Harmonized network monitoring

Olli Kauppinen

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 31.7.2020

Thesis supervisor:

Prof. Jukka Manner

Thesis advisor:

D.Sc. (Tech.) Nuutti Varis

Aalto University
School of Electrical Engineering

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
MASTER'S THESIS

Author: Olli Kauppinen

Title: Harmonized network monitoring

Date: 31.7.2020 Language: English Number of pages: 7+84

Department of Communications and Networking

Professorship: Networking technology

Supervisor: Prof. Jukka Manner

Advisor: D.Sc. (Tech.) Nuutti Varis

As the modern society is increasingly dependant on computer networks especially as the Internet of Things is gaining popularity, a need for monitoring computer networks along with associated devices increases. Additionally, the amount of cyber attacks is increasing and certain malware such as Mirai target especially network devices. In order to effectively monitor computer networks and devices, advanced solutions are required for collecting and storing the information.

This thesis designs and implements a novel network monitoring system. The presented system is capable of utilizing state-of-the-art network monitoring protocols and harmonizing the collected information using a common data model. This design allows effective queries and further processing on the collected information.

The presented system is evaluated by comparing the system against the requirements imposed on the system, by assessing the amount of harmonized information using several protocols and by assessing the suitability of the chosen data model. Additionally, the protocol overheads of the used network monitoring protocols are evaluated.

The presented system was found to fulfil the imposed requirements. Approximately 21% of the information provided by the chosen network monitoring protocols could be harmonized into the chosen data model format. The result is sufficient for effective querying and combining the information, as well as for processing the information further. The result can be improved by extending the data model and improving the information processing. Additionally, the chosen data model was shown to be suitable for the use case presented in this thesis.

Keywords: Network monitoring, data model, data processing, protocol

AALTO-YLIOPISTO SÄHKÖTEKNIIKAN KORKEAKOULU	DIPLOMITYÖN TIIVISTELMÄ

Tekijä: Olli Kauppinen

Työn nimi: Tietoverkkojen valvonnan yhdenmukaistaminen

Päivämäärä: 31.7.2020 Kieli: Englanti Sivumäärä: 7+84

Tietoliikenne- ja tietoverkkotekniikan laitos

Professuuri: Tietoverkkoteknologia

Työn valvoja: Prof. Jukka Manner

Työn ohjaaja: TkT Nuutti Varis

Yhteiskunnan ollessa jatkuvasti verkottuneempi, erityisesti Esineiden Internetin kasvattaessa suosiotaan, tarve seurata sekä verkon että siihen liitettyjen laitteiden tilaa ja mahdollisia poikkeustilanteita kasvaa. Lisäksi tietoverkkohyökkäysten määrä on kasvamassa ja erinäiset haittaohjelmat, kuten Mirai, ovat suunnattu erityisesti verkkolaitteita kohtaan. Jotta verkkoa ja sen laitteiden tilaa voidaan seurata, tarvitaan tehokkaita ratkaisuja tiedon keräämiseen sekä säilöntään.

Tässä diplomityössä suunnitellaan ja toteutetaan verkonvalvontajärjestelmä, joka mahdollistaa moninaisten verkonvalvontaprotokollien hyödyntämisen tiedonkeräykseen. Lisäksi järjestelmä säilöö kerätyn tiedon käyttäen yhtenäistä tietomallia. Yhtenäisen tietomallin käyttö mahdollistaa tiedon tehokkaan jatkojalostamisen sekä haut tietosisältöihin.
Diplomityössä esiteltävän järjestelmän ominaisuuksia arvioidaan tarkastelemalla, minkälaisia osuuksia eri verkonvalvontaprotokollien tarjoamasta informaatiosta voidaan yhdenmukaistaa tietomalliin, onko valittu tietomalli soveltuva verkonvalvontaan sekä varmistetaan esiteltävän järjestelmän täyttävän sille asetetut vaatimukset. Lisäksi työssä arvioidaan käytettävien verkonvalvontaprotokollien siirtämisen kiinteitä kustannuksia, kuten otsakkeita.

Työssä esitellyn järjestelmän todettiin täyttävän sille asetetut vaatimukset. Eri verkonvalvontaprotokollien tarjoamasta informaatiosta keskimäärin 21% voitiin harmonisoida tietomalliin. Saavutettu osuus on riittävä, jotta eri laitteista saatavaa informaatiota voidaan yhdistellä ja hakea tehokkaasti. Lukemaa voidaan jatkossa parantaa laajentamalla tietomallia sekä kehittämällä kerätyn informaation prosessointia. Lisäksi valittu tietomalli todettiin soveltuvaksi tämän diplomityön käyttötarkoitukseen.

Avainsanat: tietoliikenneverkot, valvonta, tietomalli, telemetria, protokolla

Preface

This research was carried out in Aalto University's Critical Infrastructure Research unit and I would like to thank Jukka Manner, Marko Luoma and Nuutti Varis for providing this excellent opportunity. Additionally, I would like to thank Jukka and Nuutti for their guidance. Finally, I would like to thank my family and friends who have been there for me throughout these past six years of studies, you made the experience awesome. *So Long, and Thanks for All the Fish.*

Otaniemi, 31.7.2020

Olli Kauppinen

Contents

Abbreviations

API	Application Programming Interface
CIA	Confidentiality, Integrity, Availability
CIM	Common Information Model
CRUD	Create, Read, Update, Delete
CTI	Cyber Threat Intelligence
DMTF	Distributed Management Task Force
DTLS	Datagram Transport Layer Security
FQDN	Fully Qualified Domain Name
gNMI	gRPC Network Management Interface
GPB	Google Protocol Buffer
gRPC	gRPC Remote Procedure Call
HMAC	Hash-based message authentication code
IETF	Internet Engineering Task Force
IPFIX	IP Flow Information Export
JSON	JavaScript Object Notation
MDT	Model-Driven Telemetry
MIB	Management Information Base
NTP	Network Time Protocol
ONF	Open Networking Foundation
RPC	Remote Procedure Call
SNMP	Simple Network Monitoring Protocol
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
XML	Extensible Markup Language
YAML	Yet Another Markup Language
YANG	Yet Another Next Generation

1 Introduction

In recent years, network monitoring has generated much interest due to the rapidly growing number of cyber security incidents. In a survey conducted in the United Kingdom in 2019, one-third of the companies participating had identified a data breach [1]. Of these companies, nearly half had identified at least one breach or attack per month. In 2019, the average time to detect a cyber security incident was 56 days [2]. This has led to a need for monitoring networks effectively in order to detect and investigate cyber security incidents.

Currently, numerous solutions have been developed for network monitoring in academia and industry, offering various monitoring capabilities [3]–[12]. However, most of these solutions accept only a narrow set of data formats and protocols. Therefore, a network comprised of heterogeneous devices is difficult to monitor, as the devices are usually restricted to only a few possible data formats and protocols. Vendor specific solutions might not even be able to accept data in a usable format from other vendors' devices.

Network monitoring is commonly divided into four categories: fault detection, performance, accounting and security [13]. Fault detection collects information on the operation of network devices, such as routers or switches, and the information is used to detect outages or other error conditions. Performance monitoring focuses on the operating conditions such as throughput and memory usage whereas accounting data typically collect information regarding network flows and used capacity for billing purposes for example. In security monitoring, the focus is on the data which could reveal intrusions, such as unauthorized usage or data exfiltration. However, these categories often overlap and are typically used to complement each other. Although numerous studies have investigated different aspects of network monitoring, no studies have focused on harmonizing the network data collection for monitoring purposes. Moreover, little research has focused on storing heterogeneous monitoring data.

The aim of this thesis is to develop a network monitoring system for monitoring heterogeneous fleets of network appliances and storing the collected data into a standardized format to allow additional processing and analytics. In order to develop this solution, this thesis will evaluate state-of-the-art monitoring protocols and data models for storing network monitoring data. The requirements for the system will be defined, the architecture will be designed based on these requirements, and implemented as a proof of concept. Finally, the implemented system will be evaluated against the requirements and an analysis on the information value will be provided. Additionally, the protocol overhead of the chosen network monitoring protocols will be evaluated. The scope of this thesis is limited to developing the system using existing protocols and data models, as development of either is a complex topic that requires deep knowledge beyond the level of a master's thesis. Furthermore, the analytics of the collected monitoring data is beyond the scope of this thesis, as the topic is broad enough for a thesis on its own. The system proposed in this thesis was found to fulfil the imposed requirements as well as to provide additional value. Approximately 21% of the information collected from the evaluated protocols could be mapped to the chosen data model. However, the variance between the protocols is significant, which resulted from the protocol design choices. The evaluated protocols ranged from network flow to log collection protocols. Additionally, development areas were identified in order to improve the amount of harmonized information.

The remainder of this thesis is organized as follows. Chapter 2 introduces state-of-the-art data models for storing information regarding network devices and state. Chapter 3 introduces state-of-the-art network monitoring protocols. Chapter 4 defines the requirements and presents the architecture of a harmonized network monitoring system as well as describes the development and technical details of the presented system. Chapter 5 evaluates the system against the defined requirements, the protocol overhead of the used monitoring protocols, the suitability of the chosen data model as well as evaluates the value added by the system. Finally, Chapter 6 summarizes the performed work as well as proposes possible improvements and future research topics.

2 Data models

This chapter introduces state-of-the-art data models designed for storing network monitoring information. These models are crucial for network monitoring systems as normalizing the incoming data into a common format enables easier processing and analytics. Additionally, in this context it is important to differentiate data models from information models: Information models are used describe the modeled objects and required functionality, whereas data models are meant for implementors and thus contain lower level implementation- and protocol-specific details [14]. Therefore, data models can be seen as subordinates to information models.

The results of this chapter will be used to choose the data model for the implementation of a harmonized network monitoring system.

2.1 Introduction

Computer networks often consist of heterogeneous devices, that may convey information using different formats. For example, some devices use big-endian integers whereas others use little-endian. Common data models are trying to solve this issue by specifying the format of the data. Additionally, separation of the data models from the protocols allows improved extensibility as the data models can be added or updated without modifying the protocol specification. Therefore, organizations and vendors may easily define and distribute their own models.

The Management information base (MIB) used by Simple Network Monitoring Protocol (SNMP) was one of the first solutions trying to solve this issue. However, the MIB modules are described using Structure of Management Information specification, which introduces certain drawbacks such as being dependant to the Abstract Syntax Notation One (ASN.1) specification and having multiple exceptions to the ASN.1 as well being strictly coupled to the SNMP [15]. Next Generation Structure of Management Information (SMIng) was introduced in 1999 to address these issues, however, it was never accepted as a standard and was released as an experimental specification in order to preserve the information [15]. Yet Another Next Generation (YANG) was created in 2010 to provide a data modeling language for NETCONF-protocol [16]. YANG's syntax resembles SMIng's and C-programming language's syntax. However, in contrast to SMIng, YANG is dedicated to NETCONF protocol whereas SMIng is protocol agnostic. To a some extent, YANG is compatible with SMIv2 MIB modules in order to provide a migration path to newer protocols. The details of MIB and YANG models are discussed in Sections 3.2 and 3.4.

The data models are constantly evolving and serving various purposes and every solution has their own advantages and disadvantages. This thesis aims to find the model suitable for the use case of storing network monitoring data from heterogeneous sources.

2.2 Apache Spot Open Data Model

Apache Spot is an open-source cyber security project leveraging big-data and machine learning for detecting cyber security incidents. The project includes an open data model (ODM) in order to provide analytics in a common format and enable sharing information between organizations. The included data model is designed for security event log and alert data and is complemented by several context models. The context models are network, endpoint, user and threat intelligence context models, which are used to augment the security event and alert data model. Additionally, context models are meant for more static data that changes rarely and the specification recommends to fill the context models using user, identity or asset management systems in order to ease the administration.[17]

The data model provides concise tables regarding data model categories, attributes and attribute types with example values. Table 1 presents an example of the data model. The data model is extendable by using `additional_attrs`-field, which can contain custom key-value pairs in JSON-format. The `additional_attrs`-field may only contain a JSON-string. The ODM recognizes several attribute types in order to ensure compatibility between different implementations. The used data types are long, int, string, float, bigint, boolean, double and binary data. [17]

Table 1: Example of Apache Spot Open Data Model [17]

Category	Attribute	Data type	Description	Sample values
Common	event_time	long	Timestamp of event (UTC)	1472653952
	n_proto	string	Network protocol of event	TCP, UDP, ICMP
	additional_attrs	map	Custom event attributes	"building":"729","cube":"401"
Device	dvc_version	string	Version	“3.2.2”
Network	src_ip4	bigint	Source ip address of event	Integer representation of 10.1.1.1
	src_domain	string	Domain name of source address	companyA.com
	src_port	int	Source port of event	1025

The data model can be used with various data formats and databases. However, the data model recommends that Apache Avro, JSON or Apache Parquet is used as a data format [17]. Apache Avro is a data serialization format utilizing JSON schemas and binary encoding [18]. The JSON schemas are used for serializing the data to the Avro binary format. By separating data and schema, the data can be stored in a more compact format. Listing 1 presents an example of the JSON schema. Additionally, Avro is compatible with many of the Apache Hadoop ecosystem projects.

Using plain JSON as the data format is straightforward as all attributes can be used as JSON fields, and no additional structure is needed. Additionally, as the the plain JSON is not binary encoded, the format is human readable.

Apache Parquet is a column-oriented compressed binary file format , which can utilize multiple compression algorithms including GZIP and BZIP2 [18]. Being column-oriented, Parquet stores all columns adjacently instead of storing by row allowing more efficient query processing as redundant columns can be skipped [18].

```
{ "type": "record",
  "doc": "This event records SSHD activity",
  "name": "auth",
  "fields": [{
        "name": "event_time",
        "type": "long",
        "doc": "Stop time of event"},
    {
        "name": "net_src_ip4",
        "type": "long",
        "doc": "Source IP Address" },
    {
        "name": "net_src_host",
        "type": "string",
        "doc": "Source hostname"},
    {
        "name": "net_src_port",
        "type": "int",
        "doc": "Source port"}]}
```

Listing 1: Example of Apache Avro Schema

The Apache Spot Open Data Model is still under development and subject to change. Moreover, the data model contains some errors that seem to stem from the work in progress nature of the data model. For example, field "user_dn" is missing the data type definition. However, the data model can be easily implemented and provides concise methods for extending it. Additionally, the work in progress state can be seen as an opportunity to affect the development of the data model.

2.3 Splunk Common Information Model

Splunk common information model (CIM) is a shared semantic model consisting of multiple data models in order to normalize incoming data. Splunk is a platform for storing big data with effective search functions. It utilizes the CIM as search-time schema where the data is formatted to the data model when search results are returned. Utilizing a search-time schema has several benefits including that the data can be stored in a raw format, which allows further utilization of the data with ease. Additionally, changing the used data model is simple as data is not required to be converted to an another model. However, using a search-time schema has a processing overhead as the data has to be formatted to the data model at runtime. Version 4.15.0 of the Splunk CIM contains twenty-two data models including web, alerts, network traffic and authentication [19]. Splunk claims its CIM to be flat, simple and flexible and attributes those qualities to using only the least common denominator. [20]

Splunk CIM can be extended to support custom fields, however, this requires cloning of the original data model and creating a new data model based on the clone [21]. Splunk does not offer a possibility to contribute to the development of the data model.

The data models in CIM consists of field names and tags. Tags are used to partition log entries into categories and subcategories. For example, a web proxy event could have tags "web" and "proxy" assigned to it. Tag "web" represents the category and "proxy" subcategory of the event. As the CIM is primarily used with Splunk, the data types for fields are presented as high-level concepts as the user does not need to know the implementation details. The used data types are "string", "boolean", "number", "integer", "time" and "timestamp". [19] Examples of the web data model are presented in Tables 2 and 3.

Table 2: Splunk web data model tags [19]

Dataset	Tag name
Web	web
Proxy	proxy

Table 3: Splunk web data model example fields [19]

Dataset	Field name	Data type	Description	Example values
Web	cached	bool	Indication whether the event is cached	true, false, 0, 1
Web	http_method	string	The used HTTP method	GET, PUT, POST
Web	bytes	number	Total number of bytes transferred	1123
Web	http_user_agent	string	HTTP user agent used for the request	Mozilla/5.0

2.4 Elastic Common Schema

Elastic has developed the Elastic Common Schema (ECS) for their Elasticsearch product, which is a search and analytics engine for various types of data. ECS is an open-source data model designed for event data such as metrics and logs.

Its development in managed in GitHub and contributions to its development are welcomed. [22]

ECS has three different field definitions: core, extended and custom. Core fields are fully defined fields used ECS top-level objects and should apply to most common use cases. Extended fields are partially defined, apply to fewer use cases and can be interpret more broadly than core fields. Custom fields are undefined and unnamed fields that are user-supplied and are meant for extending ECS, if ECS does not have a suitable field. ECS does not impose any requirements for custom fields besides mandating that must not interface with core or extended fields. However, as ECS is constantly being developed, a conflict between a custom field and a future ECS field is possible. In addition to fields names, Elasticsearch data types are specified for each field. Complete set of Elasticsearch data types are presented in [23]. ECS utilizes nesting for fields and is indicated with dots in fields names. For example, "log.syslog.facility.code=1" has three layers of nesting. Listing 2 presents the previous example in JSON format. [24]

```
{
"log": {
    "syslog": {
        "facility": {
            "code":  1
}}}}
```

Listing 2: Example of ECS nesting

Certain core event fields are classified as categorization fields. These fields are "kind", "category", "type" and "outcome" and are used to partition events based on the metadata of the event regardless of the source. For example, categorization information could be used to find all events that are alerts related to authentication and the authentication attempt succeeded. [25]

Table 4 and Listing 4 demonstrate how the Syslog message presented in Listing 3 is mapped to ECS fields:

```
    <6>1 2020-05-20T15:09:22.567464+03:00 machine kernel - - -
   [2925254.857211] usb 1-10: New USB device strings: Mfr=0,
   Product=1, SerialNumber=0
```

Listing 3: Example Syslog message

2.5 CRUSOE

CRUSOE was developed by researchers from Masaryk University in order to improve cyber situational awareness for incident response. It is designed as an extendable layered data model working as an information exchange point for various domains and tools. The design allows obtaining data in an automated fashion. Furthermore, the data model is designed to be extendable and can work as a backbone for additional data by connecting the additional data to an element in the data model. The model

Table 4: Mapping of a Syslog message to ECS

Field	Description	Incoming value
log.original	Original log message	<6>1 2020-05-20T15:09:22.567464+03:00 machine kernel - - - [2925254.857211] usb 1-10: New USB device strings: Mfr=0, Product=1, SerialNumber=0
@timestamp	Time when the event occurred	2020-05-20T15:09:22.567464+03:00
message	Event message	[2925254.857211] usb 1-10: New USB device strings: Mfr=0, Product=1, SerialNumber=0
event.kind	Information on what kind of information the event contains	event
event.category	Categorization of the event	driver
event.type	Subcategory	connection
event.outcome	Indication whether the event succeeded	success
host.hostname	Device hostname	machine
log.syslog.facility.code	Syslog facility code	0
log.syslog.facility.name	Facility name	kernel
log.syslog.priority	Syslog priority	6
log.syslog.severity.code	Syslog severity	6
log.syslog.severity.name	Severity name	informational
log.level	Log level	informational

also makes a distinction between fixed and timed entities and relations. Fixed entities are relatively static and old values should be discarded whereas timed entries are tied to specific start and end times and old values should be stored as active security incidents change often and historical data provides value for incident response. [26]

The requirements for CRUSOE were derived from selected North Atlantic Treaty Organization (NATO) use cases and interviews with incident handlers from several incident response teams. The following NATO use cases were chosen:

- Single authoritative data source, the system can be considered as the authoritative data source due to quantity and quality of the data.
- View connections of asset, a user can view asset dependencies to other components using the system.
- Fuse data, the system should allow combining data from multiple sources to create a more complete view.

```
{
  "timestamp":"2020-05-20T15:09:22.567464+03:00",
  "message":"[2925254.857211] usb 1-10: New USB device strings: Mfr=0, Product=1, SerialNumber=0",
  "event":{
    "kind":"event",
    "category":"driver",
    "type":"connection",
    "outcome":"success"
  },
  "host":{
    "hostname":"machine"
  },
  "log":{
    "level":"informational",
    "original":"<6>1 2020-05-20T15:09:22.567464+03:00 machine kernel - - - [2925254.857211] usb 1-10: New USB device strings: Mfr=0, Product=1, SerialNumber=0",
    "syslog":{
      "facility":{
        "code":0,
        "name":"kernel"
      },
      "priority":6,
      "severity":{
        "code":6,
        "name":"informational"
      }
    }
  }
}
```

Listing 4: Syslog message mapped to ECS in JSON format

- Drill down / Roll up, a user can access lower- or higher-level data related to an element.
- View asset dependencies, assets should be organized within a dependency hierarchy, which would allow users to identify asset dependencies.
- View interconnectivity, a user can view both logical and physical connections between network entities.

In addition to NATO use cases, three use cases were identified from the interviews with incident handlers: criticality estimation of an attack target, finding persons responsible for affected systems and vulnerability prioritisation and dissemination. Criticality estimation of an attack target is crucial for incident response in order to effectively prioritize work. Finding responsible persons rapidly improves incident response coordination and the process of resolving the incident. Vulnerability prioritisation and dissemination is required as numerous vulnerabilities are disclosed continuously. Therefore, there should be an effective way to determine if a vulnerability affects the organization and how critical the vulnerability is. These use cases where formulated into the following requirements:

- All-embracing, the data model should be able to capture concepts and technologies commonly used, including clustering, virtualization and virtual private networks.
- Comprehensive, the model should include information regarding all aspects of cyber security and should combine collected information.
- Attainable, the data model must include essential cyber security related information. However, too detailed or unobtainable parameters should be excluded.
- Sustainable, the model should be easy to maintain.
- Time-conscious, the data model must be able to handle time-series data
- Extensible, The model should be extensible with capabilities to tie additional information to the model.

The data model is divided into seven layers, which can be seen as views on the system from different perspectives. Figure 1 presents the layers with their inter-layer relations. Overlapping in the diagram implies a relation between layers.

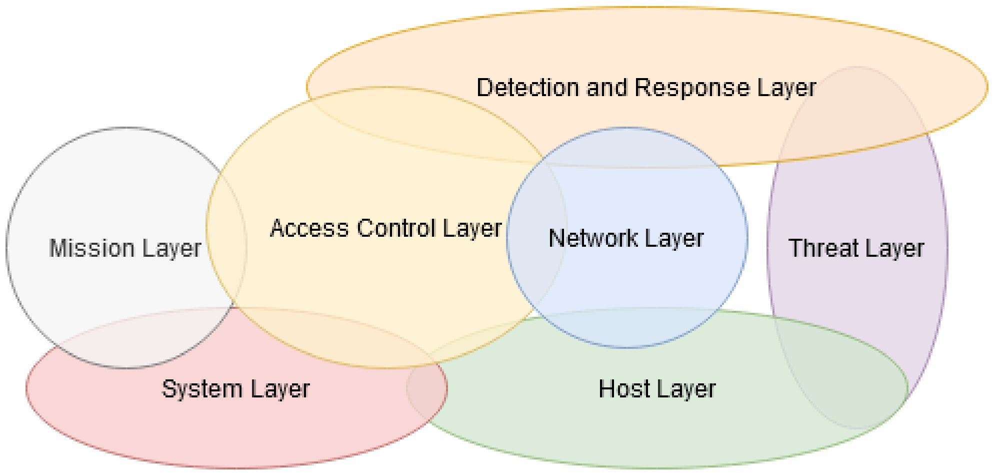

Figure 1: CRUSOE data model layers and relations [26]

CRUSOE data model does not discuss in depth how it should be implemented. For example, data types and field names are not discussed. The examples provided can be used as a guide, but contain little documentation [27]. This limits the usability of the data model and also might render different implementations of the data model incompatible. Moreover, one of the requirements was extensibility, but no details regarding how it can be extended were provided.

The layered architecture of the data model provides a clear separation of duties and allows fusing and storing data from multiple domains allowing the system to support cyber security analytics beyond networks and network devices.

2.6 I2NSF NSF Monitoring YANG Data Model

I2NSF NSF Monitoring YANG Data Model is an Internet Engineering Task Force (IETF) draft proposing an information model and a data model for monitoring network security functions (NSF), such as firewalls, intrusion prevention systems (IPS) and antivirus functions. [28]. It should be noted, that as it is a draft, it is a work in progress and will expire on 8th of November 2020. Therefore, it might not be accepted as an IETF Internet Standard, however, this model was included in this thesis as it represents state-of-the-art.

The data model is intended to be used with IETF's Framework for Interface to Network Security Functions (I2NSF), which is a project aiming to standardize interfaces and data models for network security functions in order to increase usability. Having these common models allows a user to utilize a controller to manage network security functions in a vendor agnostic manner. [29][30]

The data model lists several use cases stating the necessity of a such data model:

- Historical data can be built from NSF activity logs for operational or business use.
- NSF activity and event logs can be used for root cause analysis of various issues.
- Monitoring data can be utilized to achieve high availability as events can be used to detect failures.
- Advanced analytics can be performed on NSF logs to detect and predict security issues in deployments.
- Security information and event management systems can utilize NSF events to detect suspicious activities.
- NSF events can be used to configure additional security measures based on defined policies.

The I2NSF data model has a basic information model that all monitoring data should adhere to, and an extended information model that contains additional attributes related to e.g. logs, events and alarms. However, it should be noted that this definition does not adhere to the IETF description of data and information models presented in Section 2.1. The model is simple with no intra-model relations. The extended model should only be used with structured data whereas unstructured data is restricted to the basic model. The basic model contains seven attributes:

- message_version: Two-digit numeral indicating data format version starting from 01.
- message_type: Message types are the same as extended information models. Messages types include firewall events, virus alerts and alarm for example. For complete list of message types see [28].

- time_stamp: Message creation time.
- vendor_name: Network security function vendor.
- NSF_name: The hostname or IP-address of the NSF sending the message.
- module_name: The module responsible for sending the message.
- Severity: Log level indicator with value from 0 to 7 with zero being the most severe.

The use of Yet Another Next Generation (YANG) as the data modeling language provides an easy way to represent information needed to implement the data model including structure, field names and types as well as descriptions. Listing 5 presents the YANG description of a botnet log notification. Definitions for the extended models are presented in [28].

```
notification nsf-log-botnet {
     description
     "This notification is sent, if there is
     a new botnet event log in the nsf log";
     leaf attack-type {
        type identityref {
            base botnet-attack-type;
        }
        description
        "The botnet attack type for
        nsf-log-botnet notification";
    }
    leaf action {
        type log-action;
        description
        "Action type: allow, alert,
        block, discard, declare,
        block-ip, block-service";
    }
    leaf botnet-pkt-num{
        type uint8;
        description
        "The number of the packets sent to
        or from the detected botnet";
        }
    leaf os{
        type string;
        description
        "simple os information";
        }
    uses characteristics;
    uses common-monitoring-data;
}
```

Listing 5: Example of I2NSF botnet log notification [17]

The I2NSF data model is designed towards storing processed events and logs produced by network security functions, as most of the extended models are oriented towards alerts such as intrusions or different attempted attacks. Moreover, the data model's capabilities storing raw logs from protocols such Simple Network Management Protocol (SNMP) or Syslog, are limited as suitable extended models are not implemented in the draft. Additionally, the documentation does not discuss the extensibility of the data model. Therefore, this data model would be more suitable for storing analytics produced from the information gathered by a system such as the one presented in this thesis.

2.7 Summary

This chapter introduced state-of-the-art data models for network monitoring. The introduced data models have varying use cases and range from general wide use models to models with a narrow scope. Regarding the goal of this thesis, the general models appear to be more feasible due to being able to store data from various sources and having improved extensibility. Moreover, some of the introduced models did not discuss the implementation details. For example, data types were not presented, which may lead to incompatible implementations.

3 Network monitoring protocols

This chapter presents an overview of network monitoring and its use cases as well as introduces state-of-the-art network monitoring protocols and their properties. Additionally, certain emerging network monitoring protocols are introduced in order to recognize the recent developments in the field. In order to design an effective network monitoring system, it is important to understand the features, advantages and limitations of various network monitoring protocols. The results of this chapter are used to determine the minimum list of network monitoring protocols that the presented system should support.

3.1 Network monitoring

Network monitoring is typically divided into four categories: First category being fault detection, which aims to monitor the network devices in order to detect any hardware or link failures for example. Second category is performance, where performance of networks and network devices is monitored. This allows detecting bottlenecks in networks and upgrading device capabilities and hardware before any problems arise. Third category is accounting. For example, network flow data and interface counter values can be used for billing purposes. The fourth category is security, which aims to detect any security breaches and anomalies allowing organizations to effectively respond to security incidents. [13] As these categories require different kinds of information and networking protocols are typically designed for a certain use case, multiple network monitoring protocols are needed to capture information relevant to each particular use case, thus highlighting the need to harmonize collected data into a common format and model.

Originally network devices did not offer any protocols for extracting monitoring data. Instead command-line interfaces (CLI) were used either manually or with scripts to scrape the data. This was highly ineffective as the interfaces varied between vendors and updates might change the interfaces thus breaking any scripts. Network monitoring protocols were introduced to combat these issues with Syslog and Simple Network Management Protocol being among the first protocols. [13]

3.2 Simple Network Monitoring Protocol

Simple Network Monitoring Protocol (SNMP) was created by Internet Engineering Task Force (IETF) in 1988 for managing and monitoring Internet Protocol (IP) devices and it is currently in its third iteration (SNMPv3). However, versions 1 and 2 still see use in computer networks. SNMP allows managing network device configurations and retrieving device configurations using device and vendor agnostic interfaces. [31] This section focuses on the latest version of the protocol as it provides the most recent features and is supported by modern network devices.

Structure of Management Information

Structure of Management Information is a data modeling language based on Abstract Syntax Notation One (ASN.1). It is used to describe Management Information Base (MIB) modules, which are used by SNMP to provide interfaces for accessing configuration and state data. The Management Information Base is a virtual data store that holds all the configuration and state data and the MIB modules are used to describe the data elements. Latest version of the specification is version 2 (SMIv2), which is used with SNMP versions 2 and 3. Version 1 (SMIv1) is only used with SNMP version 1. However, with certain changes SMIv1 modules can be translated to SMIv2 format. [32][33]

The SMI has three different sets of definitions: module, object and notification. Module definitions are used to define information modules, their properties and associated objects. ASN.1 macro `MODULE-IDENTITY` is used to convey the semantics and the syntax of an information module. Currently three types of information modules exist: MIB modules, compliance statements and capability statements. MIB modules define inter-related managed objects. Compliance statements define the conditions, which have to be met in order to be compliant with the associated MIB module. Capability statements convey the information of capabilities present in a device implementing SNMP. [34] Object definitions define the single managed objects and their properties. For example, the number of present interfaces on a device can be an object. Macro `OBJECT-TYPE` is used to describe the semantics and the syntax of an object. Notification definitions define unsolicited notifications such as SNMPv2-Trap or InformRequest messages, which are used to send management information to a predetermined receiver. `NOTIFICATION-TYPE` macro describes the semantics and the syntax of the notifications. [32]

Transport layer

SNMP's modular design allows protocol messages to be sent over various transports. Originally SNMPv3 supported only IPv4 for the internet layer, however, IPv6 support was later included [35]. SNMP can be carried over the following transports: User Datagram Protocol (UDP), Transmission Control Protocol (TCP), Datagram Delivery Protocol (DDP), Internetwork Packet Exchange (IPX), Open Systems Interconnection (OSI), Connectionless-mode Transport Service (CLTS), Institute of Electrical and Electronics Engineers (IEEE) 802 (Ethernet), Secure Shell (SSH) and Transport Layer Security (TLS). Of these transports only UDP is mandatory for SNMP implementations while the rest are optional. Transports are required to support messages over 484 octets with the exception of TCP, which is required to support messages at least 8192 octets. [36]–[40]

SNMP over UDP is the preferred and mandatory transport mapping for SNMP implementations. In addition to the requirement of support for messages up to 484 octets in length, it is recommended for implementations to support messages up to 1472 octets in length. UDP port 161 dedicated for SNMP listeners whereas UDP port 162 is for SNMP trap listeners. [36]

SNMP over TCP transport mapping was defined to provide more efficient transfer method for bulk data. SNMP engines are required to send all bytes of a single message before sending bytes belonging to another message. Additionally, messages should not be interleaved in the TCP connection. These measures are required as SNMP requires the full message in one part in order to decode the message. TCP ports 161 and 162 are recommended for incoming connections and notification receivers respectively. [37]

SNMP messages can be transmitted over IEEE 802 networks, however, the scope is limited to a single logical, bridged or virtual local area network (LAN). Serialized SNMP messages are sent in an IEEE 802.3 frame with an Ethernet type of 33100. In addition to the aforementioned support for messages up to 484 octets, it is recommended that messages up to 1472 octets are supported. [38]

SNMP over IPX is an optional transport mapping for sending messages over Internetwork Packet Exchange (IPX) networks. SNMP datagrams are sent using IPX packet type 4. SNMP engines should listen to socket 36879 and socket 36880 for receiving inform and trap messages. SNMP entities using IPX transport mapping are required to support messages up to 546 octets in length and support for longer messages is recommended. [36]

SNMP over Datagram Delivery Protocol (DDP) is an optional transport mapping for sending SNMP messages over an AppleTalk network. The SNMP messages are sent using protocol type 8. Socket number 8 is used to listen for SNMP messages and socket number 9 to receive inform and trap messages. [36]

SNMP over OSI is an optional transport mapping for transporting SNMP messages using OSI's Connectionless-mode Transport Service. [36]

SSH transport model was specified for SNMP due to a demand from network administrators, as utilizing the user-based security model requires creating and maintaining a new key-infrastructure, whereas multiple networks already utilize SSH and associated key-infrastructure. SSH provides authentication, encryption and integrity protection features for the SNMP connection. The SSH transport model supports all SSH authentication methods defined in RFC4252. Additionally, SNMP leaves the selection and use of security protocols to SSH and details of these are not exposed to SNMP. Ports 5161 and 5162 have been assigned by IANA to be used with SNMP and SNMP Traps over SSH. [39]

The TLS transport model for SNMP was defined to provide authentication and privacy. The TLS transport model utilizes X.509 public-key infrastructure for authenticating endpoints and even though other authentication methods are supported by TLS and DTLS, they are not used with SNMP. Utilizing x.509 public-key infrastructure provides easy integration with existing public-key infrastructures. IANA has defined ports 10161 and 10162 for use with SNMP-DTLS and SNMP-TLS respectively. [40]

Message format

SNMP version 3 message is formed by msgVersion, msgGlobalData, msgSecurityParameters and msgData. Table 5 presents the message format. [41]

Field msgVersion is an integer identifying the SNMP version, for example value 3

Table 5: SNMPv3 message format [41]

msgVersion
msgID
msgMaxSize
msgFlags
msgSecurityModel
msgSecurityModel
msgSecurityParameters
contextEngineID
contextName
data

indicates SNMPv3. The msgID is used to correlate requests and responses between two SNMP entities as responses must use the same msgID as requests. Re-use of outstanding values should be avoided in order to reduce the risk of replay attacks. The msgMaxSize indicates the maximum length of a message supported by the sender. It is typed as an integer. The msgFlags contains bit fields for indicating the security level and reportability of the message. Currently SNMPv3 recognizes three flags: authFlag, privFlag and reportableFlag. The bit flag ordering in the byte-field is presented in Table 6. The combination of the first two determine the security level of the message. If both are zero, then the level is noAuthNoPriv, if authFlag is one and privFlag is zero, then the level is AuthNoPriv and finally if both are one, then the level is AuthPriv. Having authFlag as zero, and privFlag as one is not supported. Section 3.2 describes security levels in detail. The reportableFlag is used to aid in determining whether a ReportPDU must be sent in cases where the received protocol data unit (PDU) cannot be decoded. If the PDU is successfully decoded, the decoded PDU-type determines whether a ReportPDU will be sent. The msgSecurityModel indicates the used security model as an integer. Value 0

Table 6: msgFlags field structure in network byte order [41]

. . . . 1	authFlag
. . . . . . 1 .	privFlag
. . . . . 1 . .	reportableFlag

means "any", value 1 refers to SNMPv1, value 2 to SNMPv2c, User Security Model (USM) has value 3 and Transport Security Model (TSM) value 4 [42][43]. The msgID, msgMaxSize and msgSecurityModel fields can be referred as MsgGlobalData header containing administrative parameters. The msgSecurityParameters field is used by different security models to provide additional information and its content depends on the security model. However, the type is always an octet string. [41]

The data field contains a scoped protocol data unit (scopedPDU), which indicates the used protocol operation along with the payload. Protocol operations are described later in this section. The scopedPDU consists of contextEngineID, contextName and PDU fields. The contextEngineID is used to uniquely identify an SNMP entity within an administrative domain and contextName is used to name a context within the SNMP entity. The contextName is required to be unique within the SNMP entity. A PDU consists of request-id, error-status, error-index and variable-bindings fields. PDU type indicates the used protocol operation as a 4 byte integer. Request-id is an identifier for correlating requests and responses and is represented as an unsigned 4 byte integer. Error-status and error-index indicate the error message and the index of the variable-binding causing the error, if applicable. Both fields are signed 4 byte integers. In BulkPDU operation, the error fields are replaced by non-repeaters and max-repetitions fields, which are used to determine the processing of the bulk request. Variable-bindings fields is a list of object identifiers (OIDs), which represent entities in SNMP. Together, contextEngineID, contextName and PDU fields form a scopedPDU, which is effectively the payload of a SNMP message. The scopedPDU is encrypted if User Security Model is used along with authPriv flag set. [41]

Message encoding

SNMP messages are encoded using Basic Encoding Rules (BER). BER encoding consists of four components: an identifier, length, content and an end of content indicator. The BER encoding can be used to encode single or nested elements as presented in Tables 7 and 8. Content is a series of octets encoded by rules presented in [44]. End of content is indicated by two octets with value zero and is only used if the length component is in indefinite format. For example, version field with version number "3" could be encoded as `02 01 03`, where byte `02` identifies the value as an integer, `01` is the length of the value and `03` is the actual value. Listing 6 presents an example of a BER encoded SNMPv2-Trap PDU. [41][44]

Table 7: BER example

Identifier	Length	Content

Table 8: BER nested field example

Identifier	Length	Identifier	Length	Content	Identifier	Length	Content

```
30:7C                                          SEQUENCE {
  04:08:80:00:02:B8:04:61:62:63                 800002b804616263
  04:04:63:74:78:31                             "ctx1"
  A7:6A                                         SNMPv2-Trap-PDU{
   02:03:6D:08:67                                 INTEGER 7145575
   02:01:00                                       INTEGER 0
   02:01:00                                       INTEGER 0
   30:5D                                          SEQUENCE OF {
    30:0F                                          SEQUENCE {
      06:08:2B:06:01:02:01:01:03:00                 sysUpTime.0
      43:03:01:72:8C                                94860 }
    30:17                                          SEQUENCE {
      06:0A:2B:06:01:06:03:01:01:04:01:00           snmpTrapOID.0
      06:09:2B:06:01:06:03:01:01:05:04              linkUp }
    30:0F                                          SEQUENCE {
      06:0A:2B:06:01:02:01:02:02:01:01:03           ifIndex.3
      02:01:03                                      3 }
    30:0F                                          SEQUENCE {
      06:0A:2B:06:01:02:01:02:02:01:07:03           ifAdminStatus.3
      02:01:01                                      up(1) }
    30:0F                                          SEQUENCE {
      06:0A:2B:06:01:02:01:02:02:01:08:03           ifOperStatus.3
      02:01:01                                      up(1) } } }
```

Listing 6: BER encoding example[45]

Protocol operations

The protocol operations are divided into seven classes: read, write, response, notification, internal, confirmed and unconfirmed. Read-class contains operations, that are used to retrieve information from the management information base. Operations Get, GetNext and GetBulk belong to read class. Write-class operations modify the management information and only Set-operation has been defined for this class. Response-class operations are used to respond to other protocol operations. The response-class contains Response and Report operations. Notification-class operations send unsolicited notifications to recipients. Examples of notification-class operations are SNMPv2-Trap and Inform. Internal-class operations are only exchanged between SNMP engines and are not exposed to the user. Therefore, these operations are considered as internal. Report is the only internal operation. Confirmed-class contains all operations that require a response. Set, Get, GetNext, GetBulk and Inform are confirmed operations. Unconfirmed-class operations are not responded to. Unconfirmed operations are Report and SNMPv2-Trap. [46][47]

- Get is used to query information from SNMP entities. The GetPDU contains queried OIDs as variable bindings and the queried entity responds with a ResponsePDU.
- GetNext can be utilized to traverse the MIB-tree by requesting next item for the last received OID.

- GetBulk operation requests, that multiple GetNext-operation results are returned in a single packet.
- Response operation returns the requested information to the client.
- Set assigns values to variables and therefore configures network devices. If one of value assignments fails, then all assignments will be rolled back and and a ResponsePDU is sent with "CommitFailed" message in the error-status field.
- SNMPv2-Trap is used to send notifications when an event has occurred. Traps do not have any confirmation messages associated, therefore the delivery of traps is not guaranteed and lost messages cannot be detected. A SNMPv2-TrapPDU always contains sysUpTime.0 and snmpTrapOID.0 variable-bindings indicating how long the sender system has been running and which is the MIB responsible for the event [48].
- InformRequest operation is similar to the SNMPv2-Trap operation. However, the receiver of an InformRequest should acknowledge the message by sending a ResponsePDU with same values as in the InformRequest and error fields set as zero.
- Report operation is used for SNMP's internal communications and its use and semantics are defined by frameworks utilizing it.

Security

SNMP framework version 3 recognizes three levels of security:

- No authentication and no privacy (noAuthNoPriv)
- Authentication and no privacy (AuthNoPriv)
- Authentication and privacy (AuthPriv)

In this context privacy means that the message is encrypted and thus protects the message from eavesdropping. Additionally, if a hash-based message authentication code (HMAC) is used, then the integrity of the message is protected as well. Authentication ensures that the user carrying out actions is legitimate and is not being impersonated. [41]

The modularity of SNMP framework allows the development of multiple security models independently. Currently SNMP version 3 has two security models defined: user-based (USM) and transport security model (TSM). The USM must be implemented by SNMP entities whereas TSM is optional. [42][49]

The user-based security model utilizes a user as a security concept where security information is tied to the username. The USM fills the msgSecurityParameters field with the following sub-fields: msgAuthoritativeEngineID, msgAuthoritativeEngineBoots, msgAuthoritativeEngineTime, msgUserName, msgAuthenticationParameters and msgPrivacyParameters. The msgAuthoritativeEngineID is used to specify the engineID of the SNMP engine that is considered as authoritative in this context. The msgAuthoritativeEngineBoots and msgAuthoritativeEngineTime fields contain the number of times the authoritative engine has been started and the current time of the authoritative engine. This information can be used as a replay protection as the message is discarded if the engine boots value differs from local value or the engine time differs from local time more than 150 seconds. The msgUserName identifies the user that is used to authenticate the message. The msgAuthenticationParameters and msgPrivacyParameters fields contain information depending on the used authentication and privacy protocols. For example, they may contain salts used by authentication and privacy protocols. [42] Originally HMAC-MD5-96 and HMAC-SHA-96 hash functions were defined for SNMP version 3 authentication. Later HMAC-SHA-2 hash functions were added to the specification with 224-, 256-, 384- and 512-bit lengths [50]. USM utilizes symmetric encryption for the messages. Data Encryption Standard (DES) and Advanced Encryption Standard (AES) encryption protocols are specified to be used with SNMP version 3 USM. AES is only specified for 128-bit key length, however, some vendors have implemented 192- and 256-bit key lengths to their products [51]. Additionally, DES is considered to be insecure and should not be used if possible [52][53].

Transport security model (TSM) was defined to provide interfaces to secure transport models such as SSH and TLS. The transport security model does not use the msgSecurityParameters field, which is set as an empty string. Instead, TSM utilizes securityStateReference and tmStateReference caches for storing fields such as transportDomain, transportAddress, securityName, and securityLevel. These fields are filled by the used secure transport model and provide information regarding the authentication details of the connection to the SNMP engine. [49][40][39]

In addition to authentication and encryption, SNMPv3 provides authorization through View-based Access Control Model (VACM). The VACM allows controlling the users and their permissions to access or modify information elements. Users are managed using groups, which contain information regarding the security levels and access rights to views The access rights can be read, write and notify. Read-rights allow accessing information elements where as write-rights allow modifying them. Notify-rights allow sending notifications on changes to a group member. Views are simple configuration elements containing a MIB or an MIB-subtree which is related to the view. [54]

3.3 Syslog

Syslog protocol is used to transport log messages over networks utilizing a textual format. The protocol is described in two major specifications, which are RFC3164 and RFC5424. Syslog protocol was first defined in RFC3164 - The BSD Syslog Protocol in 2001. However, as the BSD Syslog protocol was already in widespread usage, it only described the observed formats. Additionally, the BSD Syslog format has a major issue as the various implementations currently in use have incompatible formats as typically only the priority value is a shared property. RFC5424 was the first attempt to standardize Syslog protocol. [55][56]

Transport layer

Syslog protocol defined in RFC3164 uses User datagram Protocol (UDP) as a transport layer protocol with port 514 as the source and destination. Reliably transport by using Transmission Control Protocol (TCP) over Blocks Extensible Exchange Protocol (BEEP) is also supported. Port 601 should be used with reliable transport. [55][57]

RFC5424 is designed in a protocol agnostic manner, and can utilize any transmission protocol. However, RFC5424 requires that all implementations support Transmission Layer Security (TLS) -based transports and should support UDP as a transport layer protocol. In addition, Datagram Transport Layer Security (DTLS) can be used. Port 514 is used with UDP and 6514 with DTLS or TLS. [56][58][59][60]

Message format

There are two commonly used message formats regarding Syslog: RFC3164 and RFC5424. Both are plain-text formats with main differences regarding field types and ordering.

RFC3164 was used to describe a transport for Syslog messages with a recommended message format. As the message format is only a recommendation, the payload of packets destined to UDP port 514 should be considered as a Syslog message regardless of the content. The message format recommended by RFC3164 is defined as following:

```
<PRI>HEADER MSG
```

The PRI is defined to have three to five characters starting and ending with angle brackets. The numeric priority value is constrained with the angle brackets. The priority is calculated by multiplying facility number with 8 and adding the severity value. Facilities with corresponding codes are presented in Table 9 and severities with codes in Table 10.

The HEADER contains two fields: timestamp and hostname. Timestamp format is `Mmm dd hh:mm:ss` and is in local time instead of Coordinated Universal Time (UTC). The hostname is device's hostname and if it is not known, device's IPv4- or IPv6-address. The MSG field of the message is divided into two parts: TAG and CONTENT. TAG contains the name of the program or the process responsible for generating the message. The length of TAG is restricted to 32 alphanumeric characters with non-alphanumeric characters terminating the field. CONTENT is a free-form field describing the event. Listing 7 presents a valid message.

```
<34>Oct 11 22:14:15 mymachine su: 'su root' failed for lonvick on /
    dev/pts/8
```

Listing 7: RFC3164 compliant Syslog message [55]

RFC3164 requires that a single Syslog packet must be less than 1024 bytes in length, whereas RFC5424 does not impose any hard limits for message length.

Table 9: Syslog Message Facilities [55][56]

Numerical Code	Facility
0	kernel messages
1	user-level messages
2	mail system
3	system daemons
4	security/authorization messages
5	messages generated internally by syslogd
6	line printer subsystem
7	network news subsystem
8	UUCP subsystem
9	clock daemon
10	security/authorization messages
11	FTP daemon
12	NTP subsystem
13	log audit
14	log alert
15	clock daemon
16	local use 0 (local0)
17	local use 1 (local1)
18	local use 2 (local2)
19	local use 3 (local3)
20	local use 4 (local4)
21	local use 5 (local5)
22	local use 6 (local6)
23	local use 7 (local7)

Instead, implementations are required to support messages up to 480 octets in length and recommended to support messages up to 2048 octets. Larger messages may be used if both sender and receiver supports them. In a case where a too long message is received, the receiver may choose between discarding and truncating the message. If the message is truncated, it occurs at the end of the message. Additionally, one has to note that after truncation the message may contain invalid data and the receiver can either discard the message or try to process it. [55][56]

The RFC5424 message format consist of a header, structured data and a message:

```
HEADER STRUCTURED-DATA MSG
```

Where HEADER format is:

```
<PRI>VERSION TIMESTAMP HOSTNAME APP-NAME PROCID MSGID
```

The priority value in PRI is calculated in the same manner as in message format described in RFC3164 by multiplying facility number with eight and adding the

Table 10: Syslog Message Severities [55][56]

Numerical Code	Severity
0	Emergency: System is unusable
1	Alert: Action must be taken immediately
2	Critical: Critical condition
3	Error: Error condition
4	Warning: Warning condition
5	Notice: Normal, but significant condition
6	Informational: Informational message
7	Debug: Debug-level message

severity value. Facilities and severities with their corresponding codes are presented in Tables 9 and 10. VERSION is the version of the Syslog protocol. For RFC5424 the value is "1". The VERSION is incremented whenever changes are introduced to HEADER format. TIMESTAMP is formatted according to RFC3339, which defines the time and date format to be used with Internet protocols [61]. Additional restrictions for timestamp are introduced in RFC5424:

- Leap second shall not be used.
- Character "T" must be used.
- Characters "T" and "Z" must be in upper case.

If the system time could not be obtained, then NILVALUE must be used as a TIMESTAMP. In RFC5424, dash is used as NILVALUE. HOSTNAME defines the machine, which sent the Syslog message. Following values are accepted in their order of preference:

1. Fully qualified domain name (FQDN)
2. Static IP-address
3. Hostname
4. Dynamic IP-address
5. NILVALUE

NILVALUE is a fallback, and should be only used in the rare case where the machine cannot obtain a hostname or an IP-address. APP-NAME defines the application that generated the Syslog message. NILVALUE can be used, if the machine cannot provide this information. PROCID is typically used to provide the process name or the process ID of the Syslog process. However, it does not have any interoperable meaning besides that if the value changes, there has been discontinuity in reporting. MSGID describes the type of the message and is used for filtering messages. STRUCTURED-DATA is used to provide data in a well defined format. It can contain multiple structured data elements or "SD-ELEMENTS" as referred in RFC5424. However, if no structured data elements have been defined, it has to contain the NILVALUE. A single SD-ELEMENT is formed from a SD-ID and one or multiple SD-PARAMS. SD-IDs are identifiers for type and purpose of the SD-ELEMENT. Currently the are two types of SD-IDs: Internet Assigned Numbers Authority (IANA) registered IDs and additional IDs.

- IANA registered names cannot contain the following symbols: '@','=','"',']', whitespace or control characters. Certain IANA defined names are presented in [56] Section 7.
- Additional names can be defined by entities that have a registered IANA enterprise ID. The format for these names is `name@<private enterprise number>`, where name does not have any specific requirements besides restricting the use of the following symbols: '@','=','"',']', whitespace or control characters. The private enterprise number must be registered with IANA before taking it into use.

SD-PARAMS consist of a PARAM-NAME and a PARAM-VALUE. PARAM-NAMEs are case sensitive and scoped to their parent SD-ID. Therefore, same PARAN-NAMEs in different SD-IDs are distinct. PARAM-VALUEs must be UTF-8 encoded and characters '"', '{ }' and ']' must be escaped with a backslash. Listing 8 presents a valid STRUCTURED DATA field. MSG is an optional field, which is used for

```
  [exampleSDID@32473 iut="3" eventSource="Application" eventID
  ="1011"][examplePriority@32473 class="high"]
```

Listing 8: STRUCTURED DATA with multiple elements [56]

free-form messages and should be UTF-8 encoded.

Examples of RFC5424 compliant messages with and without structured data are presented in Listings 9 and 10.

```
<34>1 2003-10-11T22:14:15.003Z mymachine.example.com su - ID47 -
    BOM
'su root' failed for lonvick on /dev/pts/8
```

Listing 9: RFC5424 compliant message without structured data [56]

```
<165>1 2003-10-11T22:14:15.003Z mymachine.example.com evntslog -
    ID47 [exampleSDID@32473 iut="3" eventSource="Application"
    eventID="1011"] BOMAn application event log entry
```

Listing 10: RFC5424 compliant message with structured data [56]

Security

RFC3164 has many security concerns regarding the confidentiality, integrity and availability of the Syslog protocol. These concerns yield from use of UDP for transport, as the channel is not authenticated nor encrypted. Additionally, UDP is not a reliable protocol as delivery is not guaranteed. In the absence of these attributes it is possible for an attacker to modify messages during transport and/or spoof messages. In addition, as the maximum length of a packet is 1024 bytes, problems could be caused in some older versions of Syslog by sending large packets. To combat aforementioned issues RFC3195 was defined [57]. TLS and DTLS may be used as the transport with the RFC5424 specification, which solves the aforementioned issues as both offer protection for confidentiality and integrity. Additionally, TLS and DTLS allow mutual authentication with X.509 certificates. However, if UDP is used as the transport protocol, the same considerations apply as for RFC3164. [55][56][60]

3.4 NETCONF

NETCONF protocol was created to provide a simple way of configuring, managing and retrieving data from network devices through application programming interfaces (APIs). It utilizes the remote procedure call (RPC) paradigm and the Extensible Markup Language (XML) encoding for the messages. Some of the key aspects of NETCONF are allowing clients to discover capabilities supported by the server and easy mirroring of device native functionalities, which reduces implementation costs. Additionally, NETCONF can be used to automate the entire configuration process of network devices. [62]

NETCONF separates configuration and state data to make configuration and management easier. Configuration data can be modified in order to configure device settings, whereas state data is read-only and is used to query device status and other information regarding device's working conditions. Benefits yielding from this design include smaller datasets when saving device configurations, configurations do not try to write to read-only fields and easier comparison of configurations as there is no interfering state data. The configuration and state data are acquired using different RPC operations: <get-config> and <get>. Details on these operations are presented later in this section. [62]

The protocol is divided into four layers: content, operation, messages and secure transport [62]. Figure 2 presents the layers.

1. Secure transport layer provides the transport layer protocol with required properties for NETCONF. The requirements are discussed later in this section.

2. The messages layer supports two kinds of messages: RPCs and notifications sent by the device.

3. The operations layer describes the operations carried out by RPC, such as setting an attribute or retrieving configuration.

4. The content of the messages are defined using data models written in YANG, which was created for NETCONF [16].

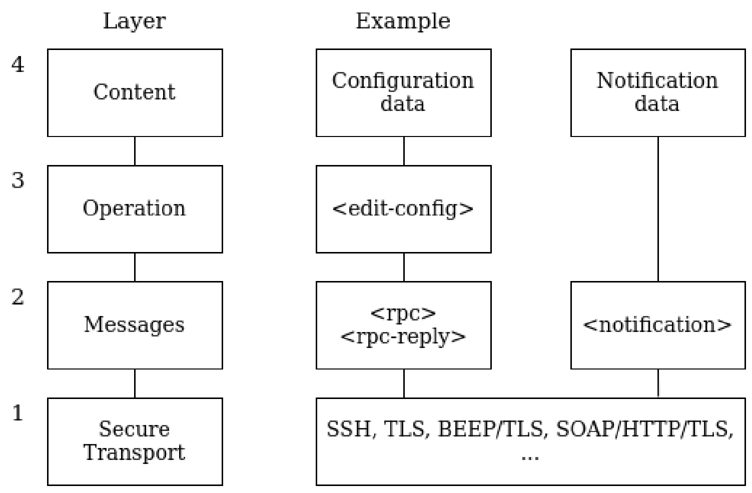

Figure 2: NETCONF protocol layers [62]

Transport layer

NETCONF is protocol independent and ca be used with any transport layer protocol that has the required functionalities. The required functionalities are connection-oriented operation, authentication, integrity and confidentiality. Additionally, all NETCONF implementations are required to support Secure Shell (SSH) transport protocol. Currently IETF specifications exist for SSH, Simple Object Access Protocol (SOAP), Blocks Extensible Exchange Protocol (BEEP) and Transport Layer Security (TLS) [63]–[66]. As NETCONF is connection-oriented, the connection has to be persistent. Additionally, due to the long-lived nature of the connection, clients are allowed to make changes to the connection which persist for the duration of the connection. An essential part of the connection-oriented operation is that the all resources requested from a server, have to be released when that particular connection is terminated. This ensures simple and robust failure recovery. The authentication requirement states that all NETCONF connections have to be authenticated and the authentication is to be done by the transport protocol. AS NETCONF provides an interface that closely mimics device's native interface, device's existing authentication mechanisms should be utilized by the underlying protocol. For example RADIUS [67] could be used for authentication. As a result of the authentication process, a user's identity and permissions are known to the device and have to be enforced during the NETCONF session. The integrity and confidentiality requirements are also handled by the transport layer protocol. [62]

Messages layer

Using Extensible Markup Language (XML) as the message encoding format allows human-readable representation of hierarchical data, that can be manipulated either with traditional text-editors or XML specific tools. NETCONF have several considerations that implementations should follow. NETCONF does not utilize document type declarations and those are forbidden in NETCONF. Used protocol elements are defined in `urn:ietf:params:xml:ns:netconf:base:1.0` -namespace. <rpc> and <rpc-reply> elements are used to distinguish requests and responses between a client and a server. Those elements contain a message-id attribute that is used to link responses to requests and it is a string chosen by the client. Commonly a monotonically increased integer is used. <rpc-error>, <data> and <ok> elements are used to represent the content of a response. These elements do not exist in a response simultaneously. The <rpc-error> element is used to signal error in processing a request. Multiple <rpc-error> elements are allowed if multiple errors were encountered during the message processing. The <rpc-error> element contains the following fields:

- error-type, defines the layer where the error occurred. Has one of the following values:
  - transport: secure transport layer
  - rpc: messages layer
  - protocol: operation layer
  - application: content layer
- error-tag, string identifying the error
- error-severity, the severity of the error. Can be either `warning` or `error`.
- error-app-tag contains a data model or implementation-specific error condition and is only included if an applicable error-app-tag can be set.
- error-path has the XPath expression identifying the node causing the error condition. This element will not be included if an applicable XPath error cannot be found.
- error-message contains a human-readable error message.
- error-info contains protocol or implementation-specific error content and is only included if an applicable error content can be set.

Standard NETCONF errors are detailed in Appendix A of [62]. Listing 11 presents an rpc-reply message containing an rpc-error. The ok element is sent when no data is returned by the operation and no errors or warnings occurred. Listing 12 presents an rpc-reply message with an ok element.

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<rpc-error>
<error-type>rpc</error-type>
<error-tag>missing-attribute</error-tag>
<error-severity>error</error-severity>
<error-info>
<bad-attribute>message-id</bad-attribute>
<bad-element>rpc</bad-element>
</error-info>
</rpc-error>
</rpc-reply>
```

Listing 11: Example of <rpc-reply> with <rpc-error> [62]

```
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<ok/>
</rpc-reply>
```

Listing 12: Example of <rpc-reply> with <ok> [62]

The contents of a data element and its format depend on the used YANG data model. An example of a data element is presented in Listing 13. [62]

```
<rpc-reply message-id="101" xmlns="
    urn:ietf:params:xml:ns:netconf:base:1.0">
<data>
<top xmlns="http://example.com/schema/1.2/config">
<users>
<user>
<name>root</name>
</user>
<user>
<name>fred</name>
</user>
<user>
<name>barney</name>
</user>
</users>
</top>
</data>
</rpc-reply>
```

Listing 13: Example of <data> element [62]

YANG

Yet Another Next Generation (YANG) is a data model designed for modeling state and configuration data for the NETCONF protocol. The latest revision of the specification is 1.1, which is a maintenance release addressing defects and ambiguities present in the original specification. The data models defined using YANG define the data hierarchy for NETCONF configuration, state, remote procedure call (RPC) and notification operations. Although YANG is specified to be used with NETCONF, it may be used with other protocols, however, that is not discussed by YANG documentation and is considered to be out of scope. [68]

YANG is a structural language and uses modules and submodules to structure data. Additionally, the modules can be augmented by other modules allowing additions to existing modules without modifying the original modules. Furthermore, the augmentations can be conditional. YANG organizes data hierarchically where every node contains either a value or child nodes. The YANG syntax resembles SMIng's and C-programming language's syntax, which is both human- and machine-readable. Listing 14 presents the YANG syntax. The YANG models can be translated to XML format called YANG Independent Notation (YIN), which can be converted back to YANG if necessary. YANG provides a set of data types for modeling data and a mechanism for defining new data types. As mentioned in Section 2.1, YANG is compatible with Simple Network Management Protocol's (SNMP's) Structure of Management Information version 2 (SMIv2) and any SMIv2 modules can be translated into YANG models in order to provide a way to migrate from SNMP to NETCONF for network management. However, converting YANG models into SMIv2 format is not supported. [68]

```
import acme-system {
    prefix "acme";
}
container http-server {
    leaf name {
        type string;
        }
    uses acme:endpoint;
}
```

Listing 14: Example of a YANG data model [68]

Protocol operations

NETCONF has currently 18 protocol operations defined that are used to perform low-level operations [62][69]–[72]:

- `get` is used to retrieve running configuration and state information.
- `get-config` is used to retrieve entire or parts of a configuration datastore.
- `edit-config` modifies the target datastore by loading all or part of a specified configuration. The configuration can be loaded from a local file, from a remote file, or inline.
- `copy-config` can be used to create or replace a configuration datastore with other datastore's content.

- `delete-config` clears a configuration datastore entirely.
- `lock` can be used to lock a datastore from modifications for a short period to ensure that other users' actions do not interfere with intended configuration changes.
- `unlock` releases a configuration lock acquired with the `lock` command.
- `close-session` is used to gracefully close the current session by releasing any locks and closing all connections.
- `kill-session` terminates the current session by aborting all current actions, releasing any locks and closing all related connections. Additionally, if this command is used while processing a confirmed commit, the configuration state is restored to the pre-commit state.
- `commit` operation sets the current candidate configuration as the running configuration.
- `discard-changes` discards the candidate configuration by reverting it to the previous state.
- `partial-lock` allows a user to lock parts of a datastore from modification, whereas the original lock command locks the entire datastore.
- `partial-unlock` is used to release the partial locks set with "partial-lock".
- `notification`s are sent by the server to the initiator of the subscription until the subscription is terminated.
- `create-subscription` creates the subscription for receiving notifications from a server on subscribed events until the subscription is terminated.
- `establish-subscription` is used to subscribe to notifications from a server on subscribed events. This operation differs from the `create-subscription` operation by allowing modification or deletion of the established subscription.
- `modify-subscription` allows modifying subscriptions created with the establish-subscription operation. However, subscriptions created with create-subscription cannot be modified.
- `delete-subscription` removes a subscription created by the same user and connection.
- `kill-subscription` allows a user to terminate any subscription present.
- `resync-subscription` is used only with on-change subscriptions and requests the server to send a push update on receiving the resync request.
- `get-data` retrieves configuration data from a specific datastore.

- `edit-data` edits the configuration in a specific datastore. Difference to edit-config operation is that the datastore can be chosen.
- `get-schema` allows a client to retrieve a data model from the server.

Notifications

NETCONF allows a client to subscribe to notifications, which can be either on-change or periodic. However, not all configuration items support on-change notifications. NETCONF notifications utilize the existing NETCONF connections and are terminated either along with the associated connection or by deleting or killing the subscription. Additionally, the subscriptions are either dynamic or configured. Dynamic subscriptions are always tied to a related connection and are removed when the connection is terminated, whereas configured subscriptions are preserved if the connection is terminated. Dynamic subscriptions are created using either `create-subscriptions` or `establish-subscriptions` protocol operation. Configured subscriptions are created by any means which allow editing a device's configuration and do not require initiating a connection in order to create the subscription. Additionally, supporting dynamic subscriptions is mandatory whereas for configured subscriptions its optional. [73][70][71].

Datastores

NETCONF utilizes datastores for storing and accessing configuration and state information. Originally configuration and state data were separated and thus had to be modeled twice in the data models. The configuration data was marked with "config true" attribute whereas state data was identified with "config false". This led to complicated data models and hindered their readability. To combat these issues, a new datastore architectural model was designed. The original datastore model had three datastores: candidate, startup and running. Additionally, operational state data was available, however, it was not defined as its own datastore. The datastores provided configurable information and could be written to and read from. Operational data was read only and contained only state data. However, support for candidate and startup datastores was optional and running datastore could be set as read only. Running datastore contains the configuration that is currently in use. However, it may include inactive configuration items or template-expansion configurations requiring further expansion. Candidate datastore contains a device's current configuration, which can be modified without affecting the running configuration. Candidate datastore can be written into running configuration by using the commit protocol operation. Startup datastore holds the configuration, which is loaded when a device starts. Figure 3 presents the original datastore architecture. The new datastore architecture presented in Figure 4 added a new intended datastore and defined operational data as a datastore. The intended data contains the read-only configuration of the startup datastore after all configuration transformations are applied to the startup datastore. [74]

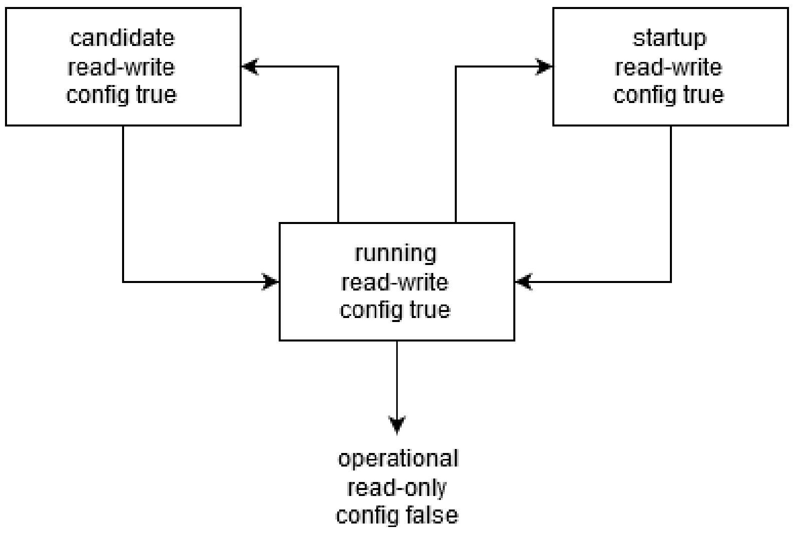

Figure 3: Original NETCONF datastore architecture [74]

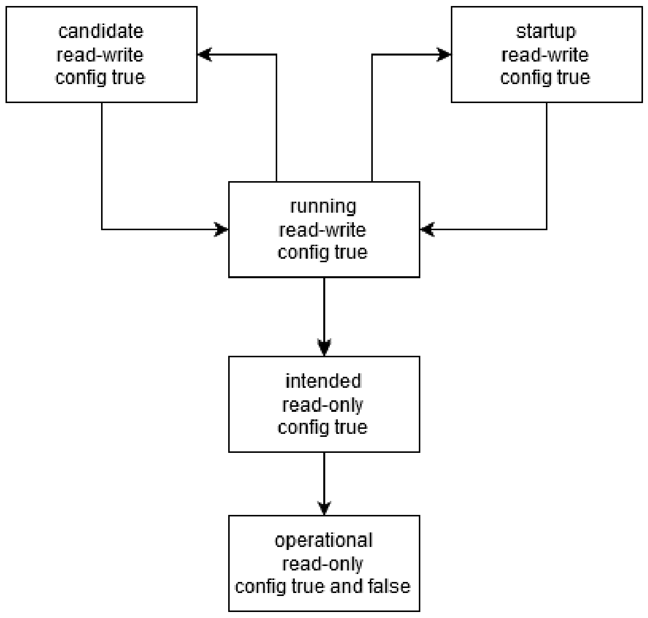

Figure 4: Revised NETCONF datastore architecture [74]

3.5 RESTCONF

RESTCONF was created to fulfil a need to access configuration, state and other network device data using web applications. It is based on the Hypertext Transfer Protocol (HTTP) and it utilizes concepts introduced in NETCONF. However, RESTCONF was not intended to replace NETCONF, but to provide a way to utilize NETCONF functionalities using a HTTP interface following Representational State Transfer (REST) principles. RESTCONF is compatible with NETCONF, however, it does not provide all the capabilities that NETCONF has. [75]

Transport protocol

As RESTCONF is built on HTTP, it may use any transport layer protocol that provides reliable transport. This has typically been Transmission Control Protocol (TCP) as HTTP has native support for TLS, which provides data integrity and confidentiality protection required by RESTCONF. Additionally, all RESTCONF implementations are required to support TLS.

RESTCONF utilizes X.509v3 certificates to authenticate servers and clients. The use of X.509v3 certificates is compatible with NETCONF over TLS specification. The server has to present a certificate, which the client validates using X.509 certificate path validation or by matching the server's certificate with a certificate acquired from a trusted source. The trusted source can be a pinned certificate, for example. The client is required to check that the server's hostname matches the certificate in order to verify server identity and to detect any man-in-the-middle attacks.

RESTCONF requires that all clients trying to access any information must be identified. Recommended method is to require an X.509v3 certificate from the client, however, HTTP basic authentication can be used if implementing certificate authentication is not feasible. Additionally, HTTP and certificate authentication can be used in conjunction, however, implementation details left out of the specification. [75]

Message format

Message content is encoded using JSON or XML and a server is required to support either of these with supporting both being optional. Additionally, clients need to support both in order to operate with different RESTCONF servers. The XML message format is similar to NETCONF's format and the YANG data models are shared between NETCONF and RESTCONF. An example of the JSON format is presented in Listing 15. [75]

```
{
"example:interface" : [
  {
    "name" : "eth1",
    "mtu" : 1500,
    "@mtu" : {
       "ietf-netconf-with-defaults:default" : true
    },
    "status" : "up"
  }
]
}
```

Listing 15: RESTCONF JSON format [75]

Methods

RESTCONF utilizes HTTP methods to distinguish CRUD (create, read, update and delete) operations. Table 11 presents the RESTCONF methods and the corresponding NETCONF operations. However, the listing is not complete as RESTCONF does not support all NETCONF features nor operations. As shown in Table 11, a single method may have multiple purposes depending on the payload of the protocol message.

Table 11: RESTCONF methods [75]

RESTCONF	NETCONF
OPTIONS	none
HEAD	<get-config>, <get>
GET	<get-config>, <get>
POST	<edit-config> (nc:operation="create")
POST	Initiating an RPC operation
PUT	<copy-config>
PUT	<edit-config> (nc:operation="create/replace")
PATCH	<edit-config> (nc:operation depends on PATCH content)
DELETE	<edit-config> (nc:operation="delete")

Resources

RESTCONF resources are accessed using Universal Resource Identifiers (URIs) and the root resource is `/restconf`. However, the server may specify leading parts for the root resource. For example, `/config/top/restconf` is a valid RESTCONF root location. The location of the root resource is queried from the server by issuing a HTTP GET request to `/.well-known/host-meta` resource. The server provides a response presented in Listing 16. As RESTCONF utilizes XML and

```
HTTP/1.1 200 OK
Content-Type: application/xrd+xml
Content-Length: <length>
<XRD xmlns='http://docs.oasis-open.org/ns/xri/xrd-1.0'>
    <Link rel='restconf' href='/restconf'/>
</XRD>
```

Listing 16: Root resource discovery response [75]

JSON encoding, the used type is indicated in the `Content-Type` HTTP Header. Allowed values are `application/yang-data+xml` and `application/yang-data+json`. The structure of the RESTCONF Application Programming Interface (API) is defined in `ietf-restconf` YANG data model. The root `/restconf` resource contains three subresources: `data`, `operations` and `yang-library-version`. Data contains all configuration and state data resources defined by the YANG data models present on the system. RESTCONF implementations may support all the datastores defined by NETCONF, however, it is not mandatory [76]. `Operations` is an optional resource, which contains the RPC-operations defined in YANG data models. `Yang-library-version` is a mandatory resource, which contains the revision date of the `ietf-yang-library` data model present on the system. Listing 17 presents an HTTP request that invokes the `system-restart` RPC. Listing 18 presents an

```
POST /restconf/operations/ietf-system:system-restart HTTP/1.1
Host: example.invalid
Accept: application/yang-data+xml
```

Listing 17: HTTP Request for invoking `system-restart` RPC

HTTP request that queries information regarding interfaces present on the system using `ietf-interfaces` YANG data model. [75]

```
GET /restconf/data/ietf-interfaces:interfaces HTTP/1.1
Host: example.invalid
Accept: application/yang-data+xml
```

Listing 18: HTTP Request for querying present interfaces

Notifications

RESTCONF has notifications similar to the NETCONF protocol, however, RESTCONF servers are not required to support them. Clients can determine whether server supports notifications by querying the `stream` list using HTTP GET, OPTIONS or HEAD method. RESTCONF utilizes HTML5 Server-Sent Events for sending the notifications. HTML5 Server-Sent Events specification offers an API for sending push messages from a server to a client over persistent connection using HTTP or other dedicated server-push protocols [77]. Event data is structured in the NETCONF <notification> format. [75]

3.6 NetFlow and IPFIX

NetFlow was created by Cisco in 1996 to collect network flow information from network devices. This information is intended to be used for IP accounting, traffic analysis, network telemetry and security monitoring [78]. Currently 10 versions of NetFlow exist, of which version 5 is the most widely adopted by manufacturers [78][79]. Version 9 introduced support for IPv6 and templates, which allows improved flexibility and extensibility for data records [80]. IP Flow Information Export (IPFIX) was developed from NetFlow version 9 by the IETF IPFIX Working Group and is regarded as NetFlow version 10 [81][82] NetFlow and IPFIX define a network flow as an unidirectional sequence of packets with common properties [80][81].

NetFlow and IPFIX systems consist of two separate components: The collector and the exporter. The exporter's duties are creating the flows from observed traffic and forwarding the flows to the collector. Typically exporters are routers or switches. The collector is only responsible for receiving and storing the flows sent by one or multiple exporters. Additionally, the collector may aggregate the flows before storage. [80][81]

NetFlow version 9 and IPFIX utilize a template-based approach as it provides extensibility, modularity and compatibility. Templates are sent by the NetFlow exporter before sending the flow data. Data records corresponding to a particular template can appear in the same in subsequent export packet as the template. To reduce the amount of transmitted data, templates are not send in every packets. Therefore, the collectors are required to cache the templates in order to parse data records. This allows adding new fields to the flow records easily as changes to the record format are not required. As templates contain structural information regarding fields, the collectors can parse the flows even if they do not understand semantics of a new field. Being modular, NetFlow version 9 and IPFIX allow exporting only required fields and thus reducing data volume and computing requirements. [80][81]

Transport layer

NetFlow version 9 utilizes the user datagram protocol (UDP) as its transport layer protocol. However, it has been designed in a transport protocol independent manner. As UDP is not a congestion aware protocol, deploying NetFlow version 9 in a congestion sensitive environment may lead to decreased quality of service as the traffic volumes generated by NetFlow can be voluminous. RFC3954 recommends that in congestion sensitive environments, there should be a dedicated link between collectors and exporters. [80]

As IPFIX is based on NetFlow v9, it is designed in a transport protocol independent manner and all implementations are required to support Stream Control Transmission Protocol (SCTP) with Partially Reliable SCTP (PR-SCTP) extension whereas UDP and Transmission Control Protocol (TCP) support is defined as optional. IANA has assigned ports 4739 and 4740 to be used with IPFIX. Port 4739 is used with SCTP, UDP or TCP and port 4740 with Transport Layer Security (TLS) or Datagram Transport Layer Security (DTLS) over aforementioned protocols. Different ports may be used if deemed necessary. As IPFIX uses a 16-bit field for describing the length of a message, maximum length of a message is 65535 octets, which includes the header. [81]

Message format

NetFlow and IPFIX packets consist of a packet header and at least one FlowSet. There are three different FlowSet types: Data, Template and Options Template. A single packet can contain any combination of these FlowSets. A FlowSet contains one or multiple records, which have been grouped together. A Data Flowset can contain either Data records or Options records whereas Template and Options Template FlowSets can only contain their respective Template or Options Template records.

FlowSets are distinguished by their FlowSet IDs. Template FlowSet has an ID of "0" and Options Template ID is "1". Additionally, IDs lower than 256 are restricted for special FlowSets such as aforementioned Template and Options Template. Figure 5 presents an example of a NetFlow packet consisting of a packet header, a Template FlowSet, a Data FlowSet and an Options Template FlowSet. [80][81]

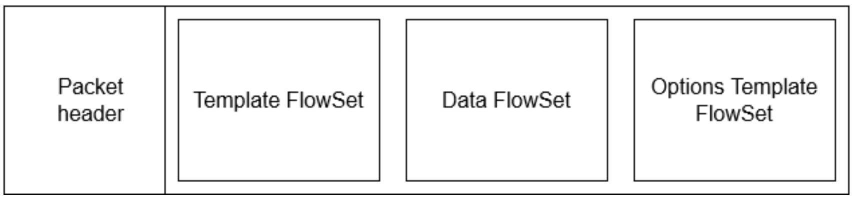

Figure 5: Example of a NetFlow version 9 packet

NetFlow v9 and IPFIX headers are similar for the most part, however, there are some slight differences. The NetFlow v9 header contains six fields whereas the IPFIX header has only five. NetFlow v9 has a sysUpTime-field for describing the time elapsed since the device was last rebooted. IPFIX has decided to drop this field from the specification. NetFlow's UNIX Secs and IPFIX's Export time fields both express the time when the packet leaves the exporter and difference being only the field name. The format of the field being seconds since 00:00 UTC 1.1.1970 as a 32-bit unsigned integer. Both protocols utilize incremental Sequence number field to detect missed packets at the collector. The sequence number is observation domain specific and thus packets originating from the same domain should have continuous values. Additionally, IPFIX has specified that when utilizing SCTP, the sequence numbers are counted per stream basis, whereas the packets are considered to be part of the same stream for a TCP connection or an UDP session. Observation domain ID is a 4 byte integer that identifies the observation domain. The IDs are unique per exporting process and the specification recommends that the IDs are unique per IPFIX device. The ID should be zero when an ID relevant to the entire message cannot be assigned. Sending exporting process statistics is an example of a case where ID should be zero. Observation domain is a set of observation points of which flow information can be aggregated. For example, a line card consisting of multiple interfaces can be an observation domain. Exporting process is responsible for sending protocol messages to collecting processes. Header formats of NetFlow v9 and IPFIX are presented in Tables 12 and 13 respectively. The numbers in the first row represent bytes in the header. [80][81]

IPFIX has recognized a need for vendors to define proprietary fields. Therefore, IPFIX uses Field Specifiers to describe the used information elements in templates. In contrast, NetFlow v9 does not offer this capability and vendors are restricted to Cisco defined field types. The information elements describe the information and the data type of the carried information. The first bit of a field specifier is an Enterprise bit, which describes whether the information element is defined by IANA or by another authority. Value zero represents IANA and value one the other authorities. If the bit is zero, then the enterprise number field is omitted from the field specifier. Otherwise the enterprise number must be provided. Information

Table 12: NetFlow version 9 header [80]

0	1	2	3	4
Version number			Count	
sysUpTime				
UNIX Secs				
Sequence number				
Source ID				

Table 13: IPFIX header [81]

0	1	2	3	4
Version number			Length	
Export time				
Sequence number				
Observation domain ID				

element identifier is a numeric value that determines the element type. IANA defined information elements are listed in [83]. Enterprise-specific information elements must be known by the collector in order to decode them. Length describes the length of the information element and the value 65535 is reserved to indicate varying information element length. The field specifier format is presented in Table 14 [81]

Table 14: Field Specifier Format [81]

0	1	2	3	4
E	Information element identifier		Length	
Enterprise number				

A FlowSet consists of a set header, one or multiple records and optional padding. The set header has two fields and is shared by NetFlow v9 and IPFIX. FlowSet ID identifies the used set and can have values from 0 to 65535. However, IPFIX has renamed the field as Set ID. NetFlow v9 uses values 0 and 1 to identify Template and Options Template Sets respectively whereas for IPFIX 0 and 1 are not used. Furthermore, IPFIX uses values 2 and 3 identify Template and Options Template Sets. Values from 4 to 255 are reserved for future use and values over 255 are used with Data Sets. The Length field contains the FlowSet length and it includes the header, all records and the optional padding. Its purpose is to identify the boundaries of different FlowSets. [80][81]

Template FlowSets informs the collector on which fields the exporter will send in Data FlowSets. The Template FlowSet may contain one or multiple Template records. Table 15 presents an example of a Template FlowSet. Numbers in the first column represent bytes in the datagram. FlowSet ID of zero defines that the FlowSet is a Template FlowSet. Length specifies the total length of the FlowSet. Template ID distinguishes the particular template and Templates are matched to Data FlowSets using this ID. The value of Template ID can vary from 256 to 65535. Field Count specifies the number of fields in the particular Template. NetFlow v9 and IPFIX define the payload of the record differently. NetFlow v9 utilizes type and length fields, whereas IPFIX utilizes field specifiers presented in Table 14. Field Type contain a number identifying the particular field and Field Length describes the length of the field. [80][81]

Table 15: NetFlow v9 Template FlowSet [80]

0 1 2	3 4
FlowSet ID = 0	Length
Template ID	Count
Field Type 1	Field Length 1
...	...
Field Type N	Field Length N
Template ID M	Count
Field Type 1	Field Length 1
...	...
Field Type N	Field Length N

Table 16: IPFIX Template FlowSet [81]

0 1 2	3 4
FlowSet ID = 0	Length
Template ID	Count
Field Specifier	
...	
Field Specifier	

Data FlowSets may contain either data or options data records. However, IPFIX does not specify options data records as it utilizes data records to provide similar information. Regardless of the record type, Data FlowSets begin with a FlowSet header. A Data record consists of field values in the order described by the used template. Options data records are formed in a similar manner. However, they contain a scope value before the option values. Both record types may contain padding. Examples of data and options data records are presented in Tables 17 and 18. Contrary to the examples, field lengths are not restricted to two or four bytes as presented, instead the field length is determined by the template. [80][81]

Table 17: Data FlowSet with a single Data record [80][81]

0 1 2	3 4
FlowSet ID	Length
Field 1	
Field 2	
...	
Field N	Padding

Table 18: Data FlowSet with multiple Options Data records [80][81]

0 1 2	3 4
FlowSet ID	Length
Record 1 Scope 1	Record 1 Option Field 1
Record 1 Option Field 2	...
Record 2 Scope 1	Record 2 Option Field 1
Record 2 Option Field 2	...
Record N Scope 1	Record N Option Field 1
Record N Option Field 2	...
Record N Option Field M	Padding

Options Template FlowSets have a FlowSet ID of 1. Options Template Records provide a capability for delivering additional information to the collectors that could not be sent in Flow records. IPFIX options template format differs from the NetFlow v9 format by utilizing the field specifiers instead of type and length. The record header fields format is shared, however the header fields have slightly different definitions. For NetFlow v9 fields `option scope length` and `option length` describe the number of scope and option fields respectively. Whereas IPFIX utilizes `field count` and `scope field count` fields to describe the total number of fields and the number of `scope` fields in the record. Format are presented in Tables 19 and 20. [80][81]

Table 19: NetFlow v9 Options Template record format [80]

0 1 2	3 4
Template ID	Option Scope Length
Option Length	Scope 1 Field Type
Scope 1 Field Length	...
Scope N Field Length	Option 1 Field Type
Option 1 Field Length	...
Option M Field Length	Padding

Table 20: IPFIX Options template record format [81]

0 1 2 3 4

Template ID	Field count
Scope field count	Field specifier 1
Field specifier 1 continued	Field specifier 2
Field specifier 2 continued	....
Field specifier N continued	Padding

Security

From a security perspective, NetFlow version 9 does not have any mechanisms to ensure the confidentiality, authenticity nor availability of the information. Therefore, an attacker positioned in the network path between exporter and collector can examine, modify and spoof flow contents. These security concerns originate from the design, as it was believed that NetFlow would be only used in trusted networks with exporters and collectors closely located within the network. [80] IPFIX combat these issues by introducing DTLS over SCTP transport mode, which provides reliable transport and integrity protection and authentication services. However, DTLS has a man-in-the-middle vulnerability that allows an attacker to remove messages from the stream unnoticed. [81]

3.7 Emerging protocols

Network monitoring protocols are constantly evolving and new protocols are being developed. Therefore, it is essential to introduce these emerging protocols. The protocols presented here are relatively new and implement by certain vendors. However, they do not yet have the same status as more established protocols such as NETCONF or SNMP.

gRPC Remote Procedure Call

gRPC Remote Procedure Call (gRPC) is an open-source remote procedure call framework created by Google in 2015 and is currently being incubated by Cloud Native Computing Foundation (CNCF) [84][85]. While gRPC is used for network management and monitoring by some vendors, it is a generic RPC framework that can be used for various purposes. Thus the information available using gRPC depends on the vendor implementation. However, common models such as MIB or YANG may be used.

gRPC services and message formats are described in a programming language agnostic manner and the client and server interfaces are generated from this description. The gRPC services may be either synchronous or asynchronous depending on the use case. Additionally, gRPC supports message streaming, where one or both parties of the connection send multiple messages on a single RPC call. This feature allows streaming telemetry data for network monitoring purposes. Typically the services and message formats are defined using protocol buffers, which is Google's language- and platform-agnostic mechanism for serializing structural data [86]. However, other Interface Definition Languages (IDLs) may be used. [85]

gRPC uses Hypertext Transfer Protocol Version 2 (HTTP/2) as the main transport protocol. Additionally, the network operations can be handled using Chromium Network Stack (Cronet) or in-process. HTTP/2 is an optimized evolution of the Hypertext Transfer Protocol version 1.1 (HTTP/1.1). However, they are only compatible on a high-level and only the semantics such as status codes and methods are shared by HTTP/1.1 and HTTP/2 whereas message format and encoding are different, for example. HTTP/2 is a binary protocol and supports multiple requests using a single connection whereas HTTP/1.1 is a textual protocol requiring a unique connection for every request. Therefore, HTTP/2 has better performance and smaller overhead than HTTP/1.1. [87] Chromium Network Stack is Google's networking stack used in mobile devices and the Chrome browser [88]. It supports HTTP/1.1, HTTP/2 and QUICK protocols. In-process transport allows connecting to the gRPC service from the same process, thus enabling local development of a gRPC service without network interactions. [85]

gRPC Network Management Interface

gRPC Network Management Interface (gNMI) is network management protocol capable of configuring network devices as well as retrieving telemetry data from the devices. It is based on the gRPC framework by introducing services providing network management capabilities. The protocol development is overseen by Google and OpenConfig. gNMI can utilize the YANG-models used with NETCONF and RESTCONF protocols, however, other interface describing languages (IDLs) may be used with gNMI. [13]

gNMI has four services specified: `capabilities`, `get`, `set` and `subscribe`. `Capabilities`-service is used as a similar handshake as NETCONF's hello message to exchange capabilities. The client and the target exchange information regarding gNMI version and supported data models as well as supported data type options. Currently gNMI supports JSON, JSON_IETF, bytes, ASCII and protobuf encoding options. JSON_IETF differs from the JSON by requiring conformance to serialization rules defined in RFC7159 whereas JSON only requires the content have a valid JSON structure. Bytes encoding utilizes a raw byte sequence and its format depends on the implementation. ASCII indicates that the content is ASCII encoded text. However, the message format is not defined by gNMI and has to be defined by the implementation. Protobuf encoding indicates that the content is formatted according to Google's protocol buffer specification. `Get`-operation provides means to retrieve parts of the target configuration by sending a `GetRequest`-message containing information regarding the part of interest. The target sends the configuration items in a `GetResponse`-message, which terminates the RPC-operation. However, the `get`-service is only designed for retrieving small parts of the configuration and if larger sets are required, subscribe-service should be used. `Set`-operation allows altering the target configuration by sending a `SetRequest`-message. The target configuration can be either replaced, updated or removed. The target responds with a `SetResponse`-message, which fields indicate success of each individual operation. Additionally, if a single operation fails, the target must rollback other changes made with the same `SetRequest`. `Subscribe`-operation is used to subscribe to telemetry updates sent by the target using a `SubscribeRequest`-message. Subscriptions can be one of the following types: one-off, poll or stream. One-off is used to query data just once. Polling retrieves data periodically and stream send notifications to the client based defined trigger conditions.

As gNMI is built on top of gRPC, HTTP/2 is used as a main transport, which provides the authentication and encryption services. gNMI requires that all connections utilize TLS and in any conditions the connection must not fallback to plain-text communications. Mutual authentication is provided by X.509 certificates in a similar manner as in RESTCONF. In addition to authentication, authorization is required in order to perform any actions. However, telemetry messages are not authorized. Actions are authorized using username and password, which are carried in RPC-messages. [89]

sFlow

Inmon Corporation developed sFlow for monitoring network traffic passing routers and switches. It is designed to monitor network traffic at gigabit and higher speeds and can handle tens of thousands of sFlow agents using a single collector. Additionally, the cost of implementation is claimed to be low. sFlow utilizes a similar exporter-collector design as NetFlow and IPFIX with the exception of calling exporter an Agent. The recorded network flows are sampled before sending the samples to the collector. sFlow supports two sampling methods: statistical packed-based and time-based. However, the sampling can be disabled and thus accounting all observed packets. Statistical packet based sampling obtains every Nth packet to be sampled, where the maximum value of N depends on the hardware capabilities. If N is one, then all packets are sampled and if N is zero, sampling is disabled. When using time-based sampling, a sample of interface counters is taken at a configured interval. [90]

Simple Network Management Protocol (SNMP) is used to control sFlow Agents and a MIB module is provided to describe the control interfaces. The provided module is compliant to SMIv2 and can be translated to SMIv1, if SNMP version 1 is required. [90]

Collected flow datagrams are sent to collectors using User Datagram Protocol (UDP), which offers multiple benefits such as lower memory footprint and robustness in the event of network overload. UDP port 6343 is defined by the sFlow MIB to be used for sending and receiving sFlow datagrams. As UDP is an unreliable transport protocol, lost packets are not detected. However, this does only affect marginally the accuracy of the obtained measurements and the most severe effect is that the effective sampling rate is slightly reduced. sFlow utilizes External Data

Representation (XDR) format as its message encoding, which provides more compact messages and simplified encoding and decoding when compared to Abstract Syntax Notation One (ASN.1) [90]

3.8 Summary

This chapter discussed the importance of network monitoring and introduced state-of-the-art network monitoring protocols as well as certain emerging protocols. The information provided by the presented network monitoring protocols varies from flow information to configuration options. Additionally, certain protocols depend on external data models for defining capabilities and available information. Therefore, these protocols may require more processing in order to extract information from the data provided by the protocol. By understanding the capabilities of the presented protocols, the protocols to be used with the system presented in this thesis may be chosen to complement each other.

4 Requirements and design

This chapter defines the requirements for a harmonized network monitoring system, introduces the architecture of the system and describes the development of the presented system as well as introduces the software components of the presented system.

4.1 Requirements

The main requirement for this system is monitoring heterogeneous fleet of network devices and storing received data into a format that is suitable for advanced processing. This main goal has been divided into the following requirements:

1. The system should be able to ingest data from state-of-the-art network monitoring protocols
2. Store the ingested data in an efficient database, that supports a large number of simultaneous write-operations
3. Process the stored data into the chosen data model format
4. The system should be extendable and modular in order to add support for additional protocols and devices in the future
5. Support additional processing of the stored data

The amount of network telemetry data can be enormous, therefore the database should be efficient in order to handle large data sets. Having the ingested data stored in a common data model eases further processing and analytics as the data can be accessed using common and well-know characteristics. Modularity allows the system to support heterogeneous fleet of devices, and can be extended as new devices or vendors are introduced to the network. Additionally, it allows keeping the system lightweight, as redundant components can be disabled or removed. The modularity also supports the additional processing of the stored data, as data processing pipelines can be integrated to the system introduced in this thesis.

Based on the literature review conducted in Chapter 3, the following protocols were chosen for the proof of concept implementation:

- SNMP version 3
- NetFlow version 9
- IPFIX
- NETCONF
- Syslog

These protocols provide means to monitor network traffic, device conditions and software running on the network devices.

4.2 Technologies

The system presented in this thesis builds on various technologies and software components. Therefore, in order to understand the architecture, it is important to introduce the used technologies. The technologies and components used are open-source software. This along with suitable licensing allows contributing to the development of the technologies. Additionally, the technologies are available for free.

Telegraf

Telegraf is a server agent capable of collecting and sending metrics from various sources such as databases, network devices and Internet of things (IoT) sensors. Telegraf has a plugin based architecture and therefore can be easily expanded. This thesis updated Telegraf's SNMP Trap plugin to support SNMPv3 and has been sent to be included into future releases of Telegraf [91]. The plugins are divided into four categories: input, output, aggregator and processor. Input plugins are used to collect measurements from various sources such as SNMP, Syslog, HTTP or CPU statistics. Output plugins are used to write data to message pipelines or databases, for example. In addition to the output plugins, Telegraf supports multiple data formats such as Carbon2, Graphite, InfluxDB Line Protocol, JSON, ServiceNow Metrics and SplunkMetric [92]–[97]. Aggregator plugins create new metrics based on collected metrics. For example, Aggregator plugins can merge metrics and create histograms. Processor plugins modify the metrics as they are collected by converting field types, processing values using regular expressions or renaming tags, for example. [94]

InfluxDB

InfluDB is a time-series database designed for high performance write and query operations and can handle millions of writes per second. It provides an SQL-like query language, InfluxQL, which supports regular expressions, arithmetic operations and functions specific to time-series data. Use cases for InfluxDB include DevOps monitoring, real-time analytics and internet-of-things (IOT) sensor data. InfluxDB stores data as points, where a single point contains a timestamp, a measurement, tags and fields. Measurement is a name describing the stored data and SQL database equivalent would be a table. Fields store the actual data and may contain strings, floats, integers or booleans. Tags are used to store metadata and are always stored as strings regardless of the actual value. Tags are indexed and therefore recommended as using tags makes database queries faster. InfluxDB accepts data in various formats with the InfluxDB line protocol being the preferred one. The other supported formats are collectd, Graphite, OpenTSDB and Prometheus [98][93][99][100]. Supported protocols are HTTP or UDP depending on the format. [94]

Kapacitor

Kapacitor is a real-time stream processing engine created by InfluxData and it is a part of InfluxData's TICK-stack (Telegraf, InfluxDB, Chronograf, Kapacitor). It can process stream data in real-time, provide alerting services and its interfaces allow integrating additional data processing engines. Kapacitor uses its own TICKscript to control processing of incoming data and can be augmented with user-defined functions (UDFs). Documentation and examples for writing UDFs in Golang and Python are provided, however UDFs can be written in any programming language. [94]

Docker

Docker is a virtualization solution that leverages containers. Containers share the kernel with the host operating system, thus having less overhead thank virtual machines. Docker utilizes several kernel-level features wrapping them in a simple interface. These features include Linux Containers (LXC), control groups (cgroups) and namespaces. This allows Docker to isolate Containers from each other and the host machine. [101]

Apache Kafka

Apache Kafka is a distributed streaming platform, that allows publishing and subscribing to message streams. The message streams are categorized as topics allowing clients to subscribe only to relevant information. Kafka brokers require one or more Apache Zookeeper instances to coordinate the brokers. Zookeeper also stores metadata for the Kafka instances, however, it does not contribute to message processing. [102]

vFlow

Verizon's vFlow is a flow collector supporting IPFIX, sFlow, NetFlow v5 and v9 protocols. It can produce messages based on decoded flow packets to multiple outputs including Kafka, NATS and NSQ using JSON format. However, it does not decode NetFlow v9 nor IPFIX fields to field names. It can be scaled horizontally and has good performance as it is able to process up to thousand IPFIX packets per second on a computer with two processor cores and 64 megabytes of memory. [103]

4.3 Data model

Based on the literature review conducted in Chapter 2, Apache Spot Open Data Model (ODM) was chosen. The model has a simple flat structure that is suitable for time-series databases as they are not designed for hierarchical data. Additionally, the model provides concise implementation details and has fields suitable for storing network state data. The data model fields were analyzed in order to identify which fields are likely used to query data and thus should be indexed in the database. This step was crucial as InfluxDB separates indexed tags and non-indexed values. The identified field were marked to be processed as tags by the processing component.

Device identification

An important aspect in network monitoring is the ability to identify the origins of the events and log messages. In relatively static networks, IP-addresses can be used to distinguish devices. However, not all protocols send IP-address as part of the protocol message. For example, Syslog prefers hostname over IP-address as described in Section 3.3. Therefore, device configurations should be carefully reviewed. The presented system uses IP-addresses as the primary identification and hostnames are used to complement the address information if available.

Deviations

The implemented data model has few deviations from the data model specification. If a field that is not known to the data model is encountered, it is stored in the database as is instead of utilizing the `additional_attrs` field. Using this field would require parsing the unknown fields into a JSON format string, which cannot be indexed by InfluxDB. This decision allows more effective queries for the database and requires less parsing. Additionally, InfluxDB only supports four data types, whereas ODM supports eight. For most part, this does not cause any issues as bigintegers and longs can be stored as InfluxDB integers, for example. However, InfluxDB does not support binary data or Maps, these data types could be stored as strings by base64 encoding the binary data and dumping Maps to JSON-strings. However, the data model only uses binary type for user profile pictures and thus it does not affect the use case presented in this thesis. Additionally, IP-addresses are stored as strings instead of integers, as having the addresses in a human readable format increases usability.

4.4 Architecture and implementation

The architecture of this system is divided to data ingestion, storage and processing. An additional component can be utilized for visualizing data, but it is not a mandatory part of the system. The system architecture is visualized in Figure 6. On a networking perspective, the system is divided into two domains: internal and external. Internal network consists of storage and processing and external contains only ingestion. The rationale behind this division is to protect the collected data, while exposing the endpoints used to collect the data.

InfluxData's TICK-stack (Telegraf, InfluxDB, Chronograf, Kapacitor) was chosen as a base for the system as it has many suitable properties such as being open source, actively developed and extensible [94].

The proposed system utilizes Docker as the virtualization solution and every software component is run in a separate container. Containers responsible for ingesting data utilize Docker's host networking in contrast to the database and processing

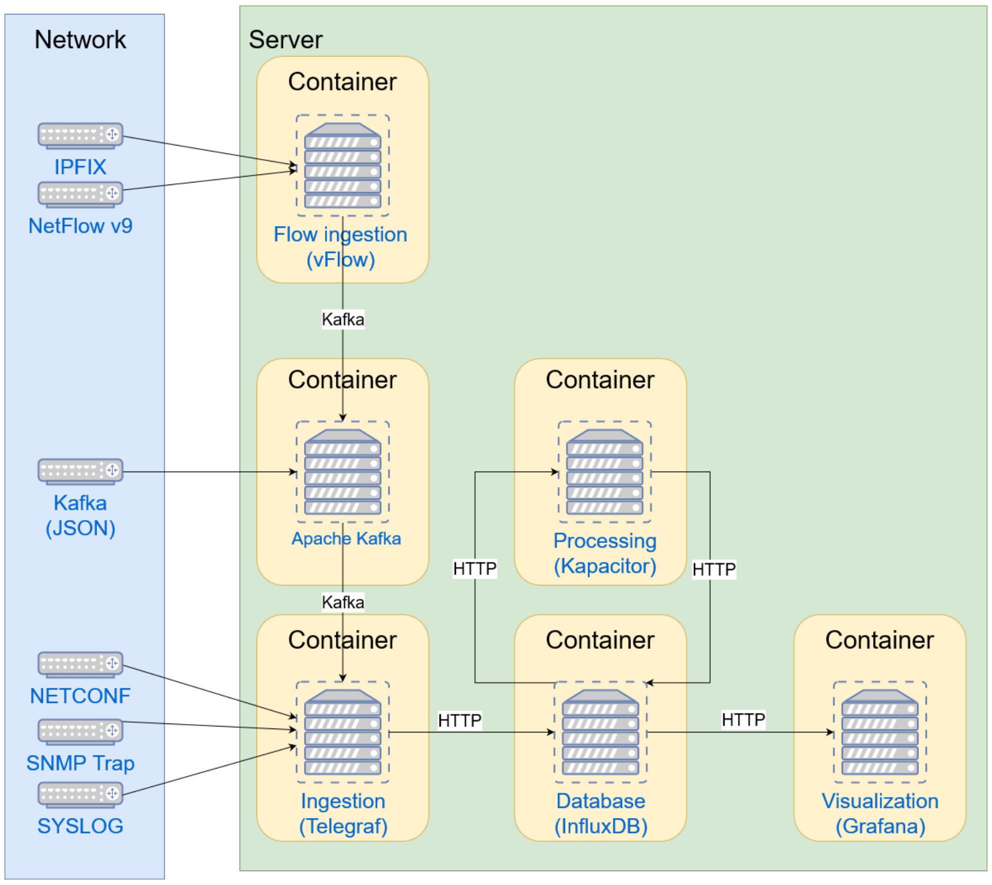

Figure 6: System architecture

components utilizing Docker's bridge-networking. Docker's bridge-networking employs network address translation (NAT), which translates source and destination addresses in order to host multiple containers behind a single IP-address. As some of the network monitoring protocols utilize UDP as a transport layer protocol and its source IP header to identify host, NAT introduces issues as this header value points to the Docker host and not to the actual client.

Figure 7 presents the sequence diagram of the system. A network device sends data to the system by using either a supported flow protocol, Apache Kafka message stream or a supported monitoring protocol. Supported flow protocols are IPFIX and NetFlow v9 and of monitoring protocols Syslog, SNMPv3 and NETCONF are supported. If a flow protocol is used, then vFlow ingest the flow data, which is then forwarded to Telegraf through the Apache Kafka message broker. If the network device supports Kafka message streams with JSON format, data can be send through Kafka to Telegraf. However, the preferable method is to use a supported monitoring protocol and send data directly to Telegraf. Telegraf writes the raw data to the database, from which Kapacitor reads the raw data and processes it before writing it back to the database. The protocols used to transmit data within the system are presented in Figure 6. Most interactions between components are carried over

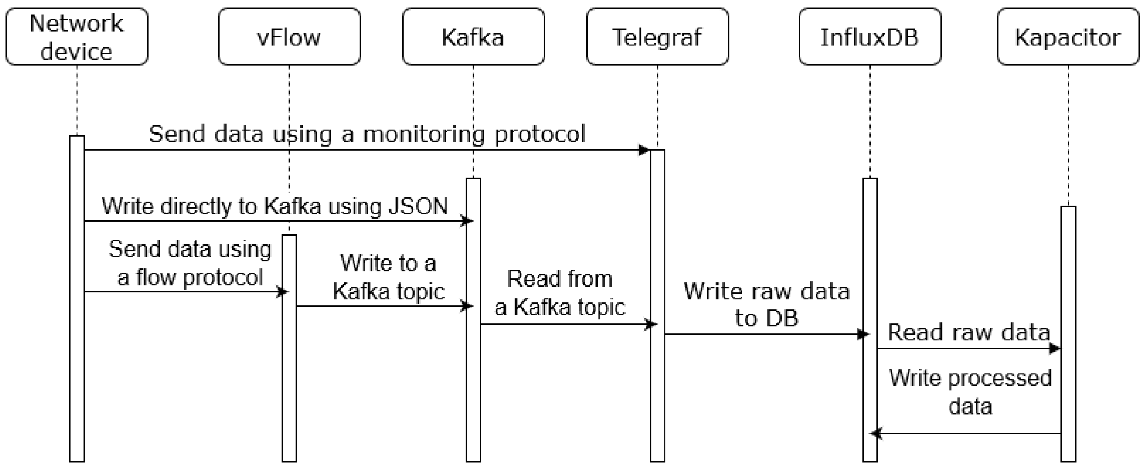

Figure 7: System sequence diagram

HTTP, however, all interactions with the Apache Kafka component use the Kafka's own binary protocol.

Docker volumes are used to provide persistent storage to containers requiring stateful operation such as the database. The volumes persist even if the associated containers are removed.

Data collection

Influxdata's Telegraf was chosen as the component for receiving data from network monitoring protocols and other sources. To complement Telegraf's capabilities, Apache Kafka was introduced to the system to ingest data from sources that are not supported by Telegraf, but support Kafka. Currently Telegraf does not support any flow protocols and therefore Verizon's vFlow was introduced to the system. It is configured to capture NetFlow version 9 and IPFIX messages and forward them to the Kafka broker in JSON format. Telegraf then ingests these protocol messages from the corresponding Kafka topics. The system described in this thesis outputs the ingested data using InfluxDB Line Protocol as InfluxDB was chosen as the data storage. Before inserting collected data into the database, Telegraf converts all fields into lowercase as processing the data in to the data model becomes easier.

Processing

Kapacitor utilizes several user-defined functions in order to parse NetFlow and IPFIX messages into a human-readable format and to map the incoming data to the data model. The user-defined function responsible for mapping the data into the data model reads the incoming data sets, translates fields to data model fields and converts data to a correct type, such as an integer. Any fields that cannot be mapped to the data model, are stored with the original field names in order to provide better usability for the data. The relations between the data model and incoming data are defined in a YAML file (Yet Another Markup Language). At runtime the YAML file is read by the processing component and imported as a Python dictionary. Data model field names are used as keys in the dictionary and key-related values are also dictionaries. The sub-dictionaries contain four attributes: `is_tag`, `type`, `fields` and `values`. Attribute type determines the data type as which the field is saved to the database. The types used in this thesis deviate from the Apache Spot Open Data Model, as Python does not make distinction between integers and longs, for example. However, this does not affect the system, as Python internally handles large integers as longs. `values` is a list of field values that should be translated. For example, if a fields "protocol" is observed with a value of zero, then the value inserted to the database would be "HOPOPT" as presented in Listing 19. Attribute `fields` lists the incoming fields that should be mapped to the data model field in question. Listing 19 presents an excerpt from the data model YAML file. The data model format data is stored in a different database than the raw data. This allows utilizing the raw data for other purposes if necessary. Even though processed and raw data reside in different databases, both databases are located in the same database instance. However, the system architecture allows adding an additional InfluxDB instance to separate raw and parsed data sets or to provide redundancy.

```
msg:
  is_tag: false
  type: int
  fields:
  - INFO_MSG
  - message
n_proto:
  is_tag: false
  type: str
  fields:
  - protocol
  - protocolIdentifier
  values:
    "0": HOPOPT
    "1": ICMP
    "2": IGMP
    "3": GGP
    "4": IPv4
    "5": ST
    "6": TCP
```

Listing 19: Excerpt from the data model YAML file

Storage

The system uses two databases located in a single instance of InfluxDB. One database is used to store the raw data from Telegraf and another is used to store the data model formatted data. InfluxDB runs on a default configuration and therefore performance optimizations and security improvements could be achieved by reviewing the configuration. For example, the Time-Series Index (TSI) could be reviewed as

InfluxData states that it should be a better option for a large number of unique time-series measurements than the default Time-Structure Merge Tree [94].

4.5 Summary

This chapter discussed the requirements and the implementation details of the presented system as well as presented the used software components. The presented system supports various modern network monitoring protocols with the option of extending the system. Additionally, the system stores the collected data in a common data model format. However, certain alterations to the data model had to be made in order to use it in a more effective manner.

5 Evaluation

This chapter focuses on three aspects: protocol overhead, value provided by use of a common data model and the effectiveness of the chosen data model. Additionally, possible improvements are discussed as well as the fulfillment of requirements.

5.1 Protocol overhead

Network protocols typically transport information other than application data, which is used to carry session information or metadata, for example. This non-payload data is referred as protocol overhead as it does not contribute to the actual carried information. This section focuses on the overhead of the application layer. Therefore, overhead caused by other layers of the networking stack is not reviewed as many of the protocols support multiple transports. More specifically, this section focuses on the overheads of a single protocol operation. Depending on the protocol, a single protocol operation may require multiple messages to be exchanged in order to complete the operation. As the amount of fields transported in a single message can vary significantly, this thesis evaluates only the absolute overhead instead of relative.

Understanding the overhead of various network monitoring protocols is important as network devices may locate in constricted networks were bandwidth is scarce. In such networks, protocols with less overhead are preferable as they do not waste the limited network capacity.

NetFlow v9 & IPFIX

NetFlow v9 and IPFIX are one-way protocols and therefore only the exporter sends messages. Thus a single protocol operation can be completed in a single message. In order to a transmit a single field of information using either of these protocols, a template has to be sent in addition to the payload. Therefore, as a minimum a protocol header, a template FlowSet and a Data FlowSet are required to form an initial message.

NetFlow v9 header has two fields with the length of two bytes and four fields of four bytes. Therefore, the header introduces an overhead of 20 bytes. As IPFIX dropped the sysUpTime field, the header is smaller with the length of 16 bytes.

Data FlowSet headers contain SetID and Length fields contributing two bytes each towards overhead, thus resulting in a 4 byte overhead. Additionally, the data FlowSet and record formats are similar for NetFlow v9 and IPFIX and thus the overhead caused by them is equivalent for both protocols. Besides the header, data FlowSets only contain actual payload data. Template FlowSets are required to be sent before data FlowSets in order for the collector to decode the data FlowSets. Therefore, the template FlowSets are counted as overhead. NetFlow v9 and IPFIX share the eight byte template FlowSet header. However, the template format differs for both protocols. NetFlow v9 utilizes field type and length fields to describe the incoming data fields. Both the type and length fields are two bytes. IPFIX uses its own field specifier format, which can be either 4 or 8 bytes in length, depending on whether generic or vendor proprietary fields are described. Vendor proprietary being eight bytes in length. Therefore, the overhead caused by template FlowSets is at least 12 bytes for both NetFlow v9 and IPFIX, when a single generic field is described in the template FlowSet. When combining packet header, data FlowSet header and template FlowSet overheads, an overhead of 36 bytes is required for NetFlow v9 and 32 byte overhead for IPFIX in order to transmit a single field from the exporter to the collector. However, this overhead applies only for the initial message as the collector caches the template and thus the template can be omitted from the future messages. Therefore, the overhead for the subsequent messages utilizing the same template is 24 bytes for NetFlow v9 and 20 bytes for IPFIX. Additionally, the field length can vary from a single byte to a large number, thus affecting the overhead to payload ratio.

SNMP

SNMP supports both solicited and unsolicited messages. For unsolicited messages such as SNMPv2-Traps, no state information is required and thus the overhead comes from the message headers. As for solicited messages, the message format is the same as for unsolicited, therefore, the overhead is effectively twice the of the unsolicited case.

SNMP version 3 packet format presented in Section 3.2 consists of a ten field header and a payload. SNMP uses basic encoding rules (BER) to encode the values into a packet. A SNMPv3 packet has byte `30` as the first byte indicating a SEQUENCE followed by bytes indicating SEQUENCE length. As SNMP implementations are required to support messages at least 484 octets in length and seven bits can only represent values up to 128, at least two bytes are required to represent the length of a SNMP message. Due BER, only seven bits of a byte are available for indicating the length as the last bit is used to indicate whether multiple bytes are required to represent the length value.

The version field requires three bytes, due to first two bytes being the BER type and length and the actual value being a single byte integer. HeaderData is a SEQUENCE containing msgID, msgMaxSize, msgFlags and msgSecurityModel fields. The encoding of a SEQUENCE requires two bytes, the BER type and length. The msgID field may contain integer values from 0 to 2147483647, and thus can be represented with four bytes or less, depending on the value. Therefore, the space requirement varies from three to six bytes, when including the BER type and length. The msgMaxSize value varies from 484 to 2147483647 and thus requires minimum of two bytes and maximum of four bytes for the value itself. Including the type and length bytes, the space requirement and thus the overhead is four to six bytes. The msgFlags field utilizes a single byte to indicate several flags, and the space requirement is three bytes when including the BER type and length. Even though msgSecurityModel may contain values from 1 to 2147483647, current specifications only reference values that could be presented using a single byte as described in Section 3.2. Therefore, the msgSecurityModel field requires three bytes. The msgSecurityParameters is an octet string and the length depends on the used security model.

The user-based security model defines the msgSecurityParameters octet string as a SEQUENCE. Therefore, the msgSecurityParameters field starts with a octet string type and length followed by a SEQUENCE type and length resulting in at least four byte overhead. However, this depends on the length of the security parameters. The msgAuthoritativeEngineID field has the type of an octet string and as the snmpEngineID is defined to be an octet string with the length ranging from 5 to 32 bytes [46], the overhead caused by msgAuthoritativeEngineID field is 7 to 34 bytes when accounting the BER type and length. The msgAuthoritativeEngineBoots and msgAuthoritativeEngineTime are specified to contain integer values within range from 0 to 2147483647. This range can be represented using a four byte or smaller integer value. Along with the BER type and length indicators, these fields cause an overhead of 3 to 6 bytes each. The msgUserName field is an octet string that is defined to contain up to 32 characters therefore causing overhead of 3 to 34 bytes. Even though the specification allows the msgUserName field to be empty, some implementations do not allow empty values. One example of such software is the open source net-snmp. If the security setting is no authentication and no privacy, msgAuthenticationParameters and msgPrivacyParameters are empty. Thus the overhead caused by these two fields is 4 bytes as the type and length of these fields have to be specified. If authentication or privacy is used, then the lengths of these fields depend on the used encryption and authentication protocols. In total the overhead caused by the user-based security model is at least 23 bytes.

The scopedPDU is typed as a SEQUENCE and consists of ContextEngineId, ContextName and PDU fields. In order to describe the scopedPDU length, at least two bytes are required. Additionally, the upper limit depends on the maximum number of variable-bindings. Therefore, in addition to its fields, the scopedPDU contributes at least three bytes towards the protocol overhead. The ContextEngineId and ContextName fields are typed as octet strings and their values depend on the implementation [46]. However, these parameters can be blank and therefore both only affecting the overhead by two bytes due to the BER type and value indicators.

The PDU identifier is an implicit identifier [47], which is encoded to the first byte of the PDU. This yields an overhead of two bytes as determined by the BER. Request-id, error-index and error-status fields are typed as integers. Of these fields, the error-status field may have values ranging from 1 to 18 and thus can be represented with a single byte. However, request-id and error-index may contain values up to 214783647. Therefore, the space requirement ranges from one to four bytes. After taking the BER type and length bytes into account, these three field yield an overhead of nine to fifteen bytes. The variable-bindings field contains the actual payload and therefore does not affect the overhead. Adding the two bytes from the beginning of the PDU, the actual overhead of the PDU is 11 to 17 bytes.

The different fields and their overheads are presented in Table 21. The minimum overhead for a non-acknowledged message such as a trap is at least 60 bytes. If a protocol operation requiring a response is used, then the overhead is effectively doubled as the message format is similar for different protocol operations. Therefore, for a get operation the overhead is at least 120 bytes.

Table 21: SNMPv3 message encoding and overhead

BER Type	Length (octets)	Container	Field	Overhead (bytes)
30	<length>	SNMPv3Message	SEQUENCE	> 3
02	01		msgVersion	3
30	<length>	HeaderData	SEQUENCE	> 2
02	01-04		msgID	3-6
02	02-04		msgMaxSize	4-6
04	01		msgFlags	3
02	01		msgSecurityModel	3
04	<length>	msgSecurityParameters	OCTET STRING	> 2
30	<length>	UsmSecurityParameters	SEQUENCE	> 2
04	05-32		msgAuthoritativeEngineID	7-34
02	01-04		msgAuthoritativeEngineBoots	3-6
02	01-04		msgAuthoritativeEngineTime	3-6
04	00-32		msgUserName	> 2
04	<length>		msgAuthenticationParameters	> 2
04	<length>		msgPrivacyParameters	> 2
30	<length>	scopedPDU	SEQUENCE	> 2
04	<length>		ContextEngineId	> 2
04	<length>		ContextName	> 2
04	>01		PDU identifier	> 2
02	<length>		request-id	> 3
02	<length>		error-index	> 3
02	01		error-status	> 2
		Sum		> 60

Syslog

Syslog does not utilize any application level protocol headers and thus the payload contains only the message format presented in Section 3.3. Furthermore, Syslog does not require any state information and only unsolicited messages are supported. Therefore, Syslog does not have any application layer protocol overhead and the only overhead affecting Syslog results from transport, network and data link layer protocols. Additionally, Syslog messages are unsolicited, which reduces the lower layer overhead as the information is not requested nor acknowledged. However, it should be noted that if TCP is used, then the message is acknowledged on the transport layer, but not on the application layer.

As the Syslog protocol is a textual protocol, the protocol message itself could be optimized further by utilizing binary encoding. For example, the priority indicator requires three to five characters depending on the actual priority value and thus requires three to five bytes. The priority value can vary from 0 to 999, which can be represented using a two byte integer. Therefore, depending on the priority value, one or multiple bytes could be saved. Additionally, this applies to other fields that could be represented as integers. However, varying string fields such as hostname and message would require additional information to be transmitted in order to detect boundaries of these fields.

NETCONF

Regarding NETCONF, this thesis considers the protocol operations along with their attributes to be part of the payload, whereas everything else is considered as overhead.

NETCONF requires both parties of a connection to send a hello message in order to exchange capabilities before any protocol operations can be carried out. This ensures the compatibility between a client and a server. The hello message is similar for both parties, however, the server sends an additional session-id parameter in the message. As a minimum, NETCONF requires that the base capability is advertised and thus the minimum overhead can be acquired by evaluating a message with only the base capability. Listings 20 and 21 present minimal hello messages. Assuming the session-id is a one digit integer and only the base capability is advertised,

```xml
<?xml version="1.0" encoding="UTF-8"?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <capabilities>
        <capability>urn:ietf:params:netconf:base:1.1</capability>
    </capabilities>
    <session-id>2</session-id>
</hello>
```

Listing 20: Minimal server hello

```xml
<?xml version="1.0" encoding="UTF-8"?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <capabilities>
        <capability>urn:ietf:params:netconf:base:1.1</capability>
    </capabilities>
</hello>
```

Listing 21: Minimal client hello

the minimum hello message length for the server is 175 bytes and 149 bytes for the client, which results in a minimum of 324 byte overhead for the connection. This evaluating assumes that the XML content is minified by removing redundant whitespace characters. Describing additional capabilities increases the overhead.

The client protocol operation consists of xml and rpc tags and the protocol operation along with optional parameters. Of these elements, the xml and rpc tags are considered as overhead. Furthermore, the minimal message-id parameter would a single digit integer as described in Section 3.4. Therefore, the overhead is for the client initiated protocol operation is 112 bytes. Listing 22 presents an example of the get-config protocol operation, which queries the running config.

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
    <get-config>
        <source>
            <running/>
        </source>
    </get-config>
</rpc>
```

Listing 22: Get-config example

The server reply operation resembles the client operation, as the main difference being an rpc-reply tag instead of an rpc tag. Therefore, the overhead is similar to the client operation and the actual overhead is 124 bytes. Listing 23 presents a server-reply message. The contents of the data tag are redacted in order to improve readability as they are not relevant for the overhead analysis due to being part of the payload.

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
    <data>
        ...response data...
    </data>
</rpc-reply>
```

Listing 23: Server response to get-config protocol operation

Combining the overhead caused by the hello messages, the client protocol operation and the server response, an overhead of 560 bytes is acquired. Additionally, the connection teardown requires an additional protocol operation, which can be counted entirely as overhead as it does not contribute to the payload. Listing 24 presents the messages sent by the client and the server in order to close the connection. The lengths of these messages are 128 and 129 bytes respectively, thus resulting in a 257 byte overhead. When taking this into account, the overall overhead accumulates to 817 bytes. However, the hello and close-connection messages have to be only sent once per connection and if only the actual protocol operation and its response are examined, the overhead is 236 bytes. Therefore, the longevity of the connection affects the overhead-to-payload ratio. However, the protocol operations and responses still contain a significant amount of overhead.

NETCONF supports unsolicited notification from the server to the client after the connection has been initiated. An example of such notification is presented in Listing 25. As the rpc and xml tags were considered as overhead for the RPC operations, the xml and notification tags are considered as overhead for notifications.

All tags contained within the notification-tags are considered to be part of the message payload. After minimizing the message content, the parts considered as overhead are 125 bytes in length thus yielding an overhead less than half that of a get operation, if counting both request and response.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<rpc
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="2">
    <close-session/>
</rpc>

<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="2">
    <ok/>
</rpc-reply>
```

Listing 24: Close session message exchange

```xml
<?xml version="1.0" encoding="UTF-8"?>
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification
   :1.0">
    <eventTime>2007-07-08T00:10:00Z</eventTime>
    <event xmlns="http://example.com/event/1.0">
    <eventClass>state</eventClass>
    <reportingEntity>
        <card>Ethernet0</card>
    </reportingEntity>
    <operState>enabled</operState>
    </event>
</notification>
```

Listing 25: Notification example [73]

Summary

Of the evaluated protocols, NETCONF and Syslog are textual protocol whereas IPFIX, NetFlow v9 and SNMPv3 use binary encoding. The large overhead for NETCONF points out the main caveat of textual protocols; it is ineffective to use textual conventions to represent data. However, the protocol messages are human-readable without additional tools whereas binary protocols require additional tooling in order to decode protocol messages into a human readable format. As this thesis defined overhead as data contributing to the actual carried information, the Syslog protocol does not have any overhead due to not having any headers. However, the non-existent overhead does not imply that the protocol is effective as Syslog requires a large number of bytes to represent timestamps or other information that could be encoded into a small number of bytes. The summarized results of this section are presented in Table 22.

Syslog, IPFIX and NetFlow v9 protocols utilize unsolicited messages, therefore requiring only one protocol message to be sent to the receiver. Furthermore, this one message often fits into a single network packet, thus minimizing overhead. NETCONF requires multiple messages in order to initialize the connection, perform the required operations and close the connection. This increases the overhead as many of these messages do not contribute to the payload. SNMPv3 and NETCONF have protocol operations for both solicited and unsolicited messages and thus the overhead depends on the used operation. Additionally, based on the performed analysis, unsolicited messages are recommended they typically reduce the overhead by a significant amount. Of the used protocols, NETCONF and SNMPv3 are much more complex than Syslog, IPFIX and NetFlow v9 as they can be also used to configure network devices in addition to providing state information.

Table 22: Protocol overheads

Protocol	Overhead for a complete solicited message exchange (bytes)	Solicited message overhead after initializing the connection (bytes)	Unsolicited message overhead after initializing the connection (bytes)
Syslog	0	N/A	0
IPFIX	32	N/A	16
NetFlow v9	36	N/A	20
SNMPv3	120	120	60
NETCONF	817	257	125

Little research has focused on the protocol overheads of the Syslog, IPFIX and NetFlow v9 protocols. Regarding NetFlow v9, processing overhead has been researched, however, the research did not consider the network protocol itself [104]. SNMP's efficiency and features have been evaluated in multiple studies [105]–[109]. However, the overhead of the SNMP protocol itself was not discussed in these studies and most of the studies utilized either SNMP version 2 or did not specify the used version. However, the results obtained by this thesis can be compared to message lengths presented in several studies [107][106].

In the 2006 study by Marinov and Schönwälder [107], SNMPv3 was used with user-based security model and context engineID discovery requiring four packets and a total of 668 bytes in order to carry out a single protocol operation. This leads to approximately 167 bytes per message and when taking into account the 42 bytes of Ethernet, IP and UDP headers and the 60 byte overhead evaluated by this thesis, the amount of data is 66 bytes. However, the actual data amount in the study might be less, as the user-based security model parameters affect the overhead. Nonetheless, the figures obtained by this thesis are consistent with the study and thus are reliable.

In another study conducted in 2010 by Yu, Shayman and Rozenblit [106], SNMP and NETCONF were compared . The version of the SNMP was not stated, however, based on the bytes transferred the probable version is 2c. The study found out that in order to obtain a single data object, NETCONF requires three packets and 1460 bytes. This thesis reviewed NETCONF to have an overhead 817 of bytes and thus the payload would be 643 bytes. This number seems to be a reasonable size for a large configuration element when taking the XML encoding into account.

Overall, the results obtained by this thesis appear to be dependable when comparing to related previous research. Additionally, this thesis found out that little research has been carried out on network monitoring protocol overheads. Therefore, additional research could be dedicated to evaluating these protocols on the aspect of protocol overhead.

5.2 Information value

The value provided by having data mapped to a common data model, is evaluated by assessing the quantity of fields per protocol that can be mapped to the data model. This section only evaluates the quantity of the fields that can be mapped as the presented system is designed for generic network monitoring and the information of interest may vary between users of the system.

As flow-protocols have structured and predefined fields, it is relatively simple to evaluate which fields can be mapped to the data model. However, not all of these fields will be filled in every case. Therefore, packets from network devices are evaluated in order to identify how the fields are used in real world scenarios.

Protocols such as NETCONF and SNMP rely on external data models for the message fields, and thus it is not feasible or even possible to evaluate all fields. Therefore, this thesis selects few definitions relevant to network monitoring and evaluates those in order to obtain an estimate.

In addition to payload data, protocol headers are evaluated if they provide additional information regarding the protocol and its operation. For example, SNMP's msgVersion header can be mapped to ODM's snmp_version-field.

IPFIX & NetFlow v9

NetFlow v9 has 127 fields of which 24 are reserved fields, 8 are vendor proprietary and 2 deprecated. These 34 fields are excluded from the evaluation as they do not provide any useful information. In addition to the data fields, the six NetFlow v9 header fields should be also considered. Therefore, 99 fields contain usable information. Of these fields, 19.19% could be mapped to the ODM. However, if additional fields introduced with IPFIX are taken into account, the number of fields increases to 461 and the percentage of mapped fields drops to 8.03%. IPFIX has 492 fields defined, of which one is reserved, two are deprecated and 29 reserved for NetFlow v9 compatibility. Additionally, the header contains five fields in contrasts to NetFlow's six. Therefore, 465 fields contain useful information. As many of the fields are shared with NetFlow v9, the percentage of mapped fields resembles that of NetFlow v9. For IPFIX, 7.74% of the fields could be mapped to the ODM. However, not every field is sent with every NetFlow message, therefore real world examples must be examined in order to evaluate the practical value of usable fields. This thesis uses Ubiquiti's EdgeRouter X and Cisco's CSR1000V routers as NetFlow v9 exporters in order to obtain protocol messages. However, due to high variance in possible router configurations, this thesis evaluates only simple use cases in order to obtain a baseline. Both routers were configured as a simple routers with only simple static routes. Therefore, use cases with Open Shortest Path First (OSPF), Border Gateway Protocol (BGP) or Routing Information Protocol (RIP) protocols, for example, might yield different results. Protocol packets were captured and decoded using the system presented in this thesis and the fields included in these packets were evaluated against the protocol mappings. For CSR1000v, the "original-input" record scheme was used. The used record scheme determines the exported fields. Furthermore, the router allows user defined record schemes, therefore only fields of interest can be exported in order to reduce the amount of data. Thus the number of fields which could be mapped to the ODM depends on the used record scheme. Edgerouter X sent 29 and 26 fields for NetFlow v9 and IPFIX respectively and this number includes the protocol header fields. Of these fields, 41.38% for NetFlow v9 and 42.31% for IPFIX could be mapped to the ODM. With CSR1000V using "original-input", 52.00% of the incoming 26 fields could be mapped to the ODM. Furthermore, 50.00% of the 24 IPFIX fields could be mapped. When taking all the presented cases into account, the average percentage of fields that could be mapped to the ODM was 31.52%. Table 23 presents the acquired results.

Table 23: Percentage of mapped fields in different scenarios

Case	Percentage of mapped fields
NetFlow v9	19.19
NetFlow v9 with IPFIX extensions	8.03
IPFIX	7.74
Edgerouter NetFlow v9	41.38
Edgerouter IPFIX	42.31
CSR1000V IPFIX	50.00
CSR1000V NetFlow v9	52.00
Average	31.52

SNMP

SNMP uses MIBs to represent information, therefore the information value provided by this system depends on the used MIBs. This section evaluates several MIBs in order to evaluate SNMP's information value. The following MIBs are evaluated: `BGP4-MIB`, `IF-MIB`, `IP-MIB`, `OSPF-MIB`, `RIPv2-MIB` and `SNMPv2-MIB`. These MIBs provide information regarding interfaces, common routing protocols and SNMP itself. However, in contrast to NETCONF, SNMP does not differentiate state and configuration data. Therefore, as this analysis takes all fields into account, configuration fields are also evaluated. In addition to the MIBs, the information transmitted in the SNMPv3 header is evaluated and the used security model is user-based security model. Thus the following header fields are evaluated: `msgVersion`, `msgID`, `msgMaxSize`, `msgFlags`, `msgSecurityModel`, `msgAuthoritativeEngineID`, `msgAuthoritativeEngineBoots`, `msgAuthoritativeEngineTime`, `msgUserName`, `msgAuthenticationParameters` and `msgPrivacyParameters`.

Of the 692 reviewed fields, 42 could be mapped to the Apache Spot Open Data Model, which is 6.36 %. The evaluated MIBs provide very explicit information on routing protocols and interfaces, which is difficult to map to a broad and generic data model. Additionally, the MIBs contain fields dedicated for configuration data in addition to state data.

Syslog

Syslog contains only a few fields, which are well documented, therefore mapping values to the data model is simple and most fields could be mapped. However, the structured-data field may contain vendor specific fields and thus the mapping of the structured-data depends on the information source. Only the facility-field could not be mapped to the ODM, if structured-data is excluded. Table 24 presents the Syslog fields and their ODM counterparts.

Table 24: Syslog ODM mapping

Syslog field	ODM field
version	version
timestamp	event_time
hostname	dvc_host
app-name	app
procid	proc
msgid	type
severity	severity
facility	–
structured-data	–
msg	msg

If structured-data is ignored then 88.89% of the incoming fields could be mapped to the ODM. If structured-data is included as a field that cannot be mapped, then the amount is reduced to 80.00%. As structured data may contain vendor-specific information, this thesis evaluates only IANA assigned items defined in [56][45][110][111][112] in order to obtain an estimate of the number of fields that could be mapped to the ODM. When including IANA structured data items, the percentage of fields which could be mapped to the ODM reduces to 26%. The main cause is the information type, as ODM is not designed to contain information such as pre-congestion notifications. Moreover, SNMP structured data contain only raw information, which requires additional parsing in order to map it to the ODM.

As Syslog relies on the free form msg-field, more accurate information could be gathered by parsing the field. However, this requires knowledge on the incoming messages as otherwise false information could be obtained by improperly parsing messages. Listing 26 presents a Syslog message ingested by Telegraf, that could be parsed for additional information. From the example, field "service" could be filled

```
syslog,appname=systemd,facility=daemon,hostname=machine.testlab.
    invalid,severity=warning version=1i,severity_code=4i,
    facility_code=3i,timestamp=1591608708661616000i,procid="1",
    message=" nginx.service: Failed with result 'exit-code'."
    1591608708661756994
```

Listing 26: Telegram ingested Syslog message

with "nginx", "state" with "Failed" and "code" with "exit-code", thus increasing the information extracted from the Syslog message. As more fields of the data model are filled, the information value is also increased as categorization and searching becomes quicker and easier.

Table 25: Percentage of mapped fields using Syslog

Case	Percentage of mapped fields
Without structured-data	89%
With structured-data as non-mappable	80%
IANA structured-data	26%
Average	65%

NETCONF

In order to evaluate the information value provided by NETCONF, YANG data models must be evaluated as NETCONF utilizes these models to provide state and configuration information. Due to the large number of YANG models defined by various organizations and vendors, only a limited sample is reviewed in this thesis. Furthermore, as the scope of this thesis is network monitoring, only YANG elements with the "config false" attribute are evaluated. These elements provides data regarding the state of the network device, whereas "config true" elements contain configuration data. Additionally, deprecated fields are excluded from the evaluation. As RESTCONF utilizes same data models as NETCONF, analysis performed in this section applies to RESTCONF also. IETF and OpenConfig models were chosen as both provide generic models that device vendors can implemented and enable vendor agnostic operation. However, vendor specific models may provide more detailed information than generic models. This thesis evaluates the following YANG-models: `ietf-ip`, `ietf-interfaces`, `ietf-routing`, `ietf-system`, `ietf-network`, `ietf-hardware`, `openconfig-interfaces`, `openconfig-network-instance-l3`, `openconfig-if-ip` and `openconfig-system`. These models were chosen on the basis that they provide core information regarding network device operation and network state.

Table 26 presents the results obtained by evaluating the YANG-models and mapping the fields to the ODM. Of the evaluated fields, 20.81% could be mapped to the ODM. For the IETF models combined, 30.88% of the field could be mapped whereas for OpenConfig models only 15.50%. When comparing IETF and OpenConfig models, OpenConfig models contained more detailed information whereas IETF models are more generic and contained higher-level information.

Table 26: Percentage of mapped fields for NETCONF

Case	Percentage of mapped fields	Number of fields
IETF	30.88%	68
OpenConfig	15.50%	129
Combined	20.81 %	197

Summary

The common data model provides more value as it allows retrieval of all events related to a single network element using simple queries. If all protocols were stored using separate data models, either complex queries has to be used or it might not even be possible to retrieve all events related to a single network element. Table 27 presents the summary of the evaluated protocols in different cases. From the results it can be deduced that when the granularity of the protocol increases, decreases the number of fields that can be mapped to the data model. When comparing the results between protocols, the variance is significant as the highest percentage being 65.00% and the lowest being 6.36%. However, the Syslog results can be seen as an outlier, as it provides significantly less detailed information than the other protocols. Also, the flow protocols had seemingly low numbers, however, when looking at the real world examples of simple cases, the number grew significantly. The difference between NETCONF and SNMP is quite large, when taking into account that both were evaluated using IETF models. However, this difference might be explained by the fact that only fields containing state data were evaluated for NETCONF whereas for SNMP all fields were evaluated. SNMP does not separate configuration data from state data in MIB definitions and therefore all fields had to be evaluated.

5.3 Comparison

This section compares the chosen Apache Spot Open Data Model (ODM) against Elasticsearch's Elastic Common Schema (ECS) in order to verify the suitability of the data model as well as to detect caveats and possible improvements. The data

Table 27: Percentage of mapped fields using different protocols

Case	Percentage of mapped fields
NetFlow v9	19.19
NetFlow v9 with IPFIX extensions	8.03
IPFIX	7.74
SNMPv3	6.36
NETCONF	20.81
Syslog	65.00
Mean	20.81
Median	13.28

models are compared using the following criteria: structure, suitability for network monitoring, extensibility, data format and state of the project.

As the ODM has a flat data model without any hierarchy within entries, it is more suitable for time-series databases, which typically do not support hierarchical data. Furthermore, the ECS is designed as a hierarchical model and thus imposes requirements on the used database. However, the hierarchical structure can be represented with dots implying the levels of hierarchy in field names as it is done in ECS' documentation. As the presented system is built on a time-series database, a flat data model is more suitable than a hierarchical model.

The ODM has nine data types: long, int, string, float, bigint, boolean, map, double and binary. The ECS has five different types: string, long, float, boolean and object. However, the InfluxDB database chosen for the proposed system stores values using four different types: double, integer, boolean and string. Of the types used by the data models, binary data is the only type that cannot be casted directly to any of the types. However, the only field using binary data is ODM's user profile picture, which is not relevant for this thesis. Additionally, the database does not support maps nor objects, however, these types can be stored as strings, which hand prevents effective indexing of those fields. Overall, both of the models utilize similar data types that are for the most part compatible with the database. As the differences are minimal, the result for the data format criterion is a tie.

The suitability for network monitoring was evaluated by assessing the number of fields suitable for storing network data. The Elastic Common Schema has 199 fields suitable for this purpose and the Apache Spot Open Data Model has 227. However, the quality of stored data is similar between these data models. Both can store information regarding network flows, their origins and destinations, events, protocols and generic host information. However, the ODM has fields for certain networks protocols such as SNMP, FTP, SMTP and SSH, which the ECS does not have. Therefore, the Apache Spot ODM can be regarded as a better solution for network monitoring than the ECS.

Both data models provide a mechanism for extending the models beyond the initially defined fields. The ODM utilizes the `additional_attrs` field to contain fields that cannot be mapped to the data model whereas the ECS stores additional data with custom field names. Therefore, the ECS provides better extensibility as the data model provides an easy and accessible way of extending the model. Additionally, in the context of a flat time-series database, custom fields can be indexed more easily than a JSON object used in the ODM. Both data models provide guidelines on contributing to the development of the model and accept third-party contributions.

The Elastic Common Schema and the Apache Spot Open Data Model are open source projects and therefore anyone can contribute to the development. However, the Elastic Common Schema is also maintained by Elastic whereas Apache Spot Open Data Model is supported by Apache but not directly maintained by it. GitHub branch SPOT-181_ODM on GitHub was last updated on 11.9.2019 [17]. This branch contains the development of the Apache Spot Open Data Model. The Elastic Common Schema GitHub repository was last updated on 29.7.2020 and latest release was made on 7.3.2020 [22]. Therefore, the Elastic Common Schema is more actively maintained and its documentation is more mature than Apache Spot Open Data Model's as the ODM is missing some fields definitions. On these grounds, the Elastic Common Schema is a better option of these two alternatives.

Table 28 presents the summary of the performed analysis. It was found that Apache Spot Open Data Model is suitable for network monitoring and the better alternative of the reviewed models. However, the Elastic Common Schema is a capable alternative and in some cases a better solution. The most weight was given on suitability to network monitoring and structure as these factors affected the development of the presented system more than extensibility or other assessed factors.

Table 28: Summary of data model analysis

Criteria	Better option
Structure	ODM
Data format	Tie
Suitability for network monitoring	ODM
Extensibility	ECS
State of the project	ECS
Overall	ODM

5.4 Summary

This chapter reviewed the overheads of network monitoring protocols, evaluated the information value provided by the presented system and the suitability of the used data model as well as verified the system to fulfil the requirements imposed on to it. The overhead of network monitoring protocols and the information value provided by the protocols were found to vary significantly between protocols. However, the information value provided by the system was significant enough to justify the system.

The system was found to fulfil the imposed requirements and development areas were identified. The requirements set for the system proposed in this thesis were:

1. The system should be able to ingest data from state-of-the-art network monitoring protocols

2. Store the ingested data in an efficient database, that supports a large number of simultaneous write-operations

3. Process the stored data into chosen data model format

4. The system should be extendable and modular in order to add support for additional protocols and devices in the future

5. Support additional processing of the stored data

The proposed system can ingest data from SNMP, NETCONF, IPFIX, NetFlow v9 and Syslog. Additional protocols are supported by the data ingestion components, however, they were not implemented as a part of this thesis. Thus the requirement number 1 is fulfilled. The chosen time-series database can store vast amounts of data points and provides effective means to index and retrieve data thus fulfilling the second requirement. The proposed system and the proof-of-concept implementation can process the incoming data using Kapacitor into the chosen data model and can be modified with ease to support additional models and databases. The architecture and structure of the system is modular, allowing changes to single components and also extensions to the system. Additionally, Telegraf utilizes a plugin based design, allowing users to develop their own plugins for additional protocols. The chosen Kapacitor data processing component allows additional processing and forwarding the data to third-party data processing engines. Therefore, requirement number five is fulfilled.

By identifying what information is required and what can be provided by various devices, an end user may extend the data model, either by utilizing the `additional_attrs` -field and parsing the incoming data into a common field or by extending the data model itself. Additionally, free form fields could parsed using dedicated parsers in order to acquire more detailed information. Furthermore, some protocol message fields cannot be mapped to the ODM without parsing the field. Listing 27 presents an example, where a field contains information regarding bytes transferred during connection along with source and destination hosts. However, this information is stored in the field name besides the byte count and therefore cannot be mapped to the ODM easily. This field could be parsed to format presented in Table 29. However, parsing these messages requires knowledge on the incoming fields and thus implementing these dedicated parsers is a time consuming task.

```
snmp_trap,mib=CISCOTRAP-MIB,name=tcpConnectionClose,oid
    =.1.3.6.1.4.1.9.0.1,source=192.168.122.121,version=2c
    loctcpconnoutbytes.192.168.122.121.22.192.168.122.1.33828=31067i
    ,tslineuser.2="cisco",sysuptimeinstance=29886i,tslinesestype
    .2.1=6i,tcpconnstate.192.168.122.121.22.192.168.122.1.33828=5i,
    loctcpconnelapsed.192.168.122.121.22.192.168.122.1.33828=22702i,
    loctcpconninbytes.192.168.122.121.22.192.168.122.1.33828=18393i
    1591609191956165531
```

Listing 27: TcpConnectionClose SNMP Trap message

Table 29: Example of a parsed SNMP field

ODM field	Value
dst_ip4	192.168.122.121
dst_port	22
src_ip4	192.168.122.1
src_port	33828
in_bytes	18393i

6 Conclusions

This thesis has designed and developed a novel network monitoring system, which harmonizes incoming data into a common data model. The proposed system consists of three main components: a data ingestion, a processing and a time-series database. The architecture of the system is modular allowing changes to only one of the components without affecting the others. The system accepts data from multiple network monitoring protocols and stores it to the database using a common data model. The data ingestion component accepts data from several protocols and formats the data to InfluxDB line protocol in order to insert it to the database. The raw data is read from the database by the processing component, which processes it into the chosen data model before inserting it back to the database.

The presented system was developed by first outlining the requirements for the system, then a literature review was conducted regarding state-of-the-art networking monitoring protocols and data models. The defined requirements along with the knowledge attained from the literature review were used to design the architecture of the system. Additionally, various database, data processing and data ingestion solutions were researched in order to choose suitable components for the system.

The proposed system was found to fulfil the set requirements. The results indicate that the chosen data model is suitable for network monitoring and provides additional value when used with state of the art network monitoring protocols. Approximately 21% of the collected information could be harmonizing to the chosen data model. Additionally, the median of the results was 13.28%. However, the variance of the results was significant as the lowest value was 6.36% and the highest was 65.00%. The variance originates from the different design perspectives of the protocols and the evaluated MIB and YANG data models as only a subset of these models could be evaluated in the context of this thesis. The protocol overheads also had significant variance as the results ranged from 0 to 817 bytes. The cause for these results was the research setting, where the entire message exchange was evaluated. Some of the protocols require capability exchanges and other stateful messages in order to transmit monitoring information and such messages were considered as overhead. When comparing only the actual protocol operations, binary protocols and protocols requiring no state information were found as the most effective. However, the Syslog protocol was found to be an outlier as it is a textual protocol and had the least overhead.

The system presented in this thesis only collects and stores the monitoring data and it does not process or utilize the data further, thus only acting as a data lake. Therefore, a data analytics component could be added to the system in order to detect error conditions and other anomalies in the network. This would increase the value provided by the system as the manual work of analyzing the data would be reduced. The data processing component responsible for mapping incoming data into the data model is written in Python. However, Python is not an efficient programming language and thus the system might not be able to handle high loads. Therefore, the component could be written in Golang in order to increase the component's performance. Additionally, even though the system is designed for monitoring network state and network conditions, the modular structure allows altering the use case. For example, by substituting or extending the data model, the system could be used to monitor cyber security incidents or device configuration changes.

References

[1] K. Finnerty, S. Fullick, H. Motha, M. Button, V. Wang, and N. S. Jayesh, "Cyber security breaches survey 2019: Statistical release", p. 66, 4th Mar. 2019. [Online]. Available: https://researchportal.port.ac.uk/portal/en/publications/cyber-security-breaches-survey-2019(293aa2ac-0a83-4a2e-a6e8-d2f2252810a2).html (visited on 14. 03. 2020).

[2] FireEye. (3rd Mar. 2020). M-trends cyber security trends, FireEye. Library Catalog: www.fireeye.com, [Online]. Available: https://www.fireeye.com/current-threats/annual-threat-report/mtrends.html (visited on 05. 05. 2020).

[3] F. Attila, "Monitoring Modern Networks with Network Telemetry", Bachelor's thesis, Budapest University of Technology and Economics, Budapest, 2019, 76 pp. [Online]. Available: https://www.hte.hu/documents/11801/4608612/Attila_FABIAN_BSc.pdf (visited on 15. 03. 2020).

[4] C. Hosmer, *Defending IoT Infrastructures with the Raspberry Pi: Monitoring and Detecting Nefarious Behavior in Real Time*, 1st edition. Apress, 2018, 193 pp., ISBN: 978-1-4842-3699-4.

[5] R. Olups, *Zabbix Network Monitoring - Second Edition*, 2nd edition. Packt Publishing, 2016, 754 pp., ISBN: 978-1-78216-128-8. [Online]. Available: https://www.safaribooksonline.com/library/view/-/9781782161288/.

[6] Z. Chen, G. Xu, V. Mahalingam, L. Ge, J. Nguyen, W. Yu, and C. Lu, "A cloud computing based network monitoring and threat detection system for critical infrastructures", *Big Data Research*, Special Issue on Big Data from Networking Perspective, vol. 3, pp. 10–23, 1st Apr. 2016, ISSN: 2214-5796. DOI: 10.1016/j.bdr.2015.11.002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2214579615000520 (visited on 24. 02. 2020).

[7] H. Hui and L. Bai, "Research and development of centrlized log management system based on syslog protocol", in *Proceedings of the 2012 International Conference on Cybernetics and Informatics*, S. Zhong, Ed., vol. 163, Series Title: Lecture Notes in Electrical Engineering, New York, NY: Springer New York, 2014, pp. 1949–1956, ISBN: 978-1-4614-3872-4. DOI: 10.1007/978-1-4614-3872-4_249. [Online]. Available: http://link.springer.com/10.1007/978-1-4614-3872-4_249 (visited on 14. 05. 2020).

[8] V. Mehta, *Icinga Network Monitoring*, 1st edition. Packt Publishing, 21st Nov. 2013, 118 pp., ISBN: 978-1-78328-229-6.

[9] H.-J. Liao, C.-H. Richard Lin, Y.-C. Lin, and K.-Y. Tung, "Intrusion detection system: A comprehensive review", *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 16–24, Jan. 2013, ISSN: 10848045. DOI: 10.1016/j.jnca.2012.09.004. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1084804512001944 (visited on 24. 02. 2020).

[10] G. Halleen, *Security Monitoring with Cisco Security MARS*, 1st edition. Cisco Press, 2007, 336 pp., ISBN: 978-1-58705-270-5.

[11] D. Zobel, “Monitoring applications and services with network monitoring”, Paessler, Dec. 2013, p. 8. [Online]. Available: https://hlassets.paessler.com/common/files/pdf/whitepaper/application-monitoring_en.pdf (visited on 16. 04. 2020).

[12] T. Dondich, *Network Monitoring with Nagios*, 1st edition. O’Reilly Media, Inc, 2006, 62 pp., ISBN: 0-596-52819-1. [Online]. Available: https://learning.oreilly.com/library/view/network-monitoring-with/0596528191/.

[13] J. Clarke, *Network Programmability with YANG: The Structure of Network Automation with YANG, NETCONF, RESTCONF, and gNMI*, 1st edition. Addison-Wesley Professional, 2019, 512 pp., ISBN: 978-0-13-518039-6. [Online]. Available: https://www.safaribooksonline.com/library/view/-/9780135180471/.

[14] A. Pras and J. Schoenwaelder, “On the difference between information models and data models”, Internet Engineering Task Force, RFC3444, Jan. 2003, RFC3444. DOI: 10.17487/rfc3444. [Online]. Available: https://www.rfc-editor.org/info/rfc3444 (visited on 14. 05. 2020).

[15] F. Strauss and J. Schoenwaelder, “SMIng - next generation structure of management information”, Internet Engineering Task Force, RFC3780, May 2004, RFC3780. DOI: 10.17487/rfc3780. [Online]. Available: https://www.rfc-editor.org/info/rfc3780 (visited on 15. 07. 2020).

[16] M. Bjorklund, “YANG - a data modeling language for the network configuration protocol (NETCONF)”, Internet Engineering Task Force, RFC6020, Oct. 2010, RFC6020. DOI: 10.17487/rfc6020. [Online]. Available: https://www.rfc-editor.org/info/rfc6020 (visited on 08. 05. 2020).

[17] Apache, *Apache/incubator-spot*, 11th Sep. 2019. [Online]. Available: https://github.com/apache/incubator-spot/tree/SPOT-181_ODM (visited on 06. 03. 2020).

[18] D. Vohra, *Practical Hadoop Ecosystem: A Definitive Guide to Hadoop-Related Frameworks and Tools*, 1st edition. Apress, 2016, 444 pp., ISBN: 978-1-4842-2198-3. [Online]. Available: https://www.safaribooksonline.com/library/view/-/9781484221990/.

[19] Splunk. (28th Jun. 2019). How to use the CIM data model reference tables - splunk documentation, splunk.com, [Online]. Available: https://docs.splunk.com/Documentation/CIM/4.15.0/User/Howtousethesereferencetables (visited on 20. 05. 2020).

[20] ——, (14th May 2020). Overview of metrics - splunk documentation, [Online]. Available: https://docs.splunk.com/Documentation/Splunk/8.0.5/Metrics/Overview (visited on 27. 07. 2020).

[21] ——, (18th Oct. 2019). Use the CIM to normalize data at search time - splunk documentation, splunk.com, [Online]. Available: https://docs.splunk.com/Documentation/CIM/4.15.0/User/UsetheCIMtonormalizedataatsearchtime#7._.28Optional.29_Extend_the_CIM_definition_with_custom_fields (visited on 26. 06. 2020).

[22] Elastic, *Elastic/ecs*, original-date: 2018-05-24, 18th May 2020. [Online]. Available: https://github.com/elastic/ecs (visited on 20. 05. 2020).

[23] ——, (2020). Field datatypes | elasticsearch reference | elastic. Library Catalog: www.elastic.co, [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping-types.html (visited on 20. 05. 2020).

[24] M. Settle and M. Martin. (13th Feb. 2019). Elastic common schema (ECS): The common event model for elasticsearch, Elastic Blog. Library Catalog: www.elastic.co, [Online]. Available: https://www.elastic.co/blog/introducing-the-elastic-common-schema (visited on 20. 05. 2020).

[25] Elastic. (5th Mar. 2020). Elastic common schema (ECS) reference [1.5] | elastic, Elastic. Library Catalog: www.elastic.co, [Online]. Available: https://www.elastic.co/guide/en/ecs/1.5/index.html (visited on 20. 05. 2020).

[26] J. Komárková, M. Husák, M. Laštovička, and D. Tovarňák, "CRUSOE: Data model for cyber situational awareness", in *Proceedings of the 13th International Conference on Availability, Reliability and Security - ARES 2018*, Hamburg, Germany: ACM Press, 2018, pp. 1–10, ISBN: 978-1-4503-6448-5. DOI: 10.1145/3230833.3232798. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3230833.3232798 (visited on 06. 03. 2020).

[27] CSIRT-MU, *CSIRT-MU/CRUSOE-data-model*, original-date: 2018-06-28T06:49:06Z, 18th Sep. 2018. [Online]. Available: https://github.com/CSIRT-MU/CRUSOE-Data-Model (visited on 18. 05. 2020).

[28] J. P. Jeong, C. Chung, S. Hares, L. Xia, and H. Birkholz, "I2nsf NSF monitoring YANG data model", Internet Engineering Task Force, Internet-Draft draft-ietf-i2nsf-nsf-monitoring-data-model-03, 2020, p. 78. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-i2nsf-nsf-monitoring-data-model-03 (visited on 14. 05. 2020).

[29] D. Lopez, E. Lopez, L. Dunbar, J. Strassner, and R. Kumar, "Framework for interface to network security functions", Internet Engineering Task Force, RFC8329, Feb. 2018, RFC8329. DOI: 10.17487/RFC8329. [Online]. Available: https://www.rfc-editor.org/info/rfc8329 (visited on 28. 05. 2020).

[30] S. Hares, D. Lopez, M. Zarny, C. Jacquenet, R. Kumar, and J. Jeong, "Interface to network security functions (i2nsf): Problem statement and use cases", Internet Engineering Task Force, RFC8192, Jul. 2017, RFC8192. DOI: 10.17487/RFC8192. [Online]. Available: https://www.rfc-editor.org/info/rfc8192 (visited on 28. 05. 2020).

[31] D. R. Mauro, *Essential SNMP*, 2nd ed, D. R. Mauro and K. J. Schmidt, Eds. Sebastopol (CA): O'Reilly, 2005, Publication Title: Help for System and Network Administrators, ISBN: 0-596-00840-6.

[32] K. McCloghrie, D. Perkins, and J. Schoenwaelder, "Structure of management information version 2 (SMIv2)", Internet Engineering Task Force, RFC2578, Apr. 1999, RFC2578. DOI: 10.17487/rfc2578. [Online]. Available: https://www.rfc-editor.org/info/rfc2578 (visited on 17. 04. 2020).

[33] R. Frye, D. Levi, S. Routhier, and B. Wijnen, "Coexistence between version 1, version 2, and version 3 of the internet-standard network management framework", Internet Engineering Task Force, RFC3584, Aug. 2003, RFC3584. DOI: 10.17487/rfc3584. [Online]. Available: https://www.rfc-editor.org/info/rfc3584 (visited on 17. 04. 2020).

[34] K. McCloghrie, D. Perkins, and J. Schoenwaelder, "Conformance statements for SMIv2", Internet Engineering Task Force, RFC2580, Apr. 1999, RFC2580. DOI: 10.17487/rfc2580. [Online]. Available: https://www.rfc-editor.org/info/rfc2580 (visited on 17. 07. 2020).

[35] M. Daniele and J. Schoenwaelder, "Textual conventions for transport addresses", Internet Engineering Task Force, RFC3419, Dec. 2002, RFC3419. DOI: 10.17487/rfc3419. [Online]. Available: https://www.rfc-editor.org/info/rfc3419 (visited on 03. 06. 2020).

[36] R. Presuhn, "Transport mappings for the simple network management protocol (SNMP)", Internet Engineering Task Force, RFC3417, Dec. 2002, RFC3417. DOI: 10.17487/rfc3417. [Online]. Available: https://www.rfc-editor.org/info/rfc3417 (visited on 03. 06. 2020).

[37] J. Schoenwaelder, "Simple network management protocol over transmission control protocol transport mapping", Internet Engineering Task Force, RFC3430, Dec. 2002, RFC3430. DOI: 10.17487/rfc3430. [Online]. Available: https://www.rfc-editor.org/info/rfc3430 (visited on 03. 06. 2020).

[38] J. Schoenwaelder and T. Jeffree, "Simple network management protocol (SNMP) over IEEE 802 networks", Internet Engineering Task Force, RFC4789, Nov. 2006, RFC4789. DOI: 10.17487/rfc4789. [Online]. Available: https://www.rfc-editor.org/info/rfc4789 (visited on 03. 06. 2020).

[39] J. Salowey, D. Harrington, and W. Hardaker, "Secure shell transport model for the simple network management protocol (SNMP)", Internet Engineering Task Force, RFC5592, Jun. 2009, RFC5592. DOI: 10.17487/rfc5592. [Online]. Available: https://www.rfc-editor.org/info/rfc5592 (visited on 03. 06. 2020).

[40] W. Hardaker, "Transport layer security (TLS) transport model for the simple network management protocol (SNMP)", Internet Engineering Task Force, RFC6353, Jul. 2011, RFC6353. DOI: 10.17487/rfc6353. [Online]. Available: https://www.rfc-editor.org/info/rfc6353 (visited on 25. 05. 2020).

[41] J. Case, D. Harrington, R. Presuhn, and B. Wijnen, “Message processing and dispatching for the simple network management protocol (SNMP)”, Internet Engineering Task Force, RFC3412, Dec. 2002, RFC3412. DOI: 10.17487/rfc3412. [Online]. Available: https://www.rfc-editor.org/info/rfc3412 (visited on 02. 06. 2020).

[42] U. Blumenthal and B. Wijnen, “User-based security model (USM) for version 3 of the simple network management protocol (SNMPv3)”, Internet Engineering Task Force, RFC3414, Dec. 2002, RFC3414. DOI: 10.17487/rfc3414. [Online]. Available: https://www.rfc-editor.org/info/rfc3414 (visited on 17. 04. 2020).

[43] D. Harrington and J. Schoenwaelder, “Transport subsystem for the simple network management protocol (SNMP)”, Internet Engineering Task Force, RFC5590, Jun. 2009, RFC5590. DOI: 10.17487/rfc5590. [Online]. Available: https://www.rfc-editor.org/info/rfc5590 (visited on 10. 04. 2020).

[44] ITU-T, “ITU-t rec. x.690 (08/2015) information technology – ASN.1 encoding rules: Specification of basic encoding rules (BER), canonical encoding rules (CER) and distinguished encoding rules (DER)”, Aug. 2015. DOI: 11.1002/1000/12483. [Online]. Available: https://www.itu.int/ITU-T/recommendations/rec.aspx?rec=12483&lang=en (visited on 20. 06. 2020).

[45] V. Marinov and J. Schoenwaelder, “Mapping simple network management protocol (SNMP) notifications to SYSLOG messages”, Internet Engineering Task Force, RFC5675, Oct. 2009, RFC5675. DOI: 10.17487/rfc5675. [Online]. Available: https://www.rfc-editor.org/info/rfc5675 (visited on 23. 06. 2020).

[46] D. Harrington, R. Presuhn, and B. Wijnen, “An architecture for describing simple network management protocol (SNMP) management frameworks”, Internet Engineering Task Force, RFC3411, Dec. 2002, RFC3411. DOI: 10.17487/rfc3411. [Online]. Available: https://www.rfc-editor.org/info/rfc3411 (visited on 01. 04. 2020).

[47] R. Presuhn, “Version 2 of the protocol operations for the simple network management protocol (SNMP)”, Internet Engineering Task Force, RFC3416, Dec. 2002, RFC3416. DOI: 10.17487/rfc3416. [Online]. Available: https://www.rfc-editor.org/info/rfc3416 (visited on 02. 06. 2020).

[48] ——, “Management information base (MIB) for the simple network management protocol (SNMP)”, Internet Engineering Task Force, RFC3418, Dec. 2002, RFC3418. DOI: 10.17487/rfc3418. [Online]. Available: https://www.rfc-editor.org/info/rfc3418 (visited on 03. 06. 2020).

[49] W. Hardaker and D. Harrington, “Transport security model for the simple network management protocol (SNMP)”, Internet Engineering Task Force, RFC5591, Jun. 2009, RFC5591. DOI: 10.17487/rfc5591. [Online]. Available: https://www.rfc-editor.org/info/rfc5591 (visited on 25. 05. 2020).

[50] J. Merkle and M. Lochter, “HMAC-SHA-2 authentication protocols in user-based security model (USM) for SNMPv3”, Internet Engineering Task Force, RFC7860, Apr. 2016, RFC7860. DOI: 10.17487/RFC7860. [Online]. Available: https://www.rfc-editor.org/info/rfc7860 (visited on 25.05.2020).

[51] Cisco. (21st Jan. 2018). SNMP configuration guide - AES and 3-DES encryption support for SNMP version 3 [cisco ASR 1000 series aggregation services routers], Cisco. Library Catalog: www.cisco.com, [Online]. Available: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/snmp/configuration/xe-16/snmp-xe-16-book/nm-snmp-encrypt-snmp-support.html (visited on 22.07.2020).

[52] G. Singh and S. Supriya, “A study of encryption algorithms (RSA, DES, 3des and AES) for information security”, *International Journal of Computer Applications*, vol. 67, no. 19, pp. 33–38, 18th Apr. 2013, ISSN: 09758887. DOI: 10.5120/11507-7224. [Online]. Available: http://research.ijcaonline.org/volume67/number19/pxc3887224.pdf (visited on 25.05.2020).

[53] S. Kelly, “Security implications of using the data encryption standard (DES)”, Internet Engineering Task Force, RFC4772, Dec. 2006, RFC4772. DOI: 10.17487/rfc4772. [Online]. Available: https://www.rfc-editor.org/info/rfc4772 (visited on 25.05.2020).

[54] B. Wijnen, R. Presuhn, and K. McCloghrie, “View-based access control model (VACM) for the simple network management protocol (SNMP)”, Internet Engineering Task Force, RFC3415, Dec. 2002, RFC3415. DOI: 10.17487/rfc3415. [Online]. Available: https://www.rfc-editor.org/info/rfc3415 (visited on 03.06.2020).

[55] C. Lonvick, “The BSD syslog protocol”, Internet Engineering Task Force, RFC3164, Aug. 2001, RFC3164. DOI: 10.17487/rfc3164. [Online]. Available: https://www.rfc-editor.org/info/rfc3164 (visited on 17.03.2020).

[56] R. Gerhards, “The syslog protocol”, Internet Engineering Task Force, RFC5424, Mar. 2009, RFC5424. DOI: 10.17487/rfc5424. [Online]. Available: https://www.rfc-editor.org/info/rfc5424 (visited on 17.03.2020).

[57] D. New and M. Rose, “Reliable delivery for syslog”, Internet Engineering Task Force, RFC3195, Nov. 2001, RFC3195. DOI: 10.17487/rfc3195. [Online]. Available: https://www.rfc-editor.org/info/rfc3195 (visited on 17.03.2020).

[58] A. Okmianski, “Transmission of syslog messages over UDP”, Internet Engineering Task Force, RFC5426, Mar. 2009, RFC5426. DOI: 10.17487/rfc5426. [Online]. Available: https://www.rfc-editor.org/info/rfc5426 (visited on 22.03.2020).

[59] F. Miao, Y. Ma, and J. Salowey, “Transport layer security (TLS) transport mapping for syslog”, Internet Engineering Task Force, RFC5425, Mar. 2009, RFC5425. DOI: 10.17487/rfc5425. [Online]. Available: https://www.rfc-editor.org/info/rfc5425 (visited on 17. 04. 2020).

[60] J. Salowey, T. Petch, R. Gerhards, and H. Feng, “Datagram transport layer security (DTLS) transport mapping for syslog”, Internet Engineering Task Force, RFC6012, Oct. 2010, RFC6012. DOI: 10.17487/rfc6012. [Online]. Available: https://www.rfc-editor.org/info/rfc6012 (visited on 27. 05. 2020).

[61] G. Klyne and C. Newman, “Date and time on the internet: Timestamps”, Internet Engineering Task Force, RFC3339, Jul. 2002, RFC3339. DOI: 10.17487/rfc3339. [Online]. Available: https://www.rfc-editor.org/info/rfc3339 (visited on 22. 03. 2020).

[62] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, “Network configuration protocol (NETCONF)”, Internet Engineering Task Force, RFC6241, Jun. 2011, RFC6241. DOI: 10.17487/rfc6241. [Online]. Available: https://www.rfc-editor.org/info/rfc6241 (visited on 17. 04. 2020).

[63] M. Wasserman, “Using the NETCONF protocol over secure shell (SSH)”, Internet Engineering Task Force, RFC6242, Jun. 2011, RFC6242. DOI: 10.17487/rfc6242. [Online]. Available: https://www.rfc-editor.org/info/rfc6242 (visited on 13. 05. 2020).

[64] T. Goddard, “Using NETCONF over the simple object access protocol (SOAP)”, Internet Engineering Task Force, RFC4743, Dec. 2006, RFC4743. DOI: 10.17487/rfc4743. [Online]. Available: https://www.rfc-editor.org/info/rfc4743 (visited on 13. 05. 2020).

[65] E. Lear and K. Crozier, “Using the NETCONF protocol over the blocks extensible exchange protocol (BEEP)”, Internet Engineering Task Force, RFC4744, Dec. 2006, RFC4744. DOI: 10.17487/rfc4744. [Online]. Available: https://www.rfc-editor.org/info/rfc4744 (visited on 13. 05. 2020).

[66] M. Badra, A. Luchuk, and J. Schoenwaelder, “Using the NETCONF protocol over transport layer security (TLS) with mutual x.509 authentication”, Internet Engineering Task Force, RFC7589, Jun. 2015, RFC7589. DOI: 10.17487/RFC7589. [Online]. Available: https://www.rfc-editor.org/info/rfc7589 (visited on 13. 05. 2020).

[67] C. Rigney, S. Willens, A. Rubens, and W. Simpson, “Remote authentication dial in user service (RADIUS)”, Internet Engineering Task Force, RFC2865, Jun. 2000, RFC2865. DOI: 10.17487/rfc2865. [Online]. Available: https://www.rfc-editor.org/info/rfc2865 (visited on 08. 05. 2020).

[68] M. Bjorklund, “The YANG 1.1 data modeling language”, Internet Engineering Task Force, RFC7950, Aug. 2016, RFC7950. DOI: 10.17487/RFC7950. [Online]. Available: https://www.rfc-editor.org/info/rfc7950 (visited on 18. 05. 2020).

[69] B. Lengyel and M. Bjorklund, “Partial lock remote procedure call (RPC) for NETCONF”, Internet Engineering Task Force, RFC5717, Dec. 2009, RFC5717. DOI: 10.17487/rfc5717. [Online]. Available: https://www.rfc-editor.org/info/rfc5717 (visited on 14. 05. 2020).

[70] E. Voit, A. Clemm, A. Gonzalez Prieto, E. Nilsen-Nygaard, and A. Tripathy, “Subscription to YANG notifications”, Internet Engineering Task Force, RFC8639, Sep. 2019, RFC8639. DOI: 10.17487/RFC8639. [Online]. Available: https://www.rfc-editor.org/info/rfc8639 (visited on 14. 05. 2020).

[71] A. Clemm and E. Voit, “Subscription to YANG notifications for datastore updates”, Internet Engineering Task Force, RFC8641, Sep. 2019, RFC8641. DOI: 10.17487/RFC8641. [Online]. Available: https://www.rfc-editor.org/info/rfc8641 (visited on 14. 05. 2020).

[72] M. Scott and M. Bjorklund, “YANG module for NETCONF monitoring”, Internet Engineering Task Force, RFC6022, Oct. 2010, RFC6022. DOI: 10.17487/rfc6022. [Online]. Available: https://www.rfc-editor.org/info/rfc6022 (visited on 14. 05. 2020).

[73] S. Chisholm and H. Trevino, “NETCONF event notifications”, Internet Engineering Task Force, RFC5277, Jul. 2008, RFC5277. DOI: 10.17487/rfc5277. [Online]. Available: https://www.rfc-editor.org/info/rfc5277 (visited on 14. 05. 2020).

[74] M. Bjorklund, J. Schoenwaelder, P. Shafer, K. Watsen, and R. Wilton, “Network management datastore architecture (NMDA)”, Internet Engineering Task Force, RFC8342, Mar. 2018, RFC8342. DOI: 10.17487/RFC8342. [Online]. Available: https://www.rfc-editor.org/info/rfc8342 (visited on 15. 07. 2020).

[75] A. Bierman, M. Bjorklund, and K. Watsen, “RESTCONF protocol”, Internet Engineering Task Force, RFC8040, Jan. 2017. DOI: 10.17487/RFC8040. [Online]. Available: https://www.rfc-editor.org/info/rfc8040 (visited on 27. 04. 2020).

[76] M. Bjorklund, J. Schoenwaelder, P. Shafer, K. Watsen, and R. Wilton, “RESTCONF extensions to support the network management datastore architecture”, Internet Engineering Task Force, RFC8527, Mar. 2019, RFC8527. DOI: 10.17487/RFC8527. [Online]. Available: https://www.rfc-editor.org/info/rfc8527 (visited on 16. 07. 2020).

[77] I. Hickson, “Server-sent events”, W3C, W3C Recommendation, Feb. 2015. [Online]. Available: https://www.w3.org/TR/eventsource/ (visited on 18. 05. 2020).

[78] J. Beasley, *A Practical Guide to Advanced Networking*, 3rd edition. Pearson IT Certification, 2015, 528 pp., ISBN: 978-0-7897-5749-4. [Online]. Available: https://www.safaribooksonline.com/library/view/-/9780132882996/.

[79] R. Hofstede, P. Celeda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras, "Flow monitoring explained: From packet capture to data analysis with NetFlow and IPFIX", *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2037–2064, 2014, ISSN: 1553-877X. DOI: 10.1109/COMST.2014.2321898. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6814316 (visited on 23. 04. 2020).

[80] B. Claise, "Cisco systems NetFlow services export version 9", Internet Engineering Task Force, RFC3954, Oct. 2004, RFC3954. DOI: 10.17487/rfc3954. [Online]. Available: https://www.rfc-editor.org/info/rfc3954 (visited on 17. 04. 2020).

[81] B. Claise, B. Trammell, and P. Aitken, "Specification of the IP flow information export (IPFIX) protocol for the exchange of flow information", Internet Engineering Task Force, RFC7011, Sep. 2013, RFC7011. DOI: 10.17487/rfc7011. [Online]. Available: https://www.rfc-editor.org/info/rfc7011 (visited on 17. 04. 2020).

[82] B. Claise and B. Trammell, "Information model for IP flow information export (IPFIX)", Internet Engineering Task Force, RFC7012, Sep. 2013, RFC7012. DOI: 10.17487/rfc7012. [Online]. Available: https://www.rfc-editor.org/info/rfc7012 (visited on 17. 04. 2020).

[83] Internet Assigned Numbers Authority. (25th Jul. 2019). IP flow information export (IPFIX) entities, iana.org, [Online]. Available: https://www.iana.org/assignments/ipfix/ipfix.xhtml (visited on 20. 05. 2020).

[84] M. Marculescu. (26th Feb. 2015). Introducing gRPC, a new open source HTTP/2 RPC framework, Google Developers Blog. Library Catalog: developers.googleblog.com, [Online]. Available: https://developers.googleblog.com/2015/02/introducing-grpc-new-open-source-http2.html (visited on 16. 07. 2020).

[85] gRPC. (2020). gRPC, gRPC. Library Catalog: grpc.io, [Online]. Available: https://grpc.io/ (visited on 29. 06. 2020).

[86] Google. (2020). Protocol buffers, Google Developers. Library Catalog: developers.google.com, [Online]. Available: https://developers.google.com/protocol-buffers (visited on 21. 07. 2020).

[87] M. Belshe, R. Peon, and M. Thomson, "Hypertext transfer protocol version 2 (HTTP/2)", Internet Engineering Task Force, RFC7540, May 2015, RFC7540. DOI: 10.17487/RFC7540. [Online]. Available: https://www.rfc-editor.org/info/rfc7540 (visited on 16. 07. 2020).

[88] Google. (17th Jul. 2020). Components/cronet - chromium/src - git at google, [Online]. Available: https://chromium.googlesource.com/chromium/src/+/master/components/cronet/ (visited on 19. 07. 2020).

[89] P. Borman, C. Lebsack, C. Morrow, R. Shakir, and A. Shaiks, *gNMI specification*, version 0.6.0, original-date: 2015-09-10, 30th Jan. 2018. [Online]. Available: https://github.com/openconfig/reference/blob/master/rpc/gnmi/gnmi-specification.md (visited on 29. 06. 2020).

[90] P. Phaal and M. Lavine. (Jul. 2004). sFlow version 5, sflow.org, [Online]. Available: https://sflow.org/sflow_version_5.txt (visited on 19. 05. 2020).

[91] O. Kauppinen. (8th Apr. 2020). SNMPv3 trap support to snmp_trap input plugin by kauppine · pull request #7294 · influxdata/telegraf, GitHub. Library Catalog: github.com, [Online]. Available: https://github.com/influxdata/telegraf/pull/7294 (visited on 24. 04. 2020).

[92] Metrics 2.0. (5th Sep. 2016). Metrics 2.0: Implementations, [Online]. Available: http://metrics20.org/implementations/ (visited on 27. 07. 2020).

[93] Graphite. (2020). Graphite, [Online]. Available: https://graphiteapp.org/ (visited on 27. 07. 2020).

[94] InfluxData. (2020). InfluxData documentation, InfluxData, [Online]. Available: https://docs.influxdata.com/ (visited on 22. 05. 2020).

[95] L. Lhotka, "JSON encoding of data modeled with YANG", Internet Engineering Task Force, RFC7951, Aug. 2016, RFC7951. DOI: 10.17487/RFC7951. [Online]. Available: https://www.rfc-editor.org/info/rfc7951 (visited on 18. 05. 2020).

[96] ServiceNow. (2020). ServiceNow – the smarter way to workflow™. Library Catalog: www.servicenow.com, [Online]. Available: https://www.servicenow.com (visited on 27. 07. 2020).

[97] Splunk. (9th Apr. 2020). Splunk® common information model add-on - splunk documentation, [Online]. Available: https://docs.splunk.com/Documentation/CIM/4.15.0 (visited on 20. 05. 2020).

[98] collectd. (2020). Start page – collectd – the system statistics collection daemon, [Online]. Available: https://collectd.org/ (visited on 27. 07. 2020).

[99] OpenTSDB. (16th Dec. 2018). OpenTSDB - a distributed, scalable monitoring system, [Online]. Available: http://opentsdb.net/ (visited on 27. 07. 2020).

[100] Prometheus. (2020). Prometheus - monitoring system & time series database, [Online]. Available: https://prometheus.io/ (visited on 27. 07. 2020).

[101] D. Merkel, "Docker: Lightweight linux containers for consistent development and deployment", *Linux journal*, vol. 2014, no. 239, p. 5, 2014, ISSN: 1075-3583. DOI: 10.5555/2600239.2600241. [Online]. Available: https://www.seltzer.com/margo/teaching/CS508.19/papers/merkel14.pdf (visited on 15. 06. 2020).

[102] Apache. (2020). Apache kafka, Apache Kafka. Library Catalog: kafka.apache.org, [Online]. Available: https://kafka.apache.org/intro (visited on 07. 07. 2020).

[103] Verizon, *VerizonDigital/vflow*, 24th Jun. 2020. [Online]. Available: https://github.com/VerizonDigital/vflow (visited on 26. 06. 2020).

[104] B.-Y. Choi and S. Bhattacharyya, "On the accuracy and overhead of cisco sampled netflow", in *Proceedings of ACM SIGMETRICS Workshop on Large Scale Network Inference (LSNI)*, 2005, pp. 1–6. [Online]. Available: https://pdfs.semanticscholar.org/2c5d/fbc02659f390dcc6ce7c0579e78a03cfad1a.pdf (visited on 15. 06. 2020).

[105] B. Hedstrom, A. Watwe, and S. Sakthidharan, "Protocol efficiencies of NETCONF versus SNMP for configuration management functions", Master's thesis, University of Colorado, 2nd May 2011, 13 pp. [Online]. Available: https://pdfs.semanticscholar.org/5664/44aa2023ac8cf9910cc33ead8582ace4c9c4.pdf (visited on 15. 06. 2020).

[106] J. Yu and I. Al Ajarmeh, "An empirical study of the NETCONF protocol", in *2010 Sixth International Conference on Networking and Services*, Cancun, Mexico: IEEE, Mar. 2010, pp. 253–258, ISBN: 978-1-4244-5927-8. DOI: 10.1109/ICNS.2010.41. [Online]. Available: http://ieeexplore.ieee.org/document/5460639/ (visited on 11. 05. 2020).

[107] V. Marinov and J. Schönwälder, "Performance analysis of SNMP over SSH", in *Large Scale Management of Distributed Systems*, R. State, S. van der Meer, D. O'Sullivan, and T. Pfeifer, Eds., vol. 4269, Series Title: Lecture Notes in Computer Science, Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 25–36, ISBN: 978-3-540-47659-7. DOI: 10.1007/11907466_3. [Online]. Available: http://link.springer.com/10.1007/11907466_3 (visited on 14. 07. 2020).

[108] X. Du, M. Shayman, and M. Rozenblit, "Implementation and performance analysis of SNMP on a TLS/TCP base", in *2001 IEEE/IFIP International Symposium on Integrated Network Management Proceedings. Integrated Network Management VII. Integrated Management Strategies for the New Millennium (Cat. No.01EX470)*, Seattle, WA, USA: IEEE, 2001, pp. 453–466, ISBN: 978-0-7803-6719-7. DOI: 10.1109/INM.2001.918059. [Online]. Available: http://ieeexplore.ieee.org/document/918059/ (visited on 14. 07. 2020).

[109] P. Gonçalves, J. L. Oliveira, and R. Aguiar, "A study of encoding overhead in network management protocols", *International Journal of Network Management*, vol. 22, no. 6, pp. 435–450, 2012, ISSN: 1099-1190. DOI: 10.1002/nem.1801. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/nem.1801 (visited on 04. 07. 2020).

[110] S. Chisholm and R. Gerhards, "Alarms in syslog", Internet Engineering Task Force, RFC5674, Oct. 2009, RFC5674. DOI: 10.17487/rfc5674. [Online]. Available: https://www.rfc-editor.org/info/rfc5674 (visited on 23. 06. 2020).

[111] J. Kelsey, J. Callas, and A. Clemm, “Signed syslog messages”, Internet Engineering Task Force, RFC5848, May 2010, RFC5848. DOI: 10.17487/rfc5848. [Online]. Available: https://www.rfc-editor.org/info/rfc5848 (visited on 23. 06. 2020).

[112] A. Charny, F. Huang, G. Karagiannis, M. Menth, and T. Taylor, “Pre-congestion notification (PCN) boundary-node behavior for the controlled load (CL) mode of operation”, Internet Engineering Task Force, RFC6661, Jul. 2012, RFC6661. DOI: 10.17487/rfc6661. [Online]. Available: https://www.rfc-editor.org/info/rfc6661 (visited on 23. 06. 2020).