Chapter 1

Internet Management Protocols

Jürgen Schönwälder
Jacobs University Bremen, Germany

Contents

1.1 Introduction .................................................. 2
1.2 Management Protocol Requirements .................. 2
  1.2.1 Configuration Requirements ....................... 2
  1.2.2 Monitoring Requirements ........................... 5
  1.2.3 Event Notification Requirements .................. 5
  1.2.4 Testing and Troubleshooting Requirements ........ 7
  1.2.5 Measurement Data Collection Requirements ...... 7
  1.2.6 Security Requirements ............................... 8
  1.2.7 Non-Functional Requirements ..................... 9
1.3 Architectural Concepts ................................. 10
  1.3.1 Protocol Engines, Functions, and Entities .......... 10
  1.3.2 Subsystems and Models ............................ 12
  1.3.3 Naming and Addressing ............................ 12
1.4 Internet Management Protocols ...................... 13
  1.4.1 Simple Network Management Protocol (SNMP) ...... 13
  1.4.2 Network Configuration Protocol (NETCONF) ...... 20
  1.4.3 System Logging Protocol (SYSLOG) ............... 24
  1.4.4 Flow Information Export Protocol (IPFIX) ....... 29
1.5 Conclusions ............................................... 34
1.1 Introduction

Operating a large communication network requires tools to assist in the configuration, monitoring, and troubleshooting of network elements such as switches, routers, or firewalls. In addition, it is necessary to collect event reports to identify and track failures or to provide a log of network activities. Finally, it is necessary to collect measurement data for network planning and billing purposes.

With a handful of devices, many of the activities mentioned above can be supported in an ad-hoc approach by using whatever functionality and interface is provided by a given device. However, when the number of network elements and services increases, it becomes essential to implement more advanced management approaches and deploy specific management tools. Since communication networks often consist of a large number of devices from different vendors, the need for standardized management interfaces becomes apparent.

One particularly important management interface for standardization and interoperability is the interface between network elements and their management systems. Network elements are typically closed systems and it is not possible to install management software on the devices themselves. This makes network elements such as routers and switches fundamentally different from general purpose computing systems. Interfaces between management systems have seen less need for standardization since management systems usually run on full featured host systems and it is therefore easier to use general middleware technologies as a means to provide connectivity between management systems.

This chapter surveys Internet management protocols that are mainly used between network elements and their management systems. Since it is impossible to discuss all protocols in detail, the discussion in this chapter will focus on the requirements that have driven the design of the protocols and the architectural aspects through which the protocols address the requirements. By focusing the discussion on requirements and architectural concepts, it becomes possible to document basic principles that are not restricted to the current set of Internet management protocols, but which will likely apply to future management protocols as well.

1.2 Management Protocol Requirements

Before discussing Internet management protocols, it is useful to look at the requirements these protocols should address. The approach taken to identify the requirements is to discuss the main functions management protocols have to support.

1.2.1 Configuration Requirements

Network configuration management has the goal to define and control the behavior of network elements such that a desired network behavior is realized in a
reliable and cost-effective manner. To understand the complexity of this task, it is important to note that the behavior of a network element is not only defined by its configuration, but also by the interaction with other elements in the network. It is therefore necessary to distinguish between configuration state and operational state. The configuration state is installed by management operations while the operational state is typically installed by control protocols or self-configuration mechanisms.

As an example, consider how the packet forwarding behavior of an IP router is determined. A simple router on a stub network likely has static configuration state in the form of a forwarding table which determines how IP packets are forwarded. This forwarding table is usually established once and subsequently changed manually when the network topology changes. A more complex router on a more dynamic network usually takes part in a routing protocol exchange which is used to compute suitable forwarding tables, taking into account the current state of the network. The forwarding behavior of a router participating in a routing protocol therefore is determined by operational state which was obtained at runtime and influenced by neighboring network elements. Note that the way the router participates in the routing protocol exchange is likely determined itself by configuration state, which sets the parameters and constraints under which shortest paths are computed.

**Requirement 1.1** A configuration management protocol must be able to distinguish between configuration state and operational state.

The example above introduces another important aspect in network configuration management: since devices typically exchange control information to establish operational state, configuration state changes on a single device can affect the operational state in many other devices. As a consequence, configuration state changes can have side effects that may affect a whole network.

With this in mind, it becomes clear that it is not sufficient to look solely at the configuration state of a device in order to identify why a certain network element does not behave as desired. To understand why an element misbehaves, it is necessary to look at the combined configuration state and operational state and it is desirable that devices report configuration information in a way which allows to distinguish between configuration state and operational state. In addition, it is desirable that the device also records sufficient information which allows an operator to figure out why and when operational state was established. For completeness, it should be noted that network elements maintain statistics in addition to configuration and operations state.

Configuration changes in a network may originate from different systems. In addition, it might be necessary to change several systems for a logical change to be complete. This leads to the following two requirements:

**Requirement 1.2** A configuration management protocol must provide primitives to prevent errors due to concurrent configuration changes.
Requirement 1.3 A configuration management protocol must provide primitives to apply configuration changes to a set of network elements in a robust and transaction-oriented way.

Requirement 1.4 It is important to distinguish between the distribution of configurations and the activation of a certain configuration. Devices should be able to hold multiple configurations.

Devices usually maintain different configurations. It is common to distinguish between the currently running configuration and the configuration that should be loaded at startup time. Some devices also support an open ended collection of different configurations. This leads to the following two requirements:

Requirement 1.5 A configuration management protocol must be able to distinguish between several configurations.

Requirement 1.6 A configurations management protocol must be clear about the persistence of configuration changes.

Since different management systems and operators may have access to the configuration of a device, it is important to log information that can be used to trace when and why a certain configuration change has been made.

Requirement 1.7 A configuration management protocol must be able to report configuration change events to help tracing back configuration changes.

Not all devices can be expected to have the resources necessary to track all configuration changes throughout their lifetime. It is therefore common practice to keep copies of device configurations on host computers and it is thus desirable to be able to use standard version control systems to track changes.

Requirement 1.8 A full configuration dump and a full configuration restore are primitive operations frequently used by operators and must be supported appropriately.

Requirement 1.9 A configuration management protocol must represent configuration state and operational state in a form which allows to use existing standard comparison and versioning tools.

Network operators are generally interested in providing smooth services. The number of service interruptions must be minimized and stability should be maintained as much as possible. A smart device should therefore be able to load a new configuration without requiring a full device reset or the loss of all operational state.

Requirement 1.10 Configurations must be described such that devices can determine a set of operations to bring the devices from a given configuration state to the desired configuration state, minimizing the impact caused by the configuration change itself on networks and systems.
1.2. MANAGEMENT PROTOCOL REQUIREMENTS

1.2.2 Monitoring Requirements

After configuring the devices making up a network, it is essential to monitor that they are functioning correctly and to collect some basic usage statistics such as link utilizations. The setup of monitoring tools can be greatly simplified if devices reveal their capabilities.

**Requirement 1.11** Monitoring protocols should support the discovery of the capabilities of a device.

Networks may contain a large number of manageable devices. Over time, monitoring requirements also tend to increase in terms of the number of data items to be monitored per device. As a consequence, monitoring protocols should be highly scalable.

**Requirement 1.12** Monitoring protocols must scale to a large number of devices as well as a large number of data items to be monitored.

Devices often contain large numbers of similar counters and in many cases not all of them are relevant for monitoring purposes. It is therefore crucial to be able to select which subsets of management data need to be retrieved for monitoring purposes, sometimes up to the level of an individual counter.

**Requirement 1.13** It must be possible to perform monitoring operations on selected subsets of management data.

**Requirement 1.14** Monitoring protocols must support a naming scheme which is capable to identify instances up to the granularity of a specific counter.

Finally, it should be noted that devices are built for some primary function (e.g., a router is built primarily to forward IP packets) and monitoring functions should thus have little negative impact on the primary functions of a device.

**Requirement 1.15** Monitoring protocols should have low impact on the primary functions of a device. In particular, it must be possible that multiple applications monitor different aspects of a single device without undue performance penalties for the device.

1.2.3 Event Notification Requirements

Operationally relevant status changes and in particular failures typically happen asynchronously. Devices therefore need a mechanism to report such events in a timely manner. As will be seen later, there are multiple protocols to transport event notifications. Regardless of the protocol, it is important to be able to identify some key parameters about an event.

**Requirement 1.16** Event notifications must have sufficient information to identify the event source, the time the event occurred, the part of the system that originated the event notification, and a broad severity classification.
Since event notifications may trigger complex procedures or work-flows, it is necessary to ensure the integrity and authenticity of an event notification. In some cases, it is even required to prove the event integrity of events stored in a permanent event log.

**Requirement 1.17** It is desirable to be able to verify the integrity of event notifications and the authenticity of the event source.

Events may originate from many different devices and processes in a network and it is likely that formats and protocols are not always consistent. Furthermore, event notifications may have to pass middleboxes such as firewalls and network address translators and there might be multiple levels of filtering and different data collectors for different purposes.

**Requirement 1.18** Event notifications may pass through a chain of relays and collectors which should be (a) transparent for the event notification originator and (b) not change the content of event notifications in any way.

One particular design issue that needs to be addressed when designing an event notification system are effective mechanisms to deal with so called event storms. Event storms happen for example after a power outage or a cut of a fiber when multiple devices and processes detect an error and issue event notifications. Event storms look very similar to denial of service attacks, except that the event messages actually originate from a valid source.

**Requirement 1.19** Notification senders should provide effective throttling mechanisms in order to deal with notification storms.

A reliable event notification transport is often desirable. However, it should be noted that choosing a reliable transport layer protocol does not by itself provide a reliable notification delivery system since the transport might simply fail while the network is under stress or attack or the notification receiver might not understand the notification. For a highly reliable event notification system, it is therefore essential to provide a local notification logging facility and a communication protocol providing confirmed delivery of event notifications.

**Requirement 1.20** Highly reliable event notification systems must provide confirmed event notification protocols and logging facilities to store event notifications at the event source.

Finally, it has proven to be valuable that event notifications can be read by both machines and humans.

**Requirement 1.21** Event notifications should include machine readable structured data as well as human readable event descriptions.
1.2.4 Testing and Troubleshooting Requirements

Troubleshooting and testing is a common activity, especially when new services and networks are deployed. Management protocols should support these activities.

Requirement 1.22 A management protocol must allow the invocation of test functions for trouble shooting purposes.

The ability to invoke test functions from different locations in a given network topology can lead to specific insights about the root cause of observed problems very quickly. In many situations, it is desirable to execute test functions periodically so that alarms can be triggered if tests fail or an expected heartbeat message is not received.

Requirement 1.23 Scheduling functions should be provided to execute testing functions periodically or following a calendar schedule.

1.2.5 Measurement Data Collection Requirements

It is import to collect accurate measurement data sets describing how networks are being used. Such information is, among other things, useful for usage-based accounting, network planning and traffic engineering, quality of service monitoring, trouble shooting, or attack and intrusion detection.

To collect measurement data, it is necessary to define observations points where meters are attached to a network in order to collect measurement data sets. These data sets are then exported by an exporting process, which transfers the data sets to a collecting process.

There are several specific measurement data collection requirements. The first one deals with scalability:

Requirement 1.24 Measurement data collection protocols must scale to hundreds of exporting processes. Furthermore, data exporting processes must be able to export to multiple data collecting processes.

Scalability and efficiency is a key requirement. Since network bandwidth tends to grow faster than processor speeds, it can be expected that only statistical sampling techniques can be deployed on very high-speed optical backbone networks in the future. Related to scalability is the requirement for measurement data transfer protocol to be congestion aware.

Requirement 1.25 Measurement data transfer protocols must be reliable and congestion aware.

Congestion during the data transfer may lead to significant data loss and may affect other network services. Data loss can, however, occur at many places, namely at the metering process, the export process, during the data transfer, or at the collecting process. To provide a complete picture, it is therefore necessary properly track all data losses.
Requirement 1.26 *It must be possible to obtain an accurate indication about lost measurement data records.*

Since metering processes often attach timestamps to data sets, it is necessary to select timestamps with a sufficiently high granularity and to synchronize the clocks on the meters.

Requirement 1.27 *Metering processes must be able to assign timestamps with a suitable resolution. Furthermore, metering processes must be able to synchronize their clocks with the necessary precision.*

Early protocols for exchanging measurement data sets assumed a specific hardwired data model. This approach has proven problematic several times, leading to the following requirement:

Requirement 1.28 *The data format used by the data transfer model must be extensible.*

Finally, it should be noted that measurement data sets and protocols that transfer them need proper protection.

Requirement 1.29 *Measurement data transfer protocols must provide data integrity, authenticity of the data transferred from an exporting process to a collecting process, and confidentiality of the measurement data.*

For additional details on measurement data collection requirements, see RFC 3917 [43].

1.2.6 Security Requirements

Network management information in general is sensitive. This seems obvious in the context of configuration, but may be less obvious in the context of monitoring. But as security attacks are getting more sophisticated, it is important to increase the protection of data that can identify ongoing attacks since attackers have a natural desire to hide their activities.

Requirement 1.30 *There is a need for secure data transport, authentication, and access control.*

Operationally, it seems that the usage of cryptographically strong security mechanisms do not cause too many problems if the required processing power is available. However, the required key management can be time consuming. Hence, it is important that key and credential management functions integrate well.

Requirement 1.31 *Any required key and credential management functions should integrate well with existing key and credential management infrastructures.*
Once you have authenticated principals, it is natural to apply access control to limit the accessible information to the subset needed to perform a specific function.

**Requirement 1.32** The granularity of access control must match operational needs. Typical requirements are a role-based access control model and the principle of least privilege, where a user can be given only the minimum access necessary to perform a required task.

Unfortunately, it is often difficult to define precisely which information needs to be accessible for a given management application to function correctly. Many applications resort to some probing and discovery algorithms and they use best guess heuristics in cases where some information is not directly accessible.

### 1.2.7 Non-Functional Requirements

Several non-functional requirements should be taken into account as well. The first one concerns human readability of protocol messages. The experience with SNMP (which uses binary encoded messages) tells us that the usage of specially encoded messages significantly increases the time programmers spend in writing new applications.

**Requirement 1.33** Human readable protocol messages significantly reduce integration efforts.

The availability of C/C++/Java libraries with well defined stable APIs does not seem to make it easy enough for developers and operators to create new applications quickly. For network management purposes, good integration into high-level scripting languages is therefore an essential requirement.

**Requirement 1.34** Management protocols should be easy to access from high-level scripting languages.

On managed devices, it has been observed that the development of management interfaces sometimes happens in a relatively uncoordinated way. This leads often to duplicate implementation and maintenance efforts which increase costs and and inconsistencies across management interfaces which decrease the operator satisfaction.

**Requirement 1.35** Implementation costs on network devices must be minimized and duplicated implementation efforts for different management protocols should be avoided.

Management systems usually contain a significant amount of code that is specific to a certain operator or network type. Furthermore, many network management tasks involve work-flows (e.g., the ordering of repair parts or a temporary allocation of bandwidth from other providers) that need some level of information technology support. Management applications and protocol interfaces should therefore be easy to integrate with standard software tools.
Requirement 1.36 Management protocol interfaces should integrate well with standard tools such as revision management tools, text analysis tools, report generation tools, trouble ticketing tools, databases, or in general work-flow support tools.

1.3 Architectural Concepts

Before discussing concrete protocols, it is useful to introduce some architectural concepts that are being used later in this chapter to describe to various Internet management protocol frameworks.

1.3.1 Protocol Engines, Functions, and Entities

The architecture of a management protocol can usually be divided in a protocol engine handling protocol messages and protocol functions that realize the protocol’s functionality. The protocol engine and the protocol functions together form a protocol entity, as shown in Figure 1.1.

![Protocol Engines, Functions, and Entities](image)

Figure 1.1: Protocol Engines, Functions, and Entities

Protocol Engine

The protocol engine is primarily concerned with the handling of the message formats used by a protocol. New protocols usually have a single message format. But during the design of a new protocol, it must already be anticipated that message formats need changes over time if a protocol is successfully deployed and new requirements must be addressed. Some of the older Internet management protocols have already been gone through this evolution while some of the newer protocols are too young to have experienced a need for different message formats.
From an architectural point of view, however, it is important to plan for message format changes that should be handled by the protocol engine.

An important service provided by the protocol engine is the identification of the principal invoking a protocol function. This usually requires cryptographic mechanisms and hence the protocol engine must deal with message security processing and provide mechanisms to ensure message integrity, data origin authentication, replay protection and privacy. There are two commonly used approaches to achieve this:

1. The first approach is to design specific security processing features into the protocol engine. This has the advantage that the protocol is self-contained and reduces its dependency on external specifications and infrastructure. The downside is that the specification becomes more complex and more difficult to review and that implementation and testing costs increase.

2. The second approach, which is becoming increasingly popular, is the usage of secure transport mechanism. The protocol engine then simply passes authenticated identities through the protocol engine. This has the advantage that prior work on the specification, testing and implementation of security protocols can be leveraged easily.

A protocol engine can also provide access control services to the protocol functions. While the actual call to check access control rules might originate from protocol functions realized outside of the engine, it is usually desirable to maintain the access control decision logic within the protocol engine to ensure that different protocol functions realize a consistent access control policy.

Finally, a protocol engine has to deal with the mapping of the protocol messages to underlying message transports. It seems that in the network management space, it has become increasingly popular to define different transports mappings for a given management protocol and to mark one of them as mandatory to implement to achieve a baseline of interoperability.

Protocol Functions

The protocol functions realize the functionality of a management protocol. A protocol engine usually determines the protocol operation that should be invoked after parsing an incoming message and then searches for a protocol function that has registered to handle the protocol operation. Such a registration can be conceptual and rather static in implementations or it can be something which is dynamic and runtime extensible.

Management protocols differ quite a bit in the number of protocol functions they support. In some management protocols, the number of protocol functions is rather fixed and there is strong consensus that adding new protocol functions should be avoided while other protocols are designed to be extensible at the protocol functions layer.
1.3.2 Subsystems and Models

The internals of a protocol engine can be decomposed into a set of subsystems. A subsystem provides a well defined functionality which is accessible through a defined subsystem interface. By documenting the subsystems and their interfaces, the data flow between the components of a protocol engine can be understood. Furthermore, defined subsystem interfaces reduce the number of surprises when a protocol design gets extended over time.

Within a subsystem, there might be one or multiple models that implement the subsystem’s interface. Subsystems designed to support multiple models make an architecture extensible. However, there are also subsystems that by design do not support multiple models since they for example organize the data flow between subsystems and models.

1.3.3 Naming and Addressing

Management protocols manipulate managed objects, which are abstractions of real resources. The term “managed object” has to be understood in a broad sense; it does not imply that a managed object is an object-oriented data structure. A managed object can be as simple as a simple typed variable or a part in a complex hierarchically structured document.

A particularly important architectural design decision is the naming, that is the identification of managed objects. There are several different approaches to name managed objects and the power, flexibility, or simplicity of the naming system has direct impact on the implementation costs and the runtime costs of a management protocol. As a consequence, the selection of a naming mechanism should be treated as an important architectural design decision.

The names of managed objects are usually implicitly scoped. Within the scope, a name provides a unique identifier for a managed object. One commonly used approach is to scope the names relative to the protocol engine that provides access to the managed object. This means that a globally scoped name within a management domain can be constructed out of the identity of the protocol engine and the name of the managed object. Hence, it is important to define how protocol engines are addressed. Some management protocols introduce their own protocol engine addressing scheme while other management protocols use transport endpoint addresses as a shortcut to identify the protocol engine. Architecturally, the first approach is clearly preferable since it decouples the protocol engine from the transport endpoints. Practically, it turns out that such a decoupling has its price since an additional mapping is needed. The selection of a suitable protocol engine addressing scheme therefore becomes an engineering trade-off between architectural purity and implementation and deployment costs.
1.4 Internet Management Protocols

This section describes several management protocols that were specifically designed for managing the Internet. The protocols discussed below are all specified and standardized by the Internet Engineering Task Force (IETF).

1.4.1 Simple Network Management Protocol (SNMP)

The core Internet architecture was developed and tested in the late 1970s by a small group of researchers [11]. During the early 1980s, several important features such as the domain name system were added to the architecture [12]. At the end of the 1980s, the Internet had grown to a world-wide network connecting all major universities and research institutions. Network management functions were at this time realized on an ad-hoc basis through explicit management.

In 1988, it became apparent that additional technology is needed in order to provide better tools to operate the growing Internet infrastructure [8]. The debate held in 1988 shows a fundamental principle in the IETF approach: The IETF seeks to make pragmatic decisions in order to address short-term technical problems. Rather than spending much time on architectural discussions, concrete proposals which are backed up by interoperable implementations have a good chance to be adopted by the IETF and finally in the market place.

The Simple Network Management Protocol (SNMP) [5], the Structure of Management Information (SMI) [47, 48, 46] and an initial Management Information Base (MIB-II) [37] were standardized in 1990 and 1991. One of the fundamental goals of the original SNMP framework was to minimize the number and complexity of management functions realized by the management agent itself [5]. The desire to minimize the complexity of management functions in the network elements reflects the constraints on processing power available in the early 1990s and the hardware/software architecture used in most networking devices at that time. Much has changed in the last 10 years in this aspect. Current network elements, such as routers and switches, implement all the primary and time critical network functions in hardware. Furthermore, the processing speed of the microprocessors embedded in networking devices has increased in magnitudes during the last ten years. But it is also important to understand that the network traffic handled by these devices did evolve exponentially. The question whether todays and tomorrows network elements have the computational power to realize much more complex management functions is not easy to answer.

The evolution of the original SNMP framework and the MIB modules during the last two decades did provide security, some efficiency improvements, and enhancements of the data definition language. Today, there are more than 100 MIB modules on the standardization track within the IETF and there is an even larger and growing number of enterprise-specific MIB modules defined unilaterally by various vendors, research groups, consortia, and the like resulting in an unknown and virtually uncountable number of defined objects [7, 52].
In December 2002, the specification of SNMP version 3 (SNMPv3) was published as an Internet Standard [25, 6, 30, 1, 58, 41, 40], retiring all earlier versions of the SNMP protocol. The second and current version of the SMI (SMIv2) was already published in April 1999 as an Internet Standard [35, 36, 34].

Deployment Scenarios

A typical SNMP deployment scenario is shown in Figure 1.2. It shows a single manager which is talking SNMP to three agents running on different SNMP-enabled devices.

For one device, a so called SNMP proxy is involved. There can be different reasons for deploying proxies. One reason can be firewalls and in particular network address translators [44]. Another reason can be version mismatches between management stations and devices.\(^1\)

Agents internally often have a master/subagent structure [17]. Subagents can be embedded into line-cards on routers or application processes to provide access to specific management information. The access to management is usually implemented by so called method functions in the instrumentation.

Many management systems use full fledged database management systems to store management information retrieved from network elements. Applications access the data stored in the database to realize management functions. In addition, they may interact directly with devices by using services provided by the SNMP manager interface.

\(^1\)Devices have picked up support for SNMPv3 much faster than some widely used management applications.
It should be noted that Figure 1.2 provides a rather traditional view. There are many situations where the notion of an agent and a manager is broken apart. In particular, it is not unusual to logically separate the forwarding and processing of notifications from the initiation and processing of commands to retrieve or modify data stored on SNMP-enabled devices.

Architecture

One of the biggest achievements of the SNMPv3 framework next to the technical improvements of the protocol itself is the architectural model defined in RFC 3411 [25]. It decomposes an SNMP protocol engine into several subsystems and it defines abstract service interfaces between these subsystems. A subsystem can have multiple concrete models where each model implements the subsystem interface defined by the abstract service interfaces. The architectural model thus provides a framework for future SNMP enhancements and extensions.

Figure 1.3 shows the decomposition of an SNMP protocol engine into subsystems. SNMP protocol messages enter an SNMP engine via the transport subsystem. Several different transport models can co-exist in the transport subsystem. The most commonly used transport model is the SNMP over UDP transport model. The transport subsystem passes messages to the dispatcher. The dispatcher, which is a singleton, coordinates the processing in an SNMP engine. It passes incoming messages to the appropriate message processing model in the message processing subsystem. The message processing models take care of the different headers used by the different versions of the SNMP protocol. The message processing model passes messages to a security model in the security subsystem which provides security services such as authentication and privacy of messages. After returning from the security and message processing

\[\text{Figure 1.3: Subsystem Architecture of SNMP}\]

\[\text{Figure 1.3 does not precisely follow the standard architecture since it includes a transport subsystem which was just recently added to the architectural model [26].}\]
subsystems, incoming messages are passed by the dispatcher to SNMP functions which implement the requested protocol operations. SNMP functions call the access control subsystem which can provide several different access control models for access control services.

The SNMP framework traditionally follows a variable-oriented approach. Management information is represented by collections of variables. Every variable has a simple primitive type. Variables can appear as simple scalars or they can be organized into so-called conceptual tables. Due to this very granular approach, every variable needs a unique name. The current SNMP architecture uses a four level naming system. The name of a variable consists of the following components:

1. The identification of the engine of the SNMP entity which hosts the variable. This is called the context engine identifier.
2. The identification of the context within an engine which hosts the variable. This is called the context name.
3. The identification of the variable type, also commonly called an object type. An object type is identified by means of a registered object identifier.
4. The identification of the variable instance, also called an instance identifier. An instance identifier is an object identifier suffix which gets appended to the object identifier of the object type.

Like for every naming system, there needs to be a mechanism to lookup names and to explore the name space. The SNMP framework addresses this requirement by restricting protocol operations to a single context. A context is identified by a context engine identifier and a context name. Within the selected context, multiple variables can be read, written, or submitted as part of an event notification.

To explore the name space, SNMP uses a two-level approach:

- Contexts are typically discovered through discovery procedures. SNMP engines can be discovered by probing their transport endpoints. Once a transport endpoint has been found, SNMP messages can be send to the SNMP engine listening on that transport endpoint to discover the engine identifier and existing contexts.

- The variables within a context are discovered by using an iterator which traverses the name space while retrieving variables.

Using an iterator, which retrieves data while traversing the name space, has significant advantages in highly dynamic situations. Management Protocols which separate name space traversal operations from data retrieval operations usually are less scalable due to the imposed overhead of managing the name space and synchronization issues in highly dynamic situations.
1.4. INTERNET MANAGEMENT PROTOCOLS

Data Modeling

Data modeling in the SNMP framework is centered around the notion of conceptual tables [35, 36, 34]. All columns of a conceptual table have a primitive data type. So-called textual conventions are used to associate special semantics with a primitive type; textual conventions are roughly comparable to typedefs in languages such as C.

Some columns of a conceptual table are marked as index columns. The values of the index columns uniquely identify a row and the index columns, or short the index, thus serve as the primary key for a conceptual table.

The basic concepts resemble relational databases. However, the SNMP framework puts several constraints on the indexing scheme. For example, conceptual tables can only have one index (key) that can be used for fast lookups and the index is subject to several additional constraints imposed by the protocol itself (index size constraints) or by access control mechanisms.

The data modeling framework supports scalars in addition to conceptual tables. For the discussion in this document, it is however sufficient to consider scalars just a degenerate form of a table which has only one row and a fixed index.

In order to assign tables a unique name, an object identifier naming system is used. Object identifier, a primitive ASN.1 data type, essentially denote a paths in the global object identifier registration tree. The tree has a virtual root and can be used to uniquely assign names to arbitrary objects [54, 19]. In the SNMP framework, tables as well as the table’s columns are registered in the tree. Note that the definition of index columns and the registration of tables happen at design time and cannot be changed once a specification has been published. As a consequence, the design of suitable indexing schemes is an important design step in any SNMP data modeling effort.

To fully understand the SNMP framework, it is crucial to properly distinguish between conceptual tables used at the data modeling layer and the registration tree used purely for naming purposes. As the next section explains, the actual protocol operations do neither understand tables nor registration trees and simply operates on an ordered list of variables.

Protocol Operations

The SNMP protocol provides a very small number of protocol operations, which is essentially unchanged since 1993 [41]. The protocol operations can be classified into write operations (set), read operations (get, getnext, getbulk), and notify operations (trap, inform). Figure 1.4 gives an overview which SNMP functions invoke the various operations and which ones are confirmed via a response.

The protocol operations do not operate on conceptual tables nor do they operate on the registration tree. Instead, they solely operate on an ordered collection of variables. The get, set, trap, and inform operations all identify variables by their full name (object type identifier plus instance identifier).
CHAPTER 1. INTERNET MANAGEMENT PROTOCOLS

getnext and getbulk operations obtain the next variable (getnext) or variables (getbulk) according to lexicographic ordering, which essentially maps the hierarchical name space into a flat ordered collection of variables.

The getnext and getbulk operations combine the traversal of the variable name space with actual data retrieval. This implies that variables can dynamically come and go without any need to register them in a name service.

Something that is often overlooked is the handling of errors and exceptions. SNMP protocol operations can fail with an error code which indicates the reason of the failure and which variable caused the failure. In the second version of the SNMP protocol operations, exceptions were introduced to report, on a per variable basis, exceptional conditions without failing the whole protocol operation. Not surprising, most error codes deal with the execution of write operations. However, exceptions are also important for read operations to indicate that specific instances do not exist or that an iterator such as getnext or getbulk has reached the end of the ordered collection of variables.

Security

A great deal of attention has been given to the security of SNMP as the standardization of a secure version of SNMP proved to be very difficult and time consuming. The SNMPv3 architecture has been designed with a message-based security model in mind where every individual message can be authenticated and also encrypted if so desired. The User-based Security Model (USM) [1] protects against the following threads:

- Modification of Information
- Masquerade
- Disclosure
- Message Stream Modification

It does not defend against denial of service attacks. The USM authenticates a user, identified by the user name, who is invoking a read or write operation or
who is receiving a notification. The user name is mapped into a model independent security name which is passed along inside the SNMP engine together with a security level and the identification of the security model.

The triple of security name, security level, and security model is used by the access control subsystem to decide whether an access to a variable is allowed or denied. The View-based Access Control Model (VACM) [58], the only access control model defined so far in the access control subsystem, provides means to define read/write/notify views on the collection of variables and to restrict access to these views.

One of the original design principles for SNMP was to keep SNMP as independent as possible of other network services so that SNMP can continue to function if the network is not running well. The original security mechanism follow this principle and thus come with an SNMP specific user, key, and access control rule management interface. Operators have reported that adding a new user and key management infrastructure has significant costs and that they prefer an approach where security better integrates with existing user, key, and access control management infrastructures. The IETF is at the time of this writing working on an additional security model, which utilizes secure session-based transports such as Secure Shell (SSH) [59] or Transport Layer Security (TLS) [18] protocol [24].

Discussion

The SNMP protocol has been very successful. It is widely deployed in network elements and management systems. However, the main usage of SNMP seems to be monitoring, discovery, and event notification.

SNMP has seen less usage for controlling and configuring devices. By looking at the configuration requirements detailed in Section 1.2.1, it becomes obvious why. SNMP fails to address most of the configuration related requirements and this is perhaps part of the reason why configuration requirements are so well documented in the RFC series [50, 51].

SNMP works well for polling large numbers of devices for a relatively small set of variables. Once the set of variables to be collected increase, SNMP becomes less efficient in terms of required bandwidth but also in terms of overall latency and processing overhead compared to other approaches. The reason is that protocol operations “single step” through the data retrieval, which makes efficient data retrieval and caching on a managed device complicated.

SNMP enjoys significant usage for event notification. However, the binary format of SNMP messages combined with the object identifier naming scheme makes the processing of such event records often difficult in practice and other event notification and logging protocols such as SYSLOG are therefore popular as well.

There were attempts to realize more generic and powerful management services on top of SNMP to delegate management functions [31, 32] or to invoke the remote execution of troubleshooting procedures [32]. These approaches have not seen much uptake since the invocation of remote management procedures
with read/write/notify primitives is rather unnatural and has relatively high
development costs.

1.4.2 Network Configuration Protocol (NETCONF)

In 2002, the Internet Architecture Board (IAB) organized a workshop in order to
guide the focus of the network management work done in the IETF. The work-
shop was attended by network operators and protocol developers and resulted
into some concrete recommendations [51]. One of the recommendations was
to focus resources on the standardization of configuration management mech-
nisms. Another recommendation was to use the Extensible Markup Language
(XML) [2] for data encoding purposes. Some of the discussions related to the
future directions are also summarized in [53].

In 2003, the a working group was formed in the IETF to produce a protocol
suitable for network configuration. The working group charter mandated that
the Extensible Markup Language (XML) [2] be used for data encoding purposes.
The protocol produced by this working group is called NETCONF [20] and the
subject of this section.

Deployment Scenarios

The driving force behind NETCONF is the need for a programmatic interface
to manipulate configuration state. The automation of command line interfaces
(CLIs) using programs and scripts has proven to be problematic, especially
when it comes to maintenance and versioning issues. Operators have reported
that it is time consuming and error prone to maintain programs or scripts that
interface with different versions of a command line interface.

Figure 1.5: NETCONF Deployment Scenario

Figure 1.5 shows a NETCONF deployment scenario. It is expected that
network-wide configuration or policy systems will use the NETCONF protocol to enforce configuration changes. In a policy framework, the manager involving a NETCONF client might act as a policy decision point while a device involving a NETCONF server might act as a policy enforcement point. Of course, such a setup requires that a policy manager can translate higher-level policies into device configurations; NETCONF only provides the protocol to communicate configuration data.

The right part of Figure 1.5 shows a CLI which talks NETCONF to a server in order to implement its functionality. It is important to realize that NETCONF must be powerful enough to drive CLIs. Real cost savings can only be achieved if there is a single method to effect configuration changes in a device which can be shared across programmatic and human operator interfaces. This implies that the scope of the NETCONF protocol is actually much broader than just device configuration.

Architecture

The NETCONF protocol does not have a very detailed architectural model. The protocol specification [20] describes a simple layered architecture which consists of a transport protocol layer, a remote procedure call (RPC) layer, an operation layer providing well defined RPC calls, and a content layer.

A more structured view of NETCONF, which resembles the SNMP architectural model, is shown in Figure 1.6. The transport subsystem provides several transport models. The working group so far has defined an SSH transport model [57], a BEEP transport model [28], and a SOAP transport model [23]. The SSH transport is mandatory to implement.

The dispatcher is the central component which organizes the data flow within the NETCONF engine. It hands messages received from the transport subsystem to the message processing subsystem which currently has only one message processing model. This message processing model handles a capabilities ex-
change, which happens after a transport has been established, and the framing or RPC messages. The operations being invoked are provided by NETCONF functions. The base NETCONF specification deals mainly with generic operations to retrieve and modify configuration state. An additional document defines operations to subscribe to notification channels and to receive notifications. It is expected that additional operations will be introduced in the future for more specific management purposes.

The base NETCONF specification follows a document-oriented approach. The configuration state of a device is considered to be a structured document that can be retrieved and manipulated. A filtering mechanism has been defined which allows to retrieve a subset of the document representing a configuration.

NETCONF supports multiple configuration datastores. A configuration datastore contains all information needed to get a device from its initial default state into the desired configuration state. The running datastore is always present and describes the currently active configuration. In addition, NETCONF supports the notion of a startup configuration datastore which is loaded at next re-initialization time and a candidate datastore which is a scratch buffer which can be manipulated and later committed to the running datastore.

Data Modeling

Since NETCONF uses XML to encode network management data and in particular configuration state, it seems obvious to use some XML schema notation to formally specify the format of these XML documents. However, there is no agreement yet which schema language to use. Furthermore, it seems necessary to incorporate additional information in data models that go beyond the capabilities of schema languages or requires to resort to hooks in schema languages where additional information can be specified which is ignored by generic tools.

During the development of the NETCONF protocol specifications, which are formally defined using XML schema [21], it was observed that the XML schema notation is difficult to read and verify for humans. Other schema notations such as Relax NG [15] and especially its compact notation [13] seem to be easier to read and write. Still, these schema languages tend to be relatively far away from an implementors view and in the light of integration with SNMP, it might also be possible to adapt a language such as SMIng [55] for NETCONF data modeling purposes.

The NETCONF working group was not chartered to address the data modeling aspects of NETCONF and as such there is no agreement yet how to approach this problem. It is, however, clear that agreement must be reached on how to write data models in order to achieve standardized configuration in a heterogeneous environment.

Protocol Operations

The NETCONF protocol has a richer set of protocol operations compared to SNMP. It is generally expected that new protocol operations will be added in the
future. This essentially means that NETCONF will likely in the future support a command-oriented approach in addition to the document-oriented approach for manipulating configuration state.

Figure 1.7: Protocol operations of NETCONF

Figure 1.7 shows the protocol operations that have been defined by the NETCONF working group. The first eight operations all operate on the configurations stored in configuration datastores. The lock and unlock operations do coarse grain locking and locks are intended to be short-lived.

The get operation is provided to retrieve a device’s configuration state and operational state; the get-config operation has been provided to only retrieve configuration state. The close-session operation initiates a graceful close of a session while the kill-session operation forces to terminate a session.

The most powerful and complex operation is the edit-config operation. While the edit-config operation allows to upload a complete new configuration, it can also be used to modify the current configuration of a datastore by creating, deleting, replacing or merging configuration elements. In essence, edit-config works by applying a patch to a datastore in order to generate a new configuration. Since a tree-based representation, is used to express configuration state, it is necessary to describe which branches in the tree should be created, deleted, replaced, or merged. NETCONF solves this by adding so called operation attributes to an XML document that is sent as part of the edit-config invocation to the NETCONF server.
The `get` and `get-config` operations both support a filter parameter, which can be used to select the parts of an XML document that should be retrieved. The protocol supports a subtree filter mechanisms which selects the branches of an XML tree matching a provided template. As an optional feature, implementations can also support XPATH [14] expressions.

The notification support in NETCONF is based on an event stream abstraction. Clients who want to receive notifications have to subscribe to an event stream. An event stream is an abstraction which allows to handle multiple event sources and includes support for transparently accessing event logs. On systems which maintain an event log, it is possible to subscribe to an event stream at some time in the history and the device will then playback all recorded event notifications at the beginning of the notification stream. A subscription to an event stream establishes a filter which is applied before event notifications are sent to the client. This allows to select only the interesting messages and improves scalability.

Security

NETCONF assumes that message security services are provided by the transports. As a consequence, Figure 1.6 does not contain a security subsystem. While it has been acknowledged by the working group that an access control subsystem and model is needed, standardization work in this area has not yet been started. Some research in this area can be found in [16].

Discussion

The design of the NETCONF protocol reflects experiences made with existing proprietary configuration protocols such as Juniper’s JunoScript. NETCONF addresses the requirements for configuration management protocol defined in section 1.2.1 and is therefore a good choice for this task. The pragmatic approach to layer the NETCONF protocol on top of a secure and reliable transport greatly simplifies the protocol.

The downside of NETCONF, as it is defined today, are the missing pieces, namely a lack of standards for data modeling and the lack of a standardized access control model. Since NETCONF implementations are well underway, it can be expected that these shortcomings will be addressed in the next few years and NETCONF might become a powerful tool for network configuration and trouble shooting.

1.4.3 System Logging Protocol (SYSLOG)

The SYSLOG protocol, originally developed in the 1980s as part of the Berkeley Software Distribution, is used to transmit event notification messages over the Internet. Due to its simplicity and usefulness, the SYSLOG protocol enjoys very wide deployment, even though there was never a specification of the protocol itself. This lead to a number of differences how SYSLOG messages
1.4. INTERNET MANAGEMENT PROTOCOLS

are handled by implementations. In 2001, an informational document was published which describes the de-facto SYSLOG protocol [33] and provides hints how implementations should deal with some of the deployed variations.

The SYSLOG protocol described in [33] has several shortcomings:

- The default transport (UDP) provides only unreliable message delivery.

- The default transport (UDP) does not protect messages. A large number of possible attacks are described in [33], among them message forgery, message modifications, and message replay.

- Some applications require mechanisms to authenticate the source of an event notification and to verify the integrity of the notification.

- SYSLOG messages contain no structured data which makes it difficult for machines to interpret them.

Since the publication of the document describing the de-facto SYSLOG protocol, the IETF has been working on a standards-track SYSLOG protocol addressing the shortcomings mentioned above [22].

Deployment Scenarios

The SYSLOG protocol can be deployed in a variety of configurations. Figure 1.8 shows some of the possible scenarios.

A SYSLOG-enabled device supports at least a SYSLOG sender. The sender can obtain event notifications from various sources. Typical sources are background processes such as routing daemons or processes implementing specific services, such as web server or database server. Another source of event notifications is often the operating system kernel, which usually provides special interfaces to pass kernel notifications to the SYSLOG process forwarding the messages.

Event notifications are collected by so called SYSLOG collectors, which typically store the notification in some database and might pass the information on to applications that are interested in processing the event notifications. Stored notifications are often used to investigate why failures happened in order to find root causes. However, stored notifications can also be useful to prove that a certain event did happen or a certain service was provided. It is therefore important that integrity, completeness, and authenticity of messages can be verified when they are fetched from the database.
The SYSLOG sender can forward event notifications via the SYSLOG protocol to notification collectors directly as shown in the right part of Figure 1.8, or it can use relays. Relays can act as simple forwarders in order to deal with firewalls or network address translators. They can, however, also be used to aggregate notifications from multiple senders or to filter notifications before passing them on towards collectors.

Architecture

The SYSLOG protocol is a so called “fire-and-forget” protocol. The originator of an event notification sends the notification to a receiver without getting an acknowledgment in any form whether the message was received, passed on to other systems, or successfully stored or processed.

SYSLOG receivers typically come in two flavors:

- A relay is a receiver that can receive messages and forward them to other receivers.
- A collector is a receiver that receives messages without relaying them.

Note that a sender is not aware whether a receiver acts as a relay or a collector. Furthermore, a single receiving entity may act as both a relay and a collector.

Figure 1.9 presents an architectural view of the SYSLOG protocol. The transport subsystem is responsible for the transmission of messages over the network. The SYSLOG protocol traditionally uses UDP as a transport. Recent work suggests to run SYSLOG over TLS [18] in order to protect messages on the wire.
The message processing subsystem currently includes two message processing models. The original message format is defined by the BSD SYSLOG protocol [33] while the standards-track message format is the new standards-track format developed by the IETF [22].

There are essentially three SYSLOG functions, namely an originator generating notifications, a relay forwarding notifications, and a collector receiving and storing notifications. As mentioned above, it is possible in the SYSLOG architecture to dispatch a received message to a relay and a collector registered on the same SYSLOG engine.

Due to the fire-and-forget nature of the SYSLOG protocol, there is no access control subsystem. An access control subsystem requires that a security subsystem establishes an authenticated identity in order to select appropriate access control rules. With a fire-and-forget protocol, it becomes difficult to establish such an authenticated identity. As a consequence, in SYSLOG there is also no need for a security subsystem.

The original SYSLOG carries only a small number of machine readable elements in notification messages, namely the identification of a facility originating the event, a severity level, a timestamp, an identification of the host originating the event, the name of the process originating the event, and optionally a process identification number. The rest of the message contains an arbitrary text message. The standards-track SYSLOG protocol essentially keeps the original format but adds more machine readable content in the form of structured data elements.

Structured data consists of zero, one, or multiple structured data elements and each structured data element contains a list of parameters in the form of name value pairs. Structured data elements are identified by a name, which is essentially a seven-bit ASCII character string. The name format supports standardized structured data element names as well as enterprise specific structured data element names. The names of the structured data element parameters
follow the same rules as structured data element names, but they are scoped by the structured data element name. The values use a simple ASCII encoding with an escape mechanism to handle special characters such as quotes.

**Data Model**

There is no data modeling framework yet for SYSLOG structured data elements. The specifications so far use plain text to define the format and semantics of structured data elements and their parameters.

Since SYSLOG notifications and SNMP notifications serve similar purposes, it can be expected that mappings will be defined to map SNMP varbind lists to SYSLOG structured data element parameters and also SYSLOG structured data elements into SNMP notification objects. Similar mappings already exist in many products between the more unstructured BSD SYSLOG protocol and SNMP.

**Protocol Operations**

The SYSLOG protocol operations are fairly simple since the protocol does not expect any bidirectional communication between SYSLOG entities.

![Protocol operations of SYSLOG](image)

Figure 1.10: Protocol operations of SYSLOG

Figure 1.10 shows that a notification is sent to a relay or a collector. A particular feature of SYSLOG is that a SYSLOG entity can act as both a relay and a collector at the same time. This means that the dispatcher of a SYSLOG engine must be able to dispatch a received message to multiple functions, something that does not exist, for example, in the SNMP architecture (a received SNMP notification is either proxied or passed to the notification receiver).

**Security**

The SYSLOG protocol traditionally provides no security. Not surprisingly, this is a major issue in environments where the management network itself can’t be properly secured. Furthermore, there are legislations in some countries that require operators to log certain events for later inspection and it is sometimes required to be able to prove that logged messages are unchanged and originated from the claimed source.

The SYSLOG specifications provide two mechanisms to deal with these security requirements. To provide hop-by-hop message protection, it is possible to run SYSLOG over TLS [38]. To achieve notification originator authentication
and integrity checking, it is possible to sign SYSLOG notifications [27]. This works by sending signature blocks as separate SYSLOG messages that sign the hashes of a sequence of recently sent SYSLOG notifications. Note that signature blocks can be logged together with SYSLOG messages and used later to verify the identity of the SYSLOG notification originator once the need arises.

Discussion

The SYSLOG protocol has been very successful in achieving wide spread deployment. Many C libraries support a simple API to generate SYSLOG messages and it is not unlikely that the availability of an easy to use and relatively portable API lead to wide-spread acceptance by software developers. From a conceptual point of view, SYSLOG notifications with structured data elements are close to unconfirmed SNMP notifications and it can be expected that mappings will be defined. The concept of SYSLOG relays is close to SNMP proxies, except that SNMP does not directly support a co-located relay and collector (although a fancy configuration of a proxy might achieve a similar effect).

There are some notable differences in the security services supported. SYSLOG provides a mechanism to sign notifications while SNMP provides confirmed notification delivery and access control for outgoing notifications. The SYSLOG signature mechanism might be easy to retrofit into the SNMP protocol, but confirmed notifications are more difficult to add to SYSLOG.

1.4.4 Flow Information Export Protocol (IPFIX)

In the 1990s, it became popular to collect traffic statistics about traffic flows in order to track the usage of networks. A flow is a set of IP packets passing an observation point in the network during a certain time interval. All packets belonging to a particular flow have a set of common properties [43].

Flows are a convenient mechanism to aggregate packet streams into something meaningful for the network operator. For example, a network operator might collect flow statistics for specific types of traffic in the network such as all email (SMTP) traffic, all web (HTTP) traffic, or all voice (SIP and RTP) traffic. In addition, the operator might collect flow statistics for the different customers.

In order to assign packets to flows, it is necessary to derive a set of properties. In general, the set of properties is constructed by applying a function to certain values and characteristics associated with a captured packet [49]:

1. One or more network layer packet header fields, transport layer header fields, or application layer header fields (e.g., IP destination address, destination port, and RTP header fields).

2. One of more characteristics of the packet itself (e.g., number of MPLS labels).

3. One or more fields derived from packet treatment (e.g., output interface).
Some early approaches to collect traffic statistics using the SNMP framework resulted in the Remote Monitoring (RMON) family of MIB modules [56]. While the RMON technology achieved wide deployment for collecting overall link statistics, it did not support the notion of flows very well. The Realtime Traffic Flow Measurement (RTFM) framework [3], centered around a Meter MIB module [4], details the criteria used by a meter to assign packets to flows. The RTFM framework is like the RMON framework based on SNMP technology. Even though the RTFM framework was quite flexible from the very beginning, it only achieved some moderate level of support.

In the 1990s, several vendors also started efforts to export flow information from their routers using special purpose protocols, most notably Cisco’s NetFlow [9], which evolved over time to the current version 9. Cisco’s NetFlow has been very successful in achieving significant deployment. This led to the formation of a working group in the IETF to standardize a flow information export protocol. The evaluation of candidate protocols [29] lead to the selection of Cisco’s NetFlow version 9 as a starting point for a standardization effort, which then lead to the IP Flow Information Protocol (IPFIX) [49] framework.

**Deployment Scenarios**

The IPFIX protocol can be used in several different scenarios. Figure 1.11 shows a deployment scenario where two IPFIX devices (typically IP router) report flow statistics to a single collector. The collector stores the data in a local database. The dashed lines indicate that the collector may include another meter and exporter process to report further aggregated statistics to another collector.

![IPFIX Deployment Scenario](image.png)

**Figure 1.11: IPFIX Deployment Scenario**

IPFIX devices can include several observation points (marked as circles in Figure 1.11) and meters. It is also possible to have multiple exporting processes on a single device, but this is not indicated in Figure 1.11. The internal structure
1.4. INTERNET MANAGEMENT PROTOCOLS

typically depends on the configuration of the device. Note that observation points and even complete meters might be running on different line cards on a high-end modular router.

Architecture

The IPFIX protocol follows a variable-oriented approach to export flow statistics [49]. Since IPFIX is a unidirectional protocol, the architecture remains fairly simple.

Figure 1.12 provides an architectural view. Like all other protocols discussed in this chapter, IPFIX has a transport subsystem which supports several different transport models. Since IPFIX is relatively new, there is currently only a single message format and message processing model. A message version number has been designed into the protocol in order to be able to deal with any changes and additions that might be necessary in the future.

There are two IPFIX functions: The exporting process is responsible for generating IPFIX messages and sending them to a collector. The collecting process receives IPFIX messages and processes them. The IPFIX requirements document also mentions concentrators capable to merge multiple IPFIX message streams and proxies, which relay IPFIX message streams [43]. None of these concepts is, however, explicitly discussed in the protocols specifications and this is why they also do not appear in Figure 1.12.

The data values transmitted in IPFIX messages belong to so called information elements. An information element is primarily identified by its element identifier, which is a numeric identifier. Globally unique element identifiers can be allocated via the Internet Assigned Numbers Authority (IANA). Vendor-specific element identifiers are created by combining the element identifier with an enterprise identifier. The later can be allocated via IANA on a first-come first-serve basis. Information elements can be scoped to a specific metering or exporting process, or they can be scoped to a specific flow measured at a specific
observation point, which again is identified by an observation identifier.

IPFIX messages can be exchanged over different transport protocols. The required to implement transport protocol is the Stream Control Transmission Protocol (SCTP) [39]. SCTP provides multiple independent streams in an association. IPFIX explores this by sending meta data and configuration data over SCTP stream zero while multiple additional streams may be allocated to send measurement data records. The optional transports for IPFIX are UDP and TCP.

Data Modeling

IPFIX uses an ad-hoc informal template mechanism for data modeling [42]. Information elements are defined by defining the following properties:

- The mandatory name property is a unique and meaningful name for an information element.

- The mandatory numeric element identifier property is used by the IPFIX protocol to identify an information element. The element identifier can be globally unique or it can be scoped by an enterprise identifier.

- The textual description property specifies the semantics of an information element.

- The mandatory data type property of an information element indicates the type of an information element and its encoding. The basic data types include signed and unsigned integers with different width (8, 16, 32, 64 bits), floating point numbers (IEEE 32-bit and 64-bit formats), a boolean type, octet string and unicode strings, several data and time types, and types for IPv4, IPv6, and MAC addresses.

- The mandatory status property indicates whether a definition is current, deprecated, or obsolete.

- The optional numeric enterprise identifier property must be present if the element identifier is not globally unique.

- Additional optional properties can indicate units, range restrictions, references and provide further type semantics (e.g., whether a numeric type is acting as a counter or an identifier).

The IPFIX specification [42] includes a non-normative XML representation of the definitions and a non-normative XML schema for the XML notation.

Protocol Operations

The IPFIX protocol is essentially a unidirectional message stream. At the time of this writing, IPFIX only has a single message format which includes a version
number, a length field, a sequence number, the observation domain identifier, and the export time [10].

![Figure 1.13: Protocol operations of IPFIX](image)

The IPFIX message header is followed by one or more sets. Sets can be either data sets, template sets, or options template sets:

- **Template sets** are collections of one or more template records that have been grouped together in an IPFIX message. A template record defines the structure and interpretation of fields in data records. Template sets are identified by a template number which is scoped by the transport session and an observation domain.

- **Data sets** consist of one or more data records of the same type, grouped together in an IPFIX message. Data records refer to previously transported template records (using the template number). The template record defines the structure and the semantics of the values carried in a data record.

- **Options template sets** are collections of one or more option template records that have been grouped together in an IPFIX message. An option template record is a template record including definitions how to scope the applicability of the data record. Option template records provide a mechanism for an exporter to provide additional information to a collector, such as the keys identifying a flow, sampling parameters, or statistics such as the number of packets processed by the meter or the number of packets dropped due to resource constraints.

Template records establish the names and types of the values contained in subsequent data records. Compared to the name-value encoding used by SNMP and SYSLOG, IPFIX has less overhead since names are not repeated in each data record. This can significantly reduce the encoding and bandwidth overhead and makes IPFIX more efficient and scalable.

**Security**

The data carried by the IPFIX protocol can be used to analyze intrusion attacks or it might be the basis for billing processes. It is therefore necessary to properly maintain the integrity of IPFIX messages and to ensure that data is delivered from an authorized exporting process to an authorized collecting process. Furthermore, it is desirable to prevent disclosure of flow data since flow data can be highly sensitive.
The IPFIX protocol mandates that the Datagram Transport Layer Security (DTLS) protocol [45] is implemented for the SCTP transport. DTLS has been chosen instead of TLS for SCTP since SCTP supports unreliable and partially reliable modes in addition to the default reliable mode. For the optional TCP transport, TLS must be implemented to secure IPFIX.

It should be noted that IPFIX does not have any mechanisms for access control. The configuration of an IPFIX exporter therefore implicitly implements an access control policy for flow data.

Discussion

IPFIX is derived from Cisco’s NetFlow, a very successful flow information export protocol. Since IPFIX can be seen as an evolutionary improvement of NetFlow, it can be expected that IPFIX also gets widely implemented and deployed. It should, however, be noted that the requirement of the SCTP transport support raises the bar for conforming implementations. Only very recent operating systems provide full SCTP support as part of the operating system kernel.

The data modeling part of the IPFIX framework may be improved. The informal ad-hoc notation makes it difficult to build tools and it would be desirable to achieve some level of integration with the data models used by other management protocols.

1.5 Conclusions

Work on Internet network management protocols started in the late 1980s. The SNMP protocol was created at that time as a temporary solution until full OSI management protocols would take over [8]. In the mid 1990s, it became clear that this will never happen.

The Internet management protocol development and standardization was for a long time almost exclusively focused on the SNMP framework, to a large extend ignoring that many other not well standardized protocols became widely used to manage networks. In addition, management interfaces originally designed for human interaction, such as command line interfaces, became increasingly used for management automation.

It is only recently that the IETF network management community has started to realize that effective network management requires a toolbox of protocols and that a “one size fits all” approach is unlikely to succeed to address all requirements. This change of mind-set is very important, especially since many of the requirements are also much better understood today than they were some 10 years ago.

The real challenge with a multi-protocol approach is to keep the protocols aligned where necessary. Such alignment concerns mainly security related issues as it is for example operationally very important to harmonize key management procedures and access control lists. With a convergence to secure transports
and the movement to support different certificate formats by secure transports, it can be hoped that this problem will be dealt with over time.

A second area where some level alignment is very important is data modeling. It is crucial that different and potentially overlapping management interfaces operate on the same underlying data structures as this improves consistency and reduces costs. Since data modeling efforts are not as far developed as some of the protocol definitions, it can be expected that work on multi-protocol Internet management data modeling remains an active area for future research and development.

Acknowledgements

The work on this chapter was supported in part by the EC IST-EMANICS Network of Excellence (#26854).
Bibliography


