSL and YANG-to-API approaches are applicable and from these two performance-wise the latter is preferable because it involves less abstraction layers. However, there are not so many NF implementations that offer a full API to configure the implementation so in some cases the YANG-to-DSL might be the only viable option. Yet, as for NF implementations that do offer a full API, the Linux kernel's Netlink API[2] is a fine example where YANG-to-API approach could be used in management daemon.

5.2 Instrumentation Development

Developing YANG models requires extensive knowledge of the NF being modelled in order to capture all the essential nuances and in order to make the model concise, simple and powerful. However, once a model is developed the review and implementation parts can be made easier with the right tooling.

In this section a de-facto standard tool for handling YANG models is discussed. The tool's internal workings are little bit opened and it is explained how this tool can help reviewing and understanding the YANG models and how it can be utilized to substantially ease the device- and application developers' work. In addition, an example of a typical network instrumentation development workflow is presented.

5.2.1 Development Tools

Currently the de-facto tool to use when modelling and reviewing YANG models and when developing software based on those models is a program called *pyang*[3]. Written with the Python programming language, the *pyang* is a small utility program that is a YANG model validator and language transpiler[4].

[2]Kernel to user communication channel. `http://www.linuxfoundation.org/collaborate/workgroups/networking/netlink`. Accessed 02.03.2016

[3]YANG validator and converter. Available at: `https://github.com/mbj4668/pyang`. Accessed 02.03.2016

[4]Transpiler is a source-to-source compiler that converts input written in one language to output generated as another language. For example this could mean converting YANG models to Java programming language.

The main use case of *pyang* for modellers and reviewers is its feature to validate YANG models according to the rules defined by YANG specification [22]. This ability allows users to quickly spot and fix errors in the YANG models under development and to validate that no backwards compatibility breakage occurs during revision bump. An example use of *pyang* in YANG model development can be seen in Figure 5.1 which shows a hypothetical revision upgrade to the YANG model for IP management [27] that removes enumeration from the model, thus breaking the backwards compatibility rules defined in RFC 6020 [22].

```
#> pyang --check-update-from=ietf-ip@2014.yang ietf-ip@2016.yang
ietf-ip@2016.yang:121: error: the enum 'dynamic', defined at
                       ietf-ip@2014.yang:121 is illegally removed
```

Figure 5.1: *YANG model validation using pyang.*

Another useful feature of *pyang* is the possibility to represent the YANG models in a much more concise, although less descriptive, tree-format that is easy to comprehend quickly. A slightly modified example output of *pyang* is presented in Figure 5.2 where a module is represented as the top-level element and then the YANG data definitions within are indented in the tree view with the *plus-minus-minus* notation. The letters *rw* indicate that the given data definition is readable and writable (configuration), whereas letters *ro* would indicate that the data definition is only readable (state). In addition, the keys of *list* data definitions are shown in brackets, *leaf* data definitions with the question mark are considered optional elements and *container* data definitions with the exclamation mark implicitly enable the given functionality if they exist in the datastore.

In addition, as YANG models define an API that model's users need to follow precisely, models can be also used to generate code stubs for the device developers to fill. Code generation can also be extended to generate the custom content validation code. Many NETCONF server implementations offer transpilers that turn YANG models into code stubs that are wired to the server implementation automatically leaving only the actual logic part for the device developer. In fact, also *pyang* can work effectively as a transpiler as shown later in this thesis.

5.2.2 Implementing YANG Models

Device developers implementing NF instrumentation using the YANG-to-DSL or YANG-to-API approaches need to read and comprehend the YANG model of the NF, then workout the method of how to interface with the NETCONF server implementation and finally convert the internal management daemon presentation to the format understood by the NF implementation. With YANG-as-DSL approach the management daemon is not needed and there is no need for conversion from internal presentation to any other format. These steps in case of YANG-to-DSL and YANG-to-API approaches are explored with an example of NF instrumentation presented in this section.

There exist IETF standardized YANG models on how to configure network device's interfaces and IP addresses, which are shown in Figure 5.2 and Figure 5.3

using the tree notation generated with the *pyang* utility program. (IPv6 parts not shown for brevity.) The interface YANG model provides a basic structure to name interfaces and control their power status [23]. The IP address YANG model extends the interface YANG model in a way which resembles the method which most operating systems use to handle IP addresses, where IP addresses are added to specific interfaces [27].

```
module: ietf-interfaces
   +--rw interfaces
      +--rw interface* [name]
         +--rw name
         +--rw description?
         +--rw type
         +--rw enabled?
         +--rw link-up-down-trap-enable?
```

Figure 5.2: *A YANG Model for Interface Management. [23]*

```
module: ietf-ip
augment /if:interfaces/if:interface
         +--rw ipv4!
            +--rw enabled?
            +--rw forwarding?
            +--rw mtu?
            +--rw address* [ip]
            |  +--rw ip
            |  +--rw (subnet)
            |     +--:(prefix-length)
            |     |  +--rw prefix-length?
            |     +--:(netmask)
            |        +--rw netmask?
            +--rw neighbor* [ip]
               +--rw ip
               +--rw link-layer-address
```

Figure 5.3: *A YANG Model for IP Management. [27]*

By using the API provided by these YANG models, a NETCONF client implementation can perform configuration operations that modify interface and IP address configuration in NFs. This kind of transaction would include an *edit-config* NETCONF operation that could for example be something like the operation shown in Figure 5.4, which modifies an interface called *FastEthernet 0/0* containing primary and secondary IP addresses to illustrate the *list* YANG data definition.

```
<rpc message-id="5" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <candidate/>
    </target>
    <config>
      <top>
        <interfaces
          xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces"
          xmlns:base="urn:ietf:params:xml:ns:netconf:base:1.0"
          xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">

          <interface base:operation="replace">
            <name>FastEthernet 0/0</name>
            <description>
              WAN Interface connecting to backbone.
            </description>
            <type>ianaift:ethernetCsmacd</type>
            <enabled>true</enabled>
            <ipv4
              xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">

              <enabled>true</enabled>
              <forwarding>true</forwarding>
              <mtu>1500</mtu>
              <address>
                <ip>10.0.1.10</ip>
                <prefix-length>24</prefix-length>
              </address>
              <address>
                <ip>10.0.1.11</ip>
                <prefix-length>24</prefix-length>
              </address>
            </ipv4>
          </interface>
        </interfaces>
      </top>
    </config>
  </edit-config>
</rpc>
```

Figure 5.4: *NETCONF edit-config RPC modifying configuration.*

By using these YANG models it is possible to configure all compliant NFs regardless of the vendor and version by leaving the responsibility of native representation to the NF instrumentation. For example, a management daemon configuring a Cisco

router using its CLI could produce commands such as presented in Figure 5.5 based on the NETCONF operation presented in Figure 5.4.

```
 > configure terminal
#> interface FastEthernet 0/0
#>  no shutdown
#>  description "WAN Interface connecting to backbone."
#>  mtu 1500
#>  ip address 10.0.1.10 255.255.255.0
#>  ip address 10.0.1.11 255.255.255.0
#> !
```

Figure 5.5: *NETCONF edit-config operation interpreted in Cisco CLI.*

That same NETCONF operation could be interpreted by the management daemon in Linux router using its CLI somewhat differently. For example the CLI commands could look something like in Figure 5.6.

```
#> ip link set eth0 up
#> # Linux has no 'description'-field for interfaces.
#> ip link set eth0 mtu 1500
#> ip address add 10.0.1.10/24 dev eth0
#> ip address add 10.0.1.11/24 dev eth0
```

Figure 5.6: *NETCONF edit-config operation interpreted in Linux CLI.*

On the other hand, as both of the previous CLI examples represented the YANG-to-DSL approach, a YANG-to-API approach is presented in Figure 5.7. The figure shows an example use of the Linux's Netlink API that can be used to directly communicate with the NF implementation within the kernel, thus skipping the unnecessary middle steps required by the CLI making this approach more efficient because there is no need to fork new processes.

```
from pyroute2 import IPRoute

ip = IPRoute()
if = ip.link_lookup(ifname="eth0")[0]
ip.link("set", index=if, state="up")
ip.link("set", index=if, mtu=1500)
ip.addr("add", index=if, address="10.0.0.10", mask=24)
ip.addr("add", index=if, address="10.0.0.11", mask=24)
```

Figure 5.7: *NETCONF edit-config operation using Linux Netlink API.*

5.3 Modelling Network Functions

For many NFs there exist already RFCs or drafts that define YANG models for the NFs' functionality, but there are also many NFs which have very immature or non-existent YANG models. For these kind of situations when there is no existing model but one is required, one needs to take on the role of a YANG modeller. Another scenario when modelling is required is a need to augment existing models with additional custom functionality. However, modelling YANG models is not quite straight forward and requires usually multiple iterations and review runs, for example the current RFC draft for a YANG model for routing management [30] has undergone over 20 iteration rounds.

In this chapter the basic workflow of creating YANG models is explored through the first hand experience of implementing an experimental YANG model for IPsec management. After the workflow study, the experimental YANG model's properties are discussed. Furthermore, a short comparison to the existing draft for IPsec YANG model is presented.

5.3.1 Modelling Workflow

The first approach to create a YANG model for an NF that has many existing implementations includes studying the configuration syntax of existing NF implementations. Deriving YANG models from the existing implementations is easy because the developers who created those configuration syntaxes have had some vision of how the NF should be configured and have unknowingly already done some ground work for the YANG modelling. Second approach would be to study the RFCs and from there generate a completely new interpretation of how the NF should be configured. The third approach is to study the possibly existing SNMP's MIB modules for that NF if they exist, which probably contain much of the necessary contents for a YANG model. Only the first approach was used in this thesis.

When studying the configuration syntax of existing NF implementations the objective is to find the least common denominator, a set of functionalities they all implement, which is usually not much larger set of mandatory functionalities than are defined as *MUST* in the RFCs defining the NF. When all the common functionalities are found, the remaining functionalities should be grouped to logical features that could later be conditionally enabled. These remaining functionalities are usually defined as *SHOULD* or *MAY* in the RFCs defining the NF. After the configuration syntaxes are analysed it is equally important to go over the RFCs defining the NFs in order to find out if all the mandatory functionalities are modelled and also separate the optional functionalities that can later be gated behind *if-feature* statement in YANG model so that more advanced implementations can offer the features without requiring non-standard augmentations.

One more thing to remember when developing YANG models is that it is very tempting to create a YANG model very close to some specific implementation, making it easy to use YANG in that particular implementation, but at the same time seriously undermining the other implementations that have configuration models vastly differing from that one specific implementation. Therefore, it is vital that the developers of different NF implementations would contribute to the modelling

or review process. However, if it is not possible to form this sort of collaboration, then the review process should be even more careful and thorough in order not to incorporate any implementation specific gimmicks into the YANG models.

5.3.2 A YANG Model for IPsec Management

In this thesis an experimental YANG model for IPsec management, which is included in this thesis as appendix A, was created and the approach of studying existing configuration syntaxes and deriving the YANG model from those was used. The IKE implementations studied were *OpenIKED*[5], *StongSwan*[6] and a proprietary IPsec implementation. A sample of *OpenIKED* configuration can be seen in Figure 5.8, which configures an IPsec tunnel between two hosts. The same configuration sample with *StongSwan* can be seen in Figure 5.9. All the implementations supported IKEv2 so the experimental YANG model was restricted to that version of the protocol.

```
ikev2 "student_connection" esp \
       from 10.1.1.0/24 to 10.2.2.0/24 \
       local 10.0.1.1 peer 10.0.2.1 \
       dstid "C=FI,O=TTY,CN=Student Stud"
```

Figure 5.8: *Example of OpenIKED configuration.*

```
conn student_connection
       leftsubnet=10.1.1.0/24
       left=10.0.1.1
       rightsubnet=10.2.2.0/24
       right=10.0.2.1
       rightid="C=FI,O=TTY,CN=Student Stud"
```

Figure 5.9: *Example of StrongSwan configuration.*

When trying to find the least common denominator between the implementations it became easy to notice functionalities that were missing from some implementations and thus those could be moved under an *if-feature* statement. Ultimately all the implementations boiled down to few concepts: policies and rules. Rules defined a ordered list of access control rules that would link to policies, which defined the local and peer identities and security parameters. A tree view of part of the created YANG model can be seen in Figure 5.10, which shows these common functionalities shared by the implementations. Unfortunately the YANG model structure was mostly dominated by the configuration syntax of the proprietary IPsec implementation, which makes the model harder to integrate with the other implementations.

[5]IKEv2 implementation. Available at: `http://www.openiked.org`. Accessed 13.02.2016
[6]IPsec-based VPN Solution. Available at: `https://www.strongswan.org`. Accessed 13.02.2016

```
module: ipsec
   +--rw ipsec
      ...
      +--rw dynamic-policies
      |  +--rw dynamic-policy* [name]
      |     +--rw name              string
      |     ...
      |     +--rw (address-family)?
      |     |  +--:(ipv4)
      |     |     +--rw ipv4
      |     |        +--rw local-ip
      |     |        +--rw remote-ip?
      |     ...
      |     +--rw local-id
      |     |  +--rw (identity)?
      |     |     ...
      |     |     +--:(fqdn)
      |     |        +--rw fqdn?
      |     +--rw remote-id
      |        +--rw (identity)?
      |           ...
      |           +--:(fqdn)
      |              +--rw fqdn?
      |
      ...
      +--rw rules
         +--rw rule* [name]
            +--rw name
            ...
            +--rw (address-family)?
               +--:(ipv4)
                  +--rw ipv4
                     +--rw sources
                     |  +--rw source* [ip]
                     |     +--rw protocol?
                     |     +--rw ip
                     |     +--rw port?
                     +--rw destinations
                        +--rw destination* [ip]
                           +--rw protocol?
                           +--rw ip
                           +--rw port?
...
```

Figure 5.10: *Snippet of A YANG Model for IPsec Management.*

As noted earlier, there exists a draft YANG model in the IETF's IPsec workgroup that also defines a YANG model for IPsec management [39]. The draft model is a much more comprehensive YANG model containing also IKEv1, state data definitions and notifications. Also, the draft seems to be modelled using the second modelling approach of creating a configuration model from scratch based on the RFCs. Whilst the draft is more comprehensive than the model presented in this thesis, it is currently at very early stages and it is not possible to create functional NF instrumentation based on the model yet. However, the YANG model presented in this thesis should be considered as a slightly naive, albeit working, attempt to model IKEv2 configuration and the further development of the IETF draft should be fully endorsed. As a further research goal it would be interesting to attempt to implement an NF instrumentation based on the IETF draft and that way review and provide feedback for the draft model to develop it further.

6 PROTOTYPE NETCONF SERVER

Whilst there are many implementations of NETCONF servers, they are all implemented in C programming language, which is reasonable because usually the server needs to operate on resource constrained environments, but also requires a thorough understanding of the C internals and prudent software development practices to get the implementations secure. Some implementations also create a very tight coupling with the input, processing and NF instrumentation by compiling the NF instrumentation into the server logic. This approach makes the servers vulnerable for software errors that might occur on the NF instrumentation and forces all the code to run with the same privileges which means in any Unix-like operating system almost certainly the need to run the whole monolithic implementation as the administrator user.

Partly to address these security and reliability issues and partly to get a deeper understanding on what goes into a NETCONF server implementation, a new prototype implementation was created as part of this thesis. The implementation utilizes a modular event-driven asynchronous design, follows multiple security principles and was implemented using the Python programming language. The prototype did not include all the parts presented in the architecture, but it contained enough logic to provide a testbed to convert the southbound NETCONF messages into internal representation and to be passed onto the management daemons that converted those into NF configuration files. The prototype is specific to Unix-like operating systems in design and thus can not easily be converted to other types of operating systems. The prototype architecture was greatly inspired by the process architecture used in OpenBSD system daemons such as *OpenBGPD*[1] and *OpenNTPD*[2].

In this chapter the software architecture of the prototype implementation is discussed, with focus being on the process interaction and standard conformance. The modularity of the prototype is discussed in its own section and the AgentX protocol's suitability to this task is explained. In addition, the security aspects of the architecture are presented to provide view on how NETCONF server implementations can be further secured atop of the protocol's security measures.

6.1 Software Architecture

The existing NETCONF server implementations have some documentation about the server architecture, but they contain very little information about the design decisions that resulted the architecture. One of the major focus areas of this thesis implementation was to gather experience and knowledge of achievable modularity and security aspects of a NETCONF server. This knowledge can help to reason for

[1]BGP routing daemon. Available at: `http://www.openbgpd.org/`. Accessed 10.03.2016
[2]NTP server daemon. Available at: `http://www.openntpd.org/`. Accessed 10.03.2016

and against some of the features offered by NETCONF.

In this section an overview of the architecture is given by presenting a diagram showing the interaction between different components and explaining each component's purpose. Furthermore, the prototype's standard conformance resulting from the made design choices is reasoned about.

6.1.1 Architecture Overview

The NETCONF server prototype created in this thesis is split into multiple components running as separate processes in order to better confine privileges required by the different parts of the system. The prototype architecture contains six components that are required to implement the NETCONF server functionality. These components are:

parent, which is a simple process used to manage the other components (except *control*). It is required to run as administrator user to be able to make system wide modifications.

control, which is a command line utility to interact with the *parent* process in order control the NETCONF server. It can be used for example to restart the NETCONF server.

server, which performs the user authentication and spawns new *subsystems* for each NETCONF session triggered by the transport protocol and passes the received data to them.

subsystem, which performs the parsing of data coming from the *server* and converts it into messages towards the *engine*. The most NETCONF capability and XML handling logic is confined within this component.

engine, which is the component that contains the bulk of the NETCONF logic as it holds the system datastores and performs actions on them based on messages received from *subsystems*, performs the NACM checks and triggers events based on datastore changes to the *daemons*.

daemon, more precisely management daemon, which is the component that converts the messages from the *engine* into the NF instrumentation. There can be multiple of different *daemons* that can run as separate users and can be limited in access the same way as *subsystems*.

An overview of the architecture is shown in Figure 6.1. The inter-process communication (IPC) utilizes three different approaches which are signals, custom IPC[3] and a slightly modified AgentX protocol [38]. The signals are used between components that require very little interaction with the parent (*server* and *daemons*). The custom IPC is used over Unix-sockets between components that need to exchange simple one-way messages between each other (*control*, *parent* and *engine*). The AgentX protocol is used over Unix-sockets between components that require transaction support (*subsystems*, *engine* and *daemons*).

[3] A IPC framework called *imsg* borrowed from the OpenBSD daemons.

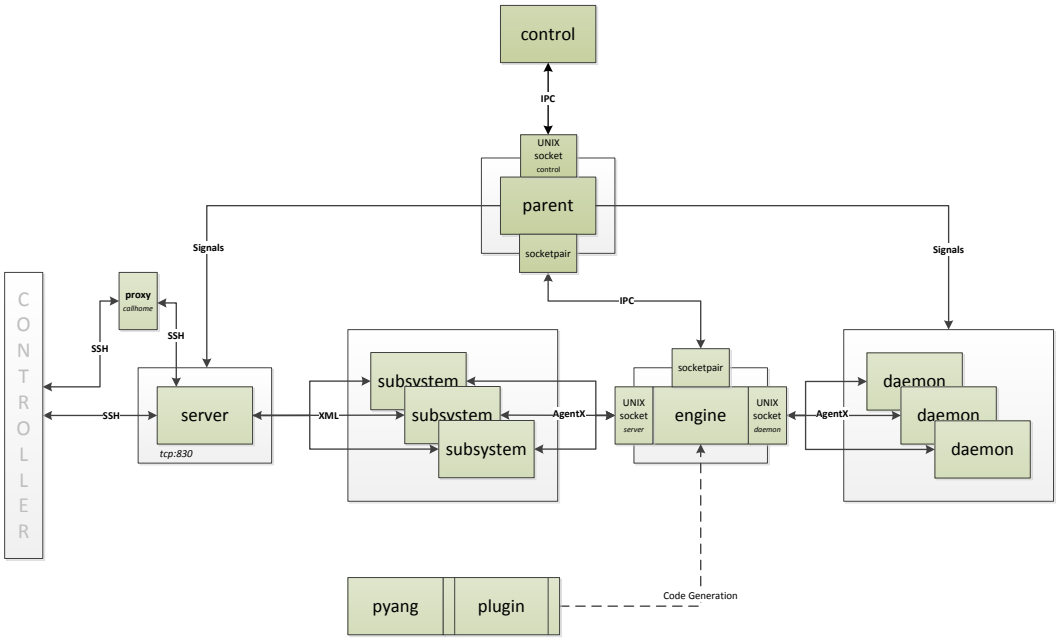

Figure 6.1: *An overview of the prototype architecture.*

The prototype attempts to make the API between *subsystems* and *engine* as simple as possible. With this architecture it is possible to move most of the functionality, that depends on capabilities supported by NETCONF clients, into the confined *subsystem* component that can be extensively limited in functionality and is always access controlled by the *engine*. Most of the capabilities and features offered in NETCONF do not need to complicate the design of the engine which can be made entirely by the rules of the most complex configuration. Furthermore, the prototype does away with XML as quickly as it can since, while XML is a good format to serialize data, it is unnecessarily complex for the server's internal use and deemed too large attack surface to use in the privileged components of the server.

In addition to the runtime components described earlier, the prototype used also the popular open source YANG model validator and transpiler *pyang* to transpile YANG models into Python source code with a custom plugin integrated into the *pyang* plugin-framework. This plugin allowed for automatic code generation for the syntax and semantic validation that is required to ensure the integrity of the datastores within the *engine* component.

6.1.2 Standard Conformance

The prototype strived for a high standard conformance with the *xpath* and *url* capabilities being the only ones not supported, since they were deemed too large attack surface to the protocol compared to the functionalities they bring. The *xpath* capability was dropped due to the fact that XML processing is limited to the *subsystem* component and would have been challenging and error prone to implement without

XML support within the *engine*. In addition, the *xpath* capability was evaluated to offer only minor benefits compared to the *subtree* filtering mechanism. The *url* capability was dropped because whilst storing configurations to arbitrary local files might have been useful, it was deemed not to be the NETCONF server's responsibility to connect and store configurations over the network, since this functionality could easily be achieved by the client.

The support for main features are listed in Table 6.1 which lists the most prominent capabilities and features currently standardized for NETCONF. The TLS transport functionality was not pursued with this implementation. However, the architecture does not limit the possibility of adding TLS transport functionality later on since even the standard emphasizes transport layer agnosticism.

Table 6.1: *Prototype features and capabilities*

Feature / Capability	**Supported?**
writeable-running	yes
candidate	yes
confirmed-commit	yes
rollback-on-error	yes
validate	yes
distinct-startup	yes
url	no
xpath	no
notification	yes
with-defaults	yes
partial-lock	no
callhome	yes
tls	no
ssh	yes

6.2 Modular Design

Since the NF instrumentation is split to multiple different components, it is safer to test and modify individual components without impact on the other components. Another good thing about modular design is that defects in the management daemons, made by device developers, which contain complex NF specific logic do not bring down the whole system, but are rather isolated to that given management daemon.

In this section the most fundamental design decisions that enable the modularity of the system are discussed. In order to convert the variable sized XML tags to a fixed internal representation a new identification scheme was developed. In addition, an existing IPC protocol called AgentX was found to be a suitable IPC mechanism for NETCONF servers. Furthermore, a technique used in the *subsystem* component to perform operations in an uniform way towards the *engine* component regardless of the supported capabilities of the NETCONF client is explained.

6.2.1 Object Identifiers and Keys

In the attempt to get rid of the XML as early as possible a new convention to identify the operations, notifications and data nodes was needed. In the prototype a new method of converting the XML namespaces and tags into 32-bit object identification (OID) values was created. The method used in OID calculation is fairly simple: pick a random key at build time that using a predefined hash function generates a unique value for every data definition in the YANG model and if a collision occurs pick a new key and start over. Using this method of calculating OIDs as shown in Figure 6.2, it is possible to refer to any data definition in the structure uniquely as 32-bit number that can be used very efficiently in hash table lookups. This method has one drawback as it limits the amount of possible data definitions[4] because collisions can become so probable that the build times become unbearable. However, since YANG models tend to contain less than 100 data definitions each this is not deemed a significant problem.

```
hash = hmac.new(key=b'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx')
hash.update(b'urn:ietf:params:xml:ns:yang:ietf-interfaces')
hash.update(b'interfaces')
hash.update(b'urn:ietf:params:xml:ns:yang:ietf-interfaces')
hash.update(b'interface')
oid = hash.digest()[:4]
```

Figure 6.2: *Object Identifier calculation for /if:interfaces/if:interface*

However, since YANG supports *lists* and *leaf-lists* that are identified by their keys another mechanism is also required to distinct those values from each other. For that purpose the following method was used: pick a random key at runtime and use that to hash the key values into a 128-bit value that uniquely identifies a single key value. This approach allows for fixed length key identifiers with the very slight risk[5] of possible collisions. The mechanism is shown in Figure 6.3.

```
hash = hmac.new(key=b'yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy')
hash.update(b'eth0')
key = hash.digest()[:16]
```

Figure 6.3: *Key calculation for /if:interfaces/if:interface['eth0']*

By combining the key and OID values it is possible to uniquely identify any instance of any data definition in the datastore with a 20 byte long identifier. Yet,

[4]The probability of randomly generating k integers with value less than N without collisions is $1-e^{\frac{k(k-1)}{2N}}$, which yields probability of about 68.78% when using 32-bit OIDs in NF instrumentation with hundred thousand data definitions.

[5]The probability of randomly generating k integers with value less than N without collisions is $1-e^{\frac{k(k-1)}{2N}}$, which yields probability of about 0.15% when using 128-bit keys in list with quintillion (10^{18}) different values.

it should be noted that depending on the depth of the datastore tree structure, one needs to concatenate these 20 byte identifiers for each layer. So for example a data node in depth of 8 would need 160 bytes to identify its location in the structure, but now with fixed length it is possible for an implementation to define maximum supported depth of datastores so that for example 200 bytes is reserved for the identifier and datastore can thus be no deeper than 10 layers.

6.2.2 AgentX Protocol

A little surprisingly during the implementation of the prototype NETCONF server it was observed that accompanying IPC protocol for SNMP, AgentX protocol [38], was almost a prefect match for the needs of NETCONF server. The AgentX protocol was standardized in 1998 [40] and later updated in 2000 [38]. The RFC was developed by Daniele et al. [38] to facilitate extensible design where subagents, equal to the *daemons* in the prototype architecture, that can dynamically extend the master agent, equal to the *engine* in the prototype architecture. The motivation for AgentX protocol is to provide a common API between the agents and thus alleviate NF vendors' problems of supporting multiple different platforms [38].

When using the AgentX protocol the *engine* process sends and receives the NETCONF protocol messages but has no access to the NF instrumentation directly. The access to NF instrumentation is provided by zero or more *daemon* processes, which do not see any of the NETCONF protocol messages, but interact with the *engine* process using the AgentX protocol. The internal use of AgentX protocol is not seen by the NETCONF clients in any way. [38]

The NETCONF protocol operations map quite nicely to the AgentX protocol PDUs when used between the *engine* and the *daemon* components. Table 6.2 shows the mappings where the component which initiates an operation is seen in the leftmost column and the PDUs that this operation would translate to is seen in the rightmost column. The mappings leave open the mechanism how AgentX can be used to execute custom RPCs.

Table 6.2: *Daemon mappings between NETCONF and AgentX*

Initiator	NETCONF Operation	AgentX PDU
engine	get	Get
engine	validate	TestSet, CleanupSet
engine	edit-config	TestSet
engine	*(confirmed-)*commit	CommitSet
engine	commit	CleanupSet
engine	cancel-commit	UndoSet, CleanupSet
engine	*custom-rpcs*	-
daemon	notification	Notify

Perhaps more surprisingly the NETCONF protocol operations also map to the AgentX protocol PDUs when used between the *subsystem* and *engine* components. Table 6.3 shows the mappings where the component which initiates an operation is seen in the leftmost column and the PDUs that this operation would translate to is

seen in the rightmost column. However, using AgentX protocol in this context would require also a mechanism to implicate a start and end of NETCONF operation, since the *engine* component requires that information in order to perform NACM checks. The mappings leave open the mechanism how AgentX can be used to execute custom RPCs.

Table 6.3: *Subsystem mappings between NETCONF and AgentX*

Initiator	**NETCONF Operation**	**AgentX PDU**
subsystem	*open-session*	Open
subsystem	*start-operation*	-
subsystem	get	Get
subsystem	get-config	Get
subsystem	validate	TestSet, CleanupSet
subsystem	edit-config	TestSet
subsystem	delete-config	TestSet, CommitSet, CleanupSet
subsystem	*(confirmed-)*commit	CommitSet
subsystem	commit	CleanupSet
subsystem	cancel-commit	UndoSet, CleanupSet
subsystem	subscribe	Register
subsystem	unsubscribe	Unregister
subsystem	lock	-
subsystem	unlock	-
subsystem	kill-session	-
subsystem	copy-config	-
subsystem	*custom-rpcs*	-
subsystem	*end-operation*	-
subsystem	close-session	Close
engine	notification	Notify

Clearly the AgentX protocol could be quite easily converted for the needs of NETCONF protocol and this conversion would most likely lower the barrier needed to implement NETCONF based services, because existing NF instrumentation implementations could be ported to use the new management protocol more easily. Notable additions needed for the protocol would be the ability to handle dynamic object identifiers and the ability to relay remote procedure calls to the *daemons*. The current prototype architecture requires the *engine* component to be aware of all YANG models, but as further research goal it would be interesting to see if the architecture could be modified to allow a YANG model agnostic approach as intended by the AgentX protocol specification.

6.2.3 Capability Unification

In order to simplify the *engine* component's design, the *subsystem* component unifies the operations received from NETCONF clients to a single common way of handling transactions; this process is defined here as *capability unification*. The capability

unification basically means that all NETCONF operations are performed in the same way as their most complex counterpart.

The most complex example of the capability unification is the unification of *edit-config* operation defined by the *writeable-running* capability, which is so primitive operation that it requires five extra steps internally to match the most complex *edit-config* operation defined by the *confirmed-commit* capability. The *edit-config* operation in this mode is converted internally to the five operations shown in Table 6.4, where the operation internally first calls for lock on the candidate datastore, then edits the locked datastore, performs confirmed-commit, finalizes the commit, copies the datastore as the startup datastore and finally unlocks the candidate datastore. However, if the client advertises the *startup* capability then the *copy-config* operation will not be performed and the responsibility is left to the client.

Table 6.4: *Unification of writable-running capability*

External Operations	External Target	Internal Operations	Internal Target
edit-config	running	lock	candidate
-	-	edit-config	candidate
-	-	*(confirmed-)*commit	candidate → running
-	-	commit	candidate
-	-	copy-config	running → startup
-	-	unlock	candidate

Another example of slightly less complex unification procedure is given in Table 6.5, where the *edit-config* operation defined by the *candidate* capability is slightly extended to include the *confirmed-commit* and the *copy-config* operations internally. In this case it is the *commit* operation that is expanded into multiple operations. However, if the client advertises the *startup* capability then the *copy-config* operation will not be performed and the responsibility is left to the client.

Table 6.5: *Unification of candidate capability*

External Operations	External Target	Internal Operations	Internal Target
lock	candidate	lock	candidate
edit-config	candidate	edit-config	candidate
commit	candidate	*(confirmed-)*commit	candidate → running
-	-	commit	candidate
-	-	copy-config	running → startup
unlock	candidate	unlock	candidate

In addition to capability unification, some operations can be reduced to a minimal functionality from *engine* component's point of view. One such operation is *get-config*, which can contain multiple *subtree* filters but the *subsystem* component can

convert each filter into an individual query as shown in Table 6.6 and concatenate the results together without requiring the *engine* component to know anything about the performed filtering.

Table 6.6: *Splitting of subtree-filters into multiple operations*

External Operations	External Target	Internal Operations	Internal Target
get-config *filter1* *filter2* *filter3*	running	get-config *filter1*	running
		get-config *filter2*	running
		get-config *filter3*	running

6.3 Security Measures

Since the management API is arguably the most critical part of the system in terms of security, much care must be taken to ensure that it can not be exploited. The prototype implementation created in this thesis explores various possibilities to limit the impact that even an authenticated user can achieve on the system. It is prevalent for a NETCONF server to prove robust and secure approach against attacks performed by authenticated users since it offers the NACM functionality that can be potentially used to provide a limited management API to the same system by multiple users of different organizations in case there is a need to share resources.

The security measures envisioned for the prototype architecture include extensive privilege separation, establishment of resource limits and process sandboxing. These are additions on top of the security features offered by the NETCONF standard and do not affect the standard compliance of the implementation. Although, there are many security measures in place, they naturally can not mitigate all possible security problems such as logic errors in NF instrumentation, but they can greatly reduce the attack surface exposed by the NETCONF server and provide security in different layers that augment the security provided by the NETCONF protocol itself.

6.3.1 Privilege Separation

In 2003 Provos et al. [41] presented a concept called privilege separation that is a mitigation technique designed to prevent security vulnerabilities from being abused for privilege escalation. Privilege escalation means the process where user running with limited privileges can abuse a vulnerability to gain more privileges, which in the worst case would be gaining the administrator privileges to the operating system [41]. This limits the risk imposed by security vulnerabilities so that even if an attacker gets full control of a vulnerable process, the attacker is limited by the privileges of the user that the process is running as [41]. Privilege separation requires the software to be designed to run as multiple processes with different privileges separated by

internal trust boundaries. Privilege separation has proven to be effective design in order to mitigate security vulnerabilities [41].

In the prototype implementation the *subsystem*, *engine* and *daemon* processes are designed to utilize privilege separation and all are intended to be running as separate users with very limited access to the underlying system. Whereas the *parent* process runs as administrator and handles operations that need administrator privileges such as modifying the NETCONF user credentials. For example the XML parsing is done by the *subsystem* processes running as an ordinary user on the system and if there would be a vulnerability in this XML parsing giving the attacker full control of the *subsystem* process it would still be running as an ordinary user with very limited access to the system. This confinement can be further enhanced by the sandboxing techniques discussed in later in this section.

Privilege separation in the case of NETCONF server implementation requires also a method of passing the user credentials on to the *engine* process, since it performs the NACM checks. In the prototype implementation the authentication between processes is achieved with a function called *getpeereid*, which is possible to implement on most Unix-like operating systems. The function basically operates by asking from a Unix-socket for the credentials of the program on the other end of the socket. Since Unix-sockets are local sockets, operating system is able to provide the effective user identity of the program. Thus, in the *engine* process we can trust what the operating system tells as the user information of the *subsystem* and *daemon* processes.

6.3.2 Resource Limits

Operating systems have multiple different resources available for the processes running on top of them. These resources include processors, memory, network interfaces and hard drives to name few. Every process on the system share these resources and thus they are a potential attack vector for denial of service attacks. One common attack against systems is resource exhaustion that can be caused by very trivially[6] if there is no mechanism to limit the impact.

In the prototype implementation the processes handling user data are all designed to be run as separate users. Therefore, further restrictions can be placed upon these processes in terms of resource limits. Users on Unix-like operating systems can be limited to certain amount of CPU, amount of memory, number of processes and other resources and since the prototype is designed to run as separate users these can be enforced quite strictly without impact on overall performance. Another complementary approach to limiting resources is to limit the allowed connections of a given user in the *server* process. In addition to these measures, the NETCONF implementation should have a configurable upper limit for YANG list and leaf-list expressions since these could be otherwise used to employ denial of service attacks by malicious users.

[6] One example is the fork bomb attack which creates infinite amount of new processes until the operating system runs out of resources.

6.3.3 Restricted Operation

Traditional model of software development has been to give all processes running with the same privileges the same set of available operations. However, usually processes do not need to perform most of the operations available to them by default. Restricted operation refers to the mitigation technique that limits the available operations of the program to the limited set defined by the program's developer. Restricted operation mode is available on operating systems such as Linux and OpenBSD. Linux offers a *seccomp-bpf*[7] and OpenBSD *pledge*[8] mechanisms which can be used to limit the system calls that a process can call. An example of *seccomp-bpf* code can be seen in Figure 6.4, which limits the process that runs it to be able to only perform input and output operations and stop execution, any other system call would lead to the process termination. An example of *pledge* can be seen in Figure 6.5, where the process is limited to similar subset of operations as with the *seccomp-bpf* example.

```
from seccomp import *

def apply_seccomp(server_sock, engine_sock):
    filter = SyscallFilter(defaction=seccomp.KILL)
    filter.add_rule(ALLOW, "read", Arg(0, EQ, server_sock))
    filter.add_rule(ALLOW, "write", Arg(0, EQ, server_sock))
    filter.add_rule(ALLOW, "read", Arg(0, EQ, engine_sock))
    filter.add_rule(ALLOW, "write", Arg(0, EQ, engine_sock))
    ...
    filter.add_rule(ALLOW, "rt_sigreturn")
    filter.load()

 ...
 apply_seccomp(server_sock, engine_sock)
```

Figure 6.4: *Restricting allowed operations in Linux*

```
from pledge import pledge

if not pledge("stdio"):
    raise SecurityException("Unable to pledge() the process.")
```

Figure 6.5: *Restricting allowed operations in OpenBSD*

[7]SECure COMPuting with filters. Available at: `https://www.kernel.org/doc/Documentation/prctl/seccomp_filter.txt`. Accessed 10.03.2016

[8]System call to restrict system operations. Available at: `http://man.openbsd.org/OpenBSD-current/man2/pledge.2`. Accessed 10.03.2016

The extensive use of privilege separation in the prototype implementation allows very fine-grained restrictions of the available system calls to the different components. For example the *subsystem* component only needs to read and write from its inputs to its outputs and perform some internal logic, but it requires no network access and no filesystem access, hence it could restrict itself to only those operations by using the same *pledge* call as show in Figure 6.5. After restricting the operations the *subsystem* component will not be able to access the filesystem, open network connections nor call other executables to name few restrictions. These restrictions further mitigate the possible security vulnerabilities that might exist in the *subsystem* component which holds the potentially error-prone XML processing and capability unification logic.

7 ANALYSIS OF YANG AND NETCONF

So far this thesis has focused on the different aspects of network management and how YANG and NETCONF perform the tasks related to it. In addition, a lot of implementation specific insights were explored in order to better understand and validate the claims driving the NETCONF protocol. Now on this chapter the intention is to gather the information written in the previous chapters and use that to analyse the network management capabilities offered by YANG and NETCONF. The intention of this analysis is to provide feedback whether YANG and NETCONF would be viable options to consider when implementing network management solutions.

This chapter starts with an comparison of the YANG and NETCONF based network management solution to the network management solutions presented in the second chapter in regards of the new rising network management paradigms NFV and SDN. In addition, the YANG and NETCONF based management solution is evaluated using a SWOT analysis based on the information gathered in the previous chapters in combination with insights from other research articles concerning the same subject.

7.1 Comparing Network Management Solutions

All of the network management solutions described in this thesis offer a different set of network management capabilities with their own strengths and weaknesses. These capabilities can be compared with each other to get an understanding what aspects of network management each of them try to achieve and how well they achieve those aspects.

In this section a simple comparison of the techniques that compete in the network management space is drawn. The comparisons are carried out in the scope of *enabling NFV* and *enabling SDN*. Enabling NFV describes how well given network management solution meets the management requirements of NFV (scalability, manageability, reliability and security). Enabling SDN describes how well given network management solution achieves the principles set by the SDN paradigm (separation of control- and data plane, logically centralized control, open interfaces, programmability). The scale of the comparison ranges from being completely agnostic to the paradigm to fully enabling the paradigm. Furthermore, the comparison is split to two parts where the first part covers the network configuration capabilities and the second part covers the network monitoring capabilities.

7.1.1 Network Configuration Capabilities

The possibility to dynamically configure the network is required to support the NFV and SDN paradigms. Some network configuration solutions presented in this thesis are not capable of delivering to the dynamic needs of these new network

paradigms, whereas some solutions just lack the adoption. In addition, aspects such as scalability, security and openness of the configuration interfaces matter for these paradigms and can be used to compare the solutions accordingly.

In Figure 7.1 the traditional CLI and configuration, SNMP, OpenFlow and NETCONF based network configuration approaches are scattered across a diagram that attempts to describe their orientation in respect to the new emerging network management paradigms SDN and NFV.

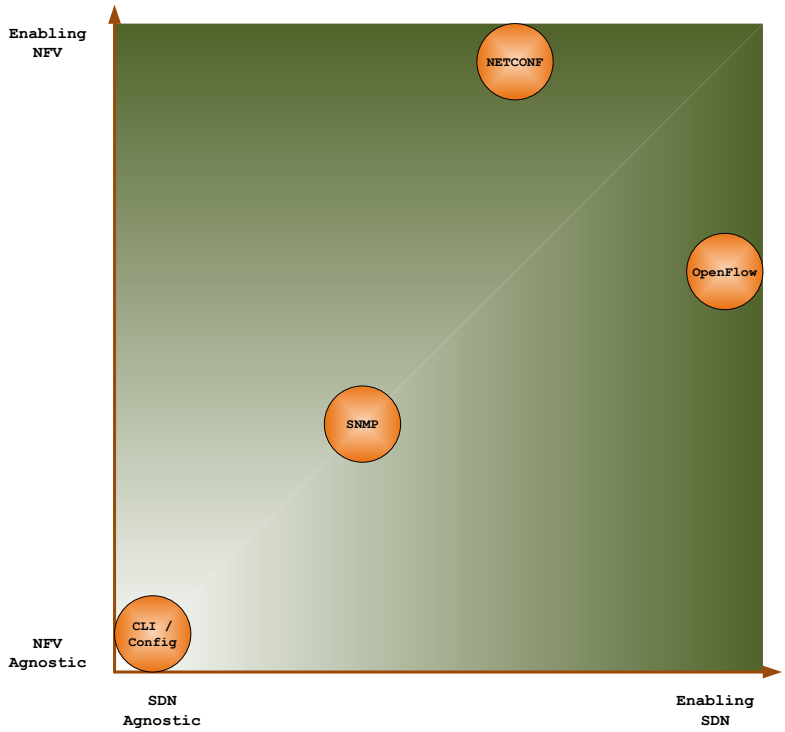

Figure 7.1: *Configuration capabilities regarding NFV and SDN*

The network configuration approaches' positions in the diagram can be explained as follows:

CLI / Config - The traditional network configuration approaches such as CLIs and configuration files are pretty much agnostic of the NFV and SDN. These approaches do not per se impede the new network paradigms, but they do not enable them either.

SNMP - SNMP offers some possibilities of open and interoperable interface for both SDN and NFV, but SNMP lacks the configuration models and has no serious push by the industry to become a standard way of configuring NFs. Hence, SNMP based network configuration is not very prominent enabler of these paradigms.

OpenFlow - OpenFlow is especially designed SDN in mind and basically is the de-facto low-level SDN management protocol, but pure SDN enabled networks

have yet to be realized and it is not even clear if everything happening in the network can be made into a clear cut of flows and controller logic thus leaving a gap in network configuration regarding the NFV.

NETCONF - NETCONF can be seen as an enabler for NFV as it is mainly focused on NF configuration. Despite NETCONF does not embrace the SDN's principle of strict data and control plane separation, NETCONF can be considered to enable SDN because it opens up the NFs' configuration to the controller in an open and centrally controllable and programmable way.

7.1.2 Monitoring Capabilities

The possibility to actively monitor the network is required to support the NFV and SDN paradigms. Some network monitoring solutions presented in this thesis are not capable of delivering to the needs of these new network management paradigms. In addition, aspects such as scalability, security and openness of the monitoring interfaces matter for these paradigms and can be used to compare the solutions accordingly.

In Figure 7.2 the logging, IPFIX, NETCONF, SNMP and OpenFlow based network monitoring approaches are scattered across a diagram that attempts to describe their orientation in respect to the new emerging network management paradigms SDN and NFV.

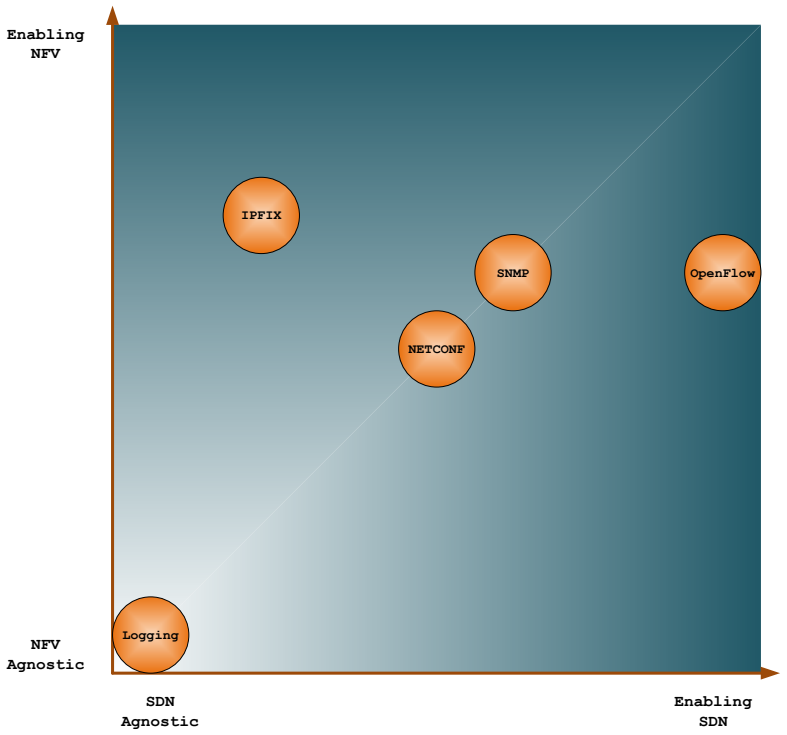

Figure 7.2: *Monitoring capabilities regarding NFV and SDN*

The network monitoring approaches' positions in the diagram can be explained as follows:

Logging - The use of ordinary logging of NF activity is very agnostic of SDN and NFV and does not really enable nor impede these technologies and serves more of as status quo than an enabler.

IPFIX - IPFIX is a very lightweight protocol and can be expanded to do traffic monitoring in a scalable way, which serves as an enabler for NFV. Furthermore, IPFIX does share the flow based view of the network that is also proposed by SDN, thus it provides monitoring mechanisms that can be used in SDN environments.

NETCONF - NETCONF has many useful features for monitoring NFs that enable both NFV and SDN and could maybe used in conjunction with other protocols to create even more effective monitoring capabilities. However, used alone NETCONF is quite resource hungry approach for monitoring, which is usually aspired to be pretty simple lightweight process.

SNMP - SNMP is currently the de-facto monitoring protocol and it is quite a bit more lightweight than NETCONF whilst offering most of the same monitoring capabilities and hence it is slightly better at enabling these new technologies.

OpenFlow - OpenFlow is designed with SDN in mind and largely moves the monitoring into the control plane and thus enables extensive monitoring capabilities for SDN, but as said earlier does not take into account other NF uses cases so well that it would enable all kinds of NFV scenarios.

7.2 SWOT Analysis

SWOT analysis is a method to evaluate the strengths, weaknesses, opportunities and threats of a solution in terms of the intended objective. The strengths describe the advantages of the solution over other alternatives. The weaknesses describe the disadvantages of the solution compared to other alternatives. The opportunities describe the changes in the problem space that could promote the solution. The threats describe the changes in the problem space that could hinder the solution.

The solution being evaluated in this section is the subject of this thesis, namely: the YANG and NETCONF based network management solution. And the objective is to determine whether this solution offers an efficient, secure and reliable way to dynamically manage networks. These two form together the question that this evaluation tries to answer and that question is:

> *Is the network management solution offered by YANG and NETCONF an efficient, secure and reliable way to dynamically manage NFs?*

7.2.1 Strengths

The strengths of YANG and NETCONF based network management solutions are the features and capabilities that offer a distinct advantage by using this solution compared to some of the other solutions. These strengths are illustrated at various chapters of this thesis and on research papers concerning YANG and NETCONF:

Standardized open interface - NETCONF and YANG are standardized solutions that have well defined specifications and are widely understood. Furthermore, they provide the open interface that is considered vital for new emerging network technologies such as SDN and NFV. Compared to custom management solutions such as CLI this is a great advantage as the development and testing tools can be shared and interoperability can be reached more easily. The capability to create interoperable APIs between different hardware and software vendors opens up possibilities to operate highly complex networks with just one or few solutions, which in turn greatly reduces costs and effort required to create network management systems.

Extendibility and programmability - There are already many viable YANG models that can be used to implement and configure NFs' basic networking functionality in a completely implementation independent way and creation of new and augmenting of existing YANG models is possible as shown in this thesis. In addition, the NETCONF protocol and YANG models are very extensible, allowing standardized and custom implementations to coexist within the same framework. Furthermore, NETCONF offers an RPC based management interface that can be easily used programmatically.

Security and reliability - One thing that makes NETCONF stand out amongst network management solutions is its many built-in security mechanisms. NETCONF offers a very comprehensive toolkit to limit NETCONF users' access and modification rights using the NACM functionality. In addition, NETCONF mandates transport layer security, which makes it secure by default, which is important considering the criticality of management interface. Furthermore, the security measures presented in this thesis' prototype NETCONF implementation show that the protocol design allows for even further security enhancements to be built into the implementation. And on top of these security measures, NETCONF is a connection oriented protocol with documented and concise error reporting API that allows a very reliable management interface even when errors occur, which is harder to achieve on network management approaches such as SNMP.

Full management solution - While NETCONF is prominent in network configuration it also incorporates many network monitoring capabilities that can be utilized to extensively monitor NFs. All these network management capabilities can be accessed using a single session, keeping the resources needed to manage NFs relatively low. This means that it is possible to achieve full network management using only NETCONF, which is not possible with most of the other presented network management solutions.

Performance and scalability - Hedstrom et al. [42] compared NETCONF's and SNMP's transaction performance and found out that with small number of managed NFs SNMP performed more efficiently, but as the networks grew larger NETCONF scaled better. Also, da Paz Ferraz Santos et al. [43] compared NETCONF and SNMP performance and their results indicate much better scalability of NETCONF compared to SNMP as the network sizes grow.

In addition, using the YANG-to-API and YANG-as-DSL approaches presented in this thesis when integrating to NF instrumentation can greatly decrease the overall time it takes to configure and deploy NFs compared to the manual approaches.

Integration to existing systems - Unlike full-blown SDN solutions like OpenFlow, NETCONF offers an easy way to integrate with existing systems with the YANG-to-DSL and YANG-to-API approaches presented in this thesis, thus offering a solution that can be used to transition smoothly to the new network management paradigms whilst still using the old battle-proven NF implementations.

7.2.2 Weaknesses

The weaknesses of YANG and NETCONF based network management solutions are the functionalities that leave the solution in a disadvantage compared to some of the other solutions. These weaknesses are illustrated at various chapters of this thesis and on research papers concerning YANG and NETCONF:

Verbose - One of the problems in NETCONF is the fact that it is by design very verbose because it uses XML as its serialization method. Whilst XML is proven technology with lots of merits, a study by da Paz Ferraz Santos et al. [43] has shown that NETCONF consumes even up to ten times more bandwidth compared to SNMP in certain scenarios. This problem can somewhat be alleviated by compressing the NETCONF messages using the transportation layer compression mechanisms. It has been studied by Augeri et al. [44] that XML is highly compressible language that can usually be compressed to around tenth of its original size. These results hold especially true for YANG-based XML that is mainly just very structured textual configuration. Nevertheless, with or without compression NETCONF usually consumes higher amount of bandwidth when compared to alternatives such as SNMP or IPFIX.

Adoption - The current challenge faced by NETCONF is its lack of adoption in open source development. There is only one open source implementation actively developed and its use seems to be low. Also the lack of standardized YANG models for the most used NFs hinder the adoption of NETCONF, but this problem might be alleviated if the YANG models currently in draft stages will be standardized in the near future. Alternatives such as OpenFlow and SNMP have much wider array of implementations available to use.

Efficiency - Since NETCONF was designed to offer full network management capabilities it has had to make compromises in terms of efficiency. This has mainly made NETCONF's monitoring capabilities quite resource consuming compared to solutions that are mainly concerned with network monitoring such as IPFIX. And due to the same reason some of the network management features could be handled more efficiently by other protocols as is done by the OpenFlow protocol. This all leads to the conclusion that by incorporating both parts of network management extensively, NETCONF has become sort of *jack of all trades, master of none.*

7.2.3 Opportunities

The opportunities of YANG and NETCONF based network management solutions are the descriptions of changes in the environment that could positively affect the solution. These opportunities are illustrated at various chapters of this thesis:

Interoperation - One of the big promises of YANG and NETCONF is the interoperability that opens up opportunities to decouple the network management system developers from the NF developers and allow a clean API that can be used to operate different NFs using the same set of methods. This in turn enables new markets as network management systems can be used to control wider range of networks and NFs can be deployed more easily on different kind of networks without large learning curve.

Collaboration - There is currently a lot of activity to standardize YANG models in the IETF work groups and these will produce a standardized configuration API with unparalleled scope to date. In addition, international companies such as Cisco Systems Inc. are pushing the NETCONF adoption forward to enable dynamic network management[1]. All this activity around YANG and NETCONF enable collaboration between different vendors by using the same set of tools to test and deploy network management solutions.

SDN and NFV - If the SDN and NFV network paradigms emerge as successful, widely used technologies, then the NF vendors that offer open and interoperable interfaces will have major advantage against closed proprietary solutions. Should this happen, network operators will likely start to demand for NFs to integrate to their logically centralized controllers and NFs to be dynamically configurable and easy to monitor to better utilize the infrastructure and cut costs.

7.2.4 Threats

The threats of YANG and NETCONF based network management solutions are the descriptions of changes in the environment that could negatively affect the solution. These threats are illustrated at various chapters of this thesis:

Full-blown SDN - Whilst NETCONF serves as an enabler for SDN it is not fully in line with the greater vision of SDN, which is a fully programmable network with no control logic embedded in the network infrastructure itself. Instead NETCONF offers concept cited in this thesis as dynamic control plane, which enables dynamic control of NFs whilst keeping the control logic within the network infrastructure. To achieve fully programmable networks there exists already protocols like OpenFlow that can control NFs as if they were scattered network of packet processors and this is something NETCONF is not designed to compete with. However, it is unlikely that all or even most of the network infrastructure would fully convert to full-blown SDN and thus there is probably

[1]Cisco Accelerates Greater Adoption of Network Programmability. Available at: `http://www.tail-f.com/cisco-accelerates-greater-adoption-of-network/`. Accessed 02.04.2016

need for protocols like NETCONF for decades to come. However, in the future it is a viable threat that NETCONF could become obsolete because of this.

Insufficient adoption - Another threat that NETCONF faces is that the development efforts of YANG models could stall before many of the vital YANG models are finished, leaving the open interface promises of YANG and NETCONF unfulfilled. Currently there is quite a push due to NFV and SDN transition to develop these models but standardization of these models is a long process and other solutions might come along and replace the NETCONF based solutions.

Change reluctance - Lastly one threat that affects nearly all changes from the status quo is the change reluctance. Network administrators have long operated their networks using the old methods and have invested a lot of time and effort on systems that are incompatible with the approach that YANG and NETCONF offer. It will take some persuasion to get users and developers to adopt a new way to manage networks and to incorporate NETCONF based solutions. If the NETCONF based network management's strengths are not valued enough and the weaknesses are deemed too severe, it might be rejected by the community.

7.2.5 SWOT Summary

This section first described the strengths and weaknesses of using a YANG and NETCONF based network management solution. As a summary it could be said that this solution offers many benefits to network management that can boost productivity, make network management more secure and reliable, cut operational costs and reduce configuration errors. Furthermore, the identified weaknesses such as high resource utilization are not severe and can be dealt with in modern networks.

The latter part of this section described the opportunities and threats of this network management solution. Possibly the most important opportunity offered by this solution is the smooth transition from traditional networks to the new network designs driven by the SDN and NFV paradigms. The other opportunities highlight the enabled collaboration that, when embraced, could offer a win-win situation for all parties developing YANG and NETCONF based solutions. However, the threats such as insufficient adoption should not be underestimated since this solution does not offer nearly all its potential if it remains as a niche technology.

8 CONCLUSIONS

Network administrators today have many capable network monitoring solutions available, but reacting to the changes in the network remains manual and slow. This thesis started off by exploring the existing network management solutions which clearly emphasise network monitoring capabilities over network configuration capabilities. Favouring of network monitoring capabilities is understandable because network configuration is hard, in fact much harder and error-prone than network monitoring. Network configuration is hard because it requires two-way collaboration between the network controller and the network device with the risk of breaking the network, whereas network monitoring usually only requires the controller to understand the network device without the risk of breaking the network. And because network configuration is so hard, it has lead to stagnation in the networks where network administrators are reluctant to change and evolve the networks as changes might break them. The virtualization technologies and the added network nodes that they bring have reached the limit where static networks are no longer an option and the advent of new network paradigms SDN and NFV has started an effort to make the networks more automated and configurable.

A modern approach for making network management, especially network configuration, more manageable was presented in this thesis. This approach consisted of the YANG data modelling language and the NETCONF network configuration protocol. YANG offers globally established interfaces to manage networks which would greatly reduce the complexity of network management since everybody would adhere to the same rules. Modelling network functions with YANG was shown in this thesis to be a straight forward process that is rather simple and powerful, although requiring collaboration to produce good abstractions for the modelled functionality. YANG alone, even without NETCONF, offers huge possibilities to advance the current disordered state of network configuration by introducing open standardized interfaces and is arguably more important of the two. On the other hand, NETCONF introduces a secure and reliable protocol that offers many advances, such as scalability improvements, over its predecessor SNMP. Whilst NETCONF is built on YANG models, it offers on top of YANG its own new standardized way to configure and monitor network devices that is, as show in the thesis, easy to integrate with existing network devices.

This new network management solution presents opportunities for open source communities and proprietary vendors alike. These opportunities promote collaboration and interoperability that benefit all included parties. In addition, this solution offers a smooth transition, which is really important for network infrastructure, from the old stagnated networks to a more dynamic future proposed by SDN and NFV. The biggest threat that this solution is susceptible to is that if the development of new YANG models stagnates before the major network functionalities are modelled,

it will prevent many of the presented opportunities from becoming reality.

It is clear that there is a need for automated network management solutions in the future and the network management solution offered by YANG and NETCONF is a very viable candidate. And if these automated network management solutions start to get traction then maybe we might see automated, widely deployed and reliable network management in the 2020s.

REFERENCES

[1] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar. Making Middleboxes Someone Else's Problem: Network Processing as a Cloud Service. *ACM SIGCOMM Computer Communication Review*, 42(4):13–24, 2012.

[2] D. Oppenheimer, A. Ganapathi, and D. Patterson. Why Do Internet Services Fail, and What Can Be Done About It? In *USENIX symposium on internet technologies and systems*, volume 67. Seattle, WA, 2003.

[3] Z. Kerravala. As the Value of Enterprise Networks Escalates, So Does the Need for Configuration Management. *The Yankee Group*, page 4, 2004.

[4] IS ISO. 7498-4: Information Processing Systems Open Systems Interconnection Basic Reference Model - Part 4: Management Framework. *International Standards Organization, Geneva, Switzerland*, 1989.

[5] European Telecommunications Standards Institute (ETSI). Network Functions Virtualization (VNF); Terminology for Main Concepts in NFV. *ETSI White Paper*, Available at: `http://www.etsi.org/deliver/etsi_gs/NFV/001_099/003/01.01.01_60/gs_NFV003v010101p.pdf`. 2014.

[6] J. Schönwälder. RFC 3535: Overview of the 2002 IAB Network Management Workshop. *IETF Request For Comments*, Available at: `http://www.ietf.org/rfc/rfc3535.txt`. 2003.

[7] J. Case, M. Fedor, M. Schoffstall, and J. Davin. RFC 1157: A Simple Network Management Protocol (SNMP). *IETF Request For Comments*, Available at: `http://www.ietf.org/rfc/rfc1157.txt`. 1990.

[8] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. RFC 1441: Introduction to version 2 of the Internet-standard Network Management Framework. *IETF Request For Comments*, Available at: `http://www.ietf.org/rfc/rfc1441.txt`. 1993.

[9] J. Liu and G. Liu. Research and Implementation of SNMP-based Network Management System. In *2011 Fourth International Conference on Intelligent Networks and Intelligent Systems*, pages 129–132. IEEE, 2011.

[10] J. Case, R. Mundy, D. Partain, and B. Stewart. RFC 2570: Introduction to version 3 of the Internet-standard Network Management Framework. *IETF Request For Comments*, Available at: `http://www.ietf.org/rfc/rfc2570.txt`. 1999.

[11] R. Gerhards. RFC 5424: The Syslog Protocol. *IETF Request For Comments*, Available at: `http://www.ietf.org/rfc/rfc5424.txt`. 2009.

[12] M. Ersue and B. Claise. RFC 6632: An Overview of the IETF Network Management Standards. *IETF Request For Comments*, Available at: `http://www.ietf.org/rfc/rfc6632.txt`. 2012.

[13] B. Claise. RFC 5101: Specification of the IP Flow Information Export (IPFIX) protocol for the Exchange of IP Traffic Flow Information. *IETF Request For Comments*, Available at: `http://www.ietf.org/rfc/rfc5101.txt`. 2008.

[14] P. Newman, W. Edwards, R. Hinden, E. Hoffman, F. Ching Liaw, T. Lyon, and G. Minshall. RFC 1987: Ipsilon's General Switch Management Protocol Specification Version 1.1. *IETF Request For Comments*, Available at: `http://www.ietf.org/rfc/1987.txt`. 1996.

[15] S. Sezer, S. Scott-Hayward, P. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao. Are We Ready for SDN? Implementation Challenges for Software-Defined Networks. *Communications Magazine, IEEE*, 51(7):36–43, 2013.

[16] Open Networking Foundation. Software-Defined Networking: The New Norm for Networks. *ONF White Paper*, Available at: `https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf`. 2012. Accessed 25.03.2016.

[17] R. Wolter. A Brief History of Network Programmability and Related Fields. In *Network-Embedded Management and Applications*, pages 23–57. Springer, 2013.

[18] M. Jarschel, T. Zinner, T. Hoßfeld, P. Tran-Gia, and W. Kellerer. Interfaces, Attributes, and Use Cases: A Compass for SDN. *Communications Magazine, IEEE*, 52(6):210–217, 2014.

[19] Open Networking Foundation. OpenFlow Switch Specification, version 1.0.0, Available at: `https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.0.0.pdf`. 2009. Accessed 25.03.2016.

[20] European Telecommunications Standards Institute (ETSI). Network Functions Virtualization (VNF); Architectural Framework. *ETSI White Paper*, Available at: `http://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.02.01_60/gs_nfv002v010201p.pdf`. 2014.

[21] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee. Network Function Virtualization: Challenges and Opportunities for Innovations. *Communications Magazine, IEEE*, 53(2):90–97, 2015.

[22] M. Björklund. RFC 6020: YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF). *IETF Request For Comments*, Available at: `http://www.ietf.org/rfc/rfc6020.txt`. 2010.

[23] M. Björklund. RFC 7223: A YANG Data Model for Interface Management. *IETF Request For Comments*, Available at: `http://www.ietf.org/rfc/rfc7223.txt`. 2014.

[24] J. Schönwälder. RFC 6991: Common YANG Data Types. *IETF Request For Comments*, Available at: `http://www.ietf.org/rfc/rfc6991.txt`. 2013.

[25] A. Bierman and M. Björklund. RFC 7317: A YANG Data Model for System Management. *IETF Request For Comments*, Available at: `http://www.ietf.org/rfc/rfc7317.txt`. 2014.

[26] L. Lhotka. RFC 6110: Mapping YANG to Document Schema Definition Languages and Validating NETCONF Content. *IETF Request For Comments*, Available at: `http://www.ietf.org/rfc/rfc6110.txt`. 2011.

[27] M. Björklund. RFC 7277: A YANG Data Model for IP Management. *IETF Request For Comments*, Available at: `http://www.ietf.org/rfc/rfc7277.txt`. 2014.

[28] M. Björklund and J. Schönwälder. RFC 7407: A YANG Data Model for SNMP Configuration. *IETF Request For Comments*, Available at: `http://www.ietf.org/rfc/rfc7407.txt`. 2014.

[29] D. Bogdanovic, K. Sreenivasa, L. Huang, and D. Blair. Internet-Draft: Network Access Control List (ACL) YANG Data Model. *IETF*, Available at: `http://www.ietf.org/id/draft-ietf-netmod-acl-model-07.txt`. 2016. Accessed 11.03.2016.

[30] L. Lhotka and A. Lindem. Internet-Draft: A YANG Data Model for Routing Management. *IETF*, Available at: `http://www.ietf.org/id/draft-ietf-netmod-routing-cfg-21.txt`. 2015. Accessed 11.03.2016.

[31] C. Wildes and K. Agrahara. Internet-Draft: SYSLOG YANG model. *IETF*, Available at: `http://www.ietf.org/id/draft-ietf-netmod-syslog-model-03.txt`. 2015. Accessed 11.03.2016.

[32] R. Enns, M. Björklund, J. Schönwälder, and A. Bierman. RFC 6241: Network Configuration Protocol (NETCONF). *IETF Request For Comments*, Available at: `http://www.ietf.org/rfc/rfc6241.txt`. 2011.

[33] Y. Ylonen and C. Lonvick. RFC 4251: The Secure Shell (SSH) Protocol Architecture. *IETF Request For Comments*, Available at: `http://www.ietf.org/rfc/rfc4251.txt`. 2006.

[34] A. Bierman and M. Björklund. RFC 6536: Network Configuration Protocol (NETCONF) Access Control Model. *IETF Request For Comments*, Available at: `http://www.ietf.org/rfc/rfc6536.txt`. 2012.

[35] M. Wasserman. RFC 6242: Using the NETCONF Protocol over Secure Shell (SSH). *IETF Request For Comments*, Available at: `http://www.ietf.org/rfc/rfc6242.txt`. 2011.

[36] M. Badra, A. Luchuk, and J. Schönwälder. RFC 7589: Using the NETCONF Protocol over Transport Layer Security (TLS) with Mutual X.509 Authentication. *IETF Request For Comments*, Available at: `http://www.ietf.org/rfc/rfc7589.txt`. 2015.

[37] P. Shafer. RFC 6244: An Architecture for Network Management Using NETCONF and YANG. *IETF Request For Comments*, Available at: `http://www.ietf.org/rfc/rfc6244.txt`. 2011.

[38] M. Daniele, B. Wijnen, M. Ellison, and D. Francisco. RFC 2741: Agent Extensibility (AgentX) Protocol Version 1. *IETF Request For Comments*, Available at: `http://www.ietf.org/rfc/rfc2741.txt`. 2000.

[39] K. Tran, H. Wang, and X. Chen. Internet-Draft: Yang Data Model for Internet Protocol Security (IPsec). *IETF*, Available at: `http://www.ietf.org/id/draft-tran-ipsecme-yang-01.txt`. 2016. Accessed 11.03.2016.

[40] M. Daniele, B. Wijnen, and D. Francisco. RFC 2257: Agent Extensibility (AgentX) Protocol Version 1. *IETF Request For Comments*, Available at: `http://www.ietf.org/rfc/rfc2257.txt`. 1998.

[41] N. Provos, M. Friedl, and P. Honeyman. Preventing Privilege Escalation. In *USENIX Security*, volume 3, 2003.

[42] B. Hedstrom, A. Watwe, and S. Sakthidharan. *Protocol Efficiencies of NETCONF versus SNMP for Configuration Management Functions*. PhD thesis, University of Colorado, Available at: `http://morse.colorado.edu/~tlen5710/11s/11NETCONFvsSNMP.pdf`. 2011. Accessed 12.02.2016.

[43] P. da Paz Ferraz Santos, R. Pereira Esteves, and L. Zambenedetti Granville. Evaluating SNMP, NETCONF, and RESTful Web Services for Router Virtualization Management. In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pages 122–130. IEEE, 2015.

[44] C. Augeri, Bulutoglu D., B. Mullins, R. Baldwin, and Baird L. An Analysis of XML Compression Efficiency. In *Proceedings of the 2007 workshop on Experimental computer science*, page 7. ACM, 2007.

APPENDIX A

YANG Data Model for IPsec Protocols

```
module ipsec {
  namespace "urn:ietf:params:xml:ns:yang:ipsec";
  prefix ipsec;

  import ietf-inet-types {
    prefix inet;
  }
  import ietf-netconf-acm {
    prefix nacm;
  }
  import ietf-interfaces {
    prefix if;
  }
  import ietf-ip {
    prefix ip;
  }
  import ipsec-types {
    prefix ipsec-types;
  }
  import keys {
    prefix keys;
  }

  organization
    "Insta DefSec Oy
     Sarankulmankatu 20
     33901 Tampere,
     Finland";

  contact
    "Joonas Ruohonen <joonas.ruohonen@insta.fi>";

  description
    "This YANG module defines essential components for the management
     of a ipsec subsystem.

     Copyright (c) 2015 IETF Trust and the persons identified as
     authors of the code. All rights reserved.

     Redistribution and use in source and binary forms, with or
     without modification, is permitted pursuant to, and subject to
     the license terms contained in, the Simplified BSD License set
     forth in Section 4.c of the IETF Trust's Legal Provisions
```

```
    Relating to IETF Documents
    (http://trustee.ietf.org/license-info).

    This version of this YANG module is part of RFC XXXX; see the
    RFC itself for full legal notices.";

  revision 2015-08-23 {
    description
      "Initial revision.";
    reference "RFC XXXX: A YANG Data Model for IPsec Management";
  }

  feature tunnel-auto-negotiation {
    description
      "Indicates support for starting IKEv2 negotiations
    automatically without any traffic flowing through the
    gateway. This lowers the connection startup times, but
    also increases the processing load on the network.";
  }

  feature certificate-revocation {
    description
      "Indicates support for certificate revocation lists.
    Revocation list's location will be read from the CA
    certificates.";
  }

  feature extra-revocation-server {
    description
      "Indicates support for defining additional CRL servers
    in the configuration.";
  }

  feature dead-peer-detection {
    description
      "Indicates support for DPD functionality which tries
    to resolve if a peer has died after configured amount of
    idle time.";
  }

  feature interface-selector {
    description
      "Indicates support for interface restrictions in traffic
    selectors.";
  }

  typedef manual-policy-ref {
```

```
    description
      "Reference to manual security association that can be
    used to for example identify the SA that requires reloading.";
    type leafref {
      path "/ipsec:ipsec/ipsec:manual-policies/ipsec:manual-policy/ipsec:name"
    }
  }

  typedef dynamic-policy-ref {
    description
      "Reference to IKEv2 security association that can be
    used to for example identify the SA that requires reloading.";
    type leafref {
      path "/ipsec:ipsec/ipsec:dynamic-policies/ipsec:dynamic-policy/ipsec:name
    }
  }

  grouping gateway-addresses {
    description
      "Defines the IP addresses that are used for IKEv2 and IPsec
    traffic.";

    choice address-family {
      description
        "Addresses can be either both IPv4 or both IPv6 but not mixed.";
      container ipv4 {
        leaf local-ip {
          description
            "Reference to locally configured interface's IP where this
          security association's gateway address is located.";
          type leafref {
            path "/if:interfaces/if:interface/ip:ipv4/ip:address/ip:ip";
          }
          mandatory true;
        }
        leaf remote-ip {
          description
            "IP address of the remote peer.";
          type inet:ipv4-address;
        }
      }
      container ipv6 {
        leaf local-ip {
          description
            "Reference to locally configured interface's IP where this
          security association's gateway address is located.";
          type leafref {
```

```
          path "/if:interfaces/if:interface/ip:ipv6/ip:address/ip:ip";
        }
        mandatory true;
      }
      leaf remote-ip {
        description
          "IP address of the remote peer.";
        type inet:ipv6-address;
      }
    }
  }
}

grouping gateway-identity {
  description
    "A gateway can be identified with a IP address or
  FQDN which maps to the certificate provided by the peer.";
  choice identity {
    leaf ipv4-address {
      type inet:ipv4-address;
    }
    leaf ipv6-address {
      type inet:ipv6-address;
    }
    leaf fqdn {
      type ipsec-types:fqdn;
    }
  }
}

grouping gateway-identities {
  description
    "A gateway can own multiple identities and thus
  those can be configured using local-id. If no local-id
  is defined, the local-ip assumed as the identity.";
  container local-id {
    uses gateway-identity;
  }
  container remote-id {
    uses gateway-identity;
  }
}

grouping selectors {
  description
    "A rule contains one-to-one mapping between two
  networks that are connected by the IPsec tunnel.
```

```
The rule can be tightened by defining the allowed
protocol and possibly the allowed port also.

Multiple rules combined form a security association
between peers and share keys. Rules are always one-way.";
choice address-family {
  description
    "Addresses can be either both IPv4 or both IPv6 but not mixed.";
  container ipv4 {
    container sources {
    list source {
      description
        "Defines the source network and possibly port where from
      the traffic is allowed to flow through the tunnel.";
      key "ip";
      max-elements 16;
      leaf protocol {
        description
          "Defines the layer 4 protocol that is to be matched
        with this rule.";
        type identityref {
          base ipsec-types:protocol-type;
        }
        default "ipsec-types:any";
      }
      leaf ip {
        type inet:ipv4-prefix;
        mandatory true;
      }
      leaf port {
        description
          "Only the protocols which utilize ports allowed.";
        when
          "../protocol='tcp' or
           ../protocol='udp' or
           ../protocol='sctp'";
        type inet:port-number;
      }
    }
    }
    container destinations {
    list destination {
      description
        "Defines the destination network and possibly port where to
      the traffic is allowed to flow through the tunnel.";
      key "ip";
      max-elements 16;
```

```
      leaf protocol {
        description
          "Defines the layer 4 protocol that is to be matched
        with this rule.";
        type identityref {
          base ipsec-types:protocol-type;
        }
        default "ipsec-types:any";
      }
      leaf ip {
        type inet:ipv4-prefix;
        mandatory true;
      }
      leaf port {
        description
          "Only the protocols which utilize ports allowed.";
        when
          "../protocol='tcp' or
           ../protocol='udp' or
           ../protocol='sctp'";
        type inet:port-number;
      }
    }
    }
  }
  container ipv6 {
    container sources {
    list source {
      description
        "Defines the source network and possibly port where from
      the traffic is allowed to flow through the tunnel.";
      key "ip";
      max-elements 16;
      leaf protocol {
        description
          "Defines the layer 4 protocol that is to be matched
        with this rule.";
        type identityref {
          base ipsec-types:protocol-type;
        }
        default "ipsec-types:any";
      }
      leaf ip {
        type inet:ipv6-prefix;
        mandatory true;
      }
      leaf port {
```

```
          description
            "Only the protocols which utilize ports allowed.";
          when
            "../protocol='tcp' or
             ../protocol='udp' or
             ../protocol='sctp'";
          type inet:port-number;
        }
      }
      }
      container destinations {
      list destination {
        description
          "Defines the destination network and possibly port where to
        the traffic is allowed to flow through the tunnel.";
        key "ip";
        max-elements 16;
        leaf protocol {
          description
            "Defines the layer 4 protocol that is to be matched
          with this rule.";
          type identityref {
            base ipsec-types:protocol-type;
          }
          default "ipsec-types:any";
        }
        leaf ip {
          type inet:ipv6-prefix;
          mandatory true;
        }
        leaf port {
          description
            "Only the protocols which utilize ports allowed.";
          when
            "../protocol='tcp' or
             ../protocol='udp' or
             ../protocol='sctp'";
          type inet:port-number;
        }
      }
      }
    }
  }
}

container ipsec {
  leaf active {
```

```
    type boolean;
  }
  container dead-peer-detection {
    if-feature dead-peer-detection;
    leaf enabled {
      type boolean;
      default "false";
    }
    leaf timeout {
      type uint16;
      units "seconds";
      default "3600";
    }
    leaf retry-limit {
      type uint8;
      default "5";
    }
    leaf retry-timer {
      type uint16;
      units "milliseconds";
      default "500";
    }
    leaf retry-timer-max {
      type uint16;
      units "milliseconds";
      default "3000";
    }
  }
  container certificate-revocation {
    if-feature certificate-revocation;
    leaf enabled {
      type boolean;
      default "true";
    }
    leaf search-limit {
      type uint8;
      default "5";
    }
    leaf search-timer {
      type uint16;
      units "seconds";
      default "20";
    }
    container servers {
    list server {
      key "host";
      leaf host {
```

```
        type inet:host;
      }
      leaf port {
        type inet:port-number;
        default "389";
      }
      leaf protocol {
        type identityref {
          base ipsec-types:protocol-type;
        }
        default "ipsec-types:ldap";
      }
    }
    }
  }
  container manual-policies {
    list manual-policy {
      key "name";
      leaf name {
        type string;
      }
      leaf direction {
        type identityref {
          base ipsec-types:direction-type;
        }
        mandatory true;
      }
      leaf mode {
        type identityref {
          base ipsec-types:mode-type;
        }
      }
      leaf encapsulation {
        type identityref {
          base ipsec-types:encapsulation-type;
        }
      }
      uses gateway-addresses;
      leaf spi {
        type uint32;
        mandatory true;
      }
      container authentication {
        when
          "../encapsulation='esp' or
           ../encapsulation='ah'";
        leaf algorithm {
```

```
          type identityref {
            base ipsec-types:authentication-type;
          }
          mandatory true;
        }
        leaf key {
          type keys:preshared-key-ref;
          mandatory true;
        }
      }
      container encryption {
        when "../encapsulation='esp'";
        leaf algorithm {
          type identityref {
            base ipsec-types:encryption-type;
          }
          mandatory true;
        }
        leaf key {
          type keys:preshared-key-ref;
          mandatory true;
        }
      }
    }
  }
  container dynamic-policies {
    list dynamic-policy {
      key "name";
      leaf name {
        type string;
      }
      leaf mode {
        type identityref {
          base ipsec-types:mode-type;
        }
      }
      leaf active {
        type boolean;
        default "true";
      }
      leaf auto-negotiate {
        if-feature tunnel-auto-negotiation;
        type boolean;
        default "false";
      }
      leaf use-crl {
        if-feature certificate-revocation;
```

```
    type boolean;
    default "true";
  }
  leaf lifetime {
    type uint32;
    units "seconds";
    default "7200";
  }
  uses gateway-addresses;
  uses gateway-identities;
  choice authentication {
    mandatory true;
    leaf preshared-key {
      type keys:preshared-key-ref;
    }
    container public-key {
      leaf private-key {
        type keys:private-key-ref;
        mandatory true;
      }
      leaf public-key {
        type keys:certificate-ref;
        mandatory true;
      }
      container certificate-authorities {
        leaf-list certificate-authority {
          type keys:certificate-authority-ref;
          min-elements 1;
        }
      }
    }
  }
  container algorithms {
    leaf-list authentication {
      type identityref {
        base ipsec-types:authentication-type;
      }
      min-elements 1;
    }
    leaf-list encryption {
      type identityref {
        base ipsec-types:encryption-type;
      }
      min-elements 1;
    }
    leaf-list prf {
      type identityref {
```

```
        base ipsec-types:authentication-type;
      }
      min-elements 1;
    }
    leaf-list dh-group {
      type identityref {
        base ipsec-types:dh-group-type;
      }
      min-elements 1;
    }
  }
  container transforms {
      leaf encapsulation {
        type identityref {
          base ipsec-types:encapsulation-type;
        }
      }
      leaf lifetime {
        type uint32;
        units "seconds";
        default "3600";
      }
      leaf lifebytes {
        type uint32;
        units "bytes";
        default "500000000";
      }
      container algorithms {
        leaf-list authentication {
          type identityref {
            base ipsec-types:authentication-type;
          }
          min-elements 1;
        }
        leaf-list encryption {
          type identityref {
            base ipsec-types:encryption-type;
          }
          min-elements 1;
        }
        leaf-list dh-group {
          type identityref {
            base ipsec-types:dh-group-type;
          }
          min-elements 1;
        }
      }
```

```
      }
    }
  }
  container rules {
    list rule {
      key "name";
      ordered-by "user";
      leaf name {
        type string;
      }
      leaf action {
        type identityref {
          base ipsec-types:action-type;
        }
        mandatory true;
      }
      leaf direction {
        type identityref {
          base ipsec-types:direction-type;
        }
        mandatory true;
      }
      leaf interface {
        if-feature interface-selector;
        description
          "...";
        type leafref {
          path "/if:interfaces/if:interface/if:name";
        }
      }
      choice security-association {
        when "../action='encapsulate'";
        leaf manual-policy {
          type manual-policy-ref;
        }
        leaf dynamic-policy {
          type dynamic-policy-ref;
        }
      }
      uses selectors;
    }
  }
}
rpc activate {
  nacm:default-deny-all;
  input {
    leaf dynamic-policy {
```

```
        type dynamic-policy-ref;
      }
    }
  }
  rpc reset {
    nacm:default-deny-all;
  }
}
```