The Simple Times™

THE QUARTERLY NEWSLETTER OF SNMP TECHNOLOGY, COMMENT, AND EVENTS
VOLUME 5, NUMBER 1 DECEMBER, 1997

The Simple Times is an openly-available publication devoted to the promotion of the Simple Network Management Protocol. In each issue, *The Simple Times* presents technical articles and featured columns, along with a standards summary and a list of Internet resources. In addition, some issues contain summaries of recent publications and upcoming events.

In this Issue:

The Simple Times is available as an online journal in HTML and PostScript. New issues are announced via an electronic mailing list. For information on subscriptions, see page 18.

SNMPv3 Working Group - A View From the Chair

Russ Mundy, Trusted Information Systems

Thanks to lots of hard work by a number of people, the core set of specifications for SNMPv3 has successfully completed the IETF Last-Call process. This will result in the specifications being published as RFCs with the status of Proposed Standards.

Before giving more details on how we reached this point and where I see us going from here, I want to express my special thanks to the following individuals that made up the specification editor teams. Their commitment and actually doing the hard work of writing down the Working Group consensus made our core set of specifications possible.

- For the Architecture for Describing SNMP Management Frameworks specification;
  - David Harrington, Cabletron Systems Inc.
  - Randy Presuhn, BMC Software Inc.
  - Bert Wijnen, IBM T. J. Watson Research
- For the SNMPv3 Applications specification;
  - David Levi, SNMP Research Inc.
  - Paul Meyer, Secure Computing Corporation
  - Bob Stewart, Cisco Systems Inc.
- For the Message Processing and Dispatching specification;
  - Jeffrey Case, SNMP Research Inc.
  - David Harrington, Cabletron Systems Inc.
  - Randy Presuhn, BMC Software Inc.
  - Bert Wijnen, IBM T. J. Watson Research
- For the View-based Access Control Model specification;
  - Bert Wijnen, IBM T. J. Watson Research
  - Randy Presuhn, BMC Software Inc.
  - Keith McCloghrie, Cisco Systems Inc.

- For the User-based Security Model specification;
  - Uri Blumenthal, IBM T. J. Watson Research
  - Bert Wijnen, IBM T. J. Watson Research

Additionally, I'd like to thank Mike O'Dell as the responsible Internet Engineering Steering Group (IESG) Area Director who has provided significant motivation and support. It is clear to me that the Working Group would have not been able to make such a substantial amount of forward progress without Mike's leadership.

This seems like a reasonable time to give my views about where we are. A few words on the Internet Standards Process seem in order since the SNMPv3 core specifications are about to be published as Proposed Standards.

The Internet Standards Process is described in RFC 2026. I encourage everyone to read this RFC to get a broader understanding of the various types and maturity designations for RFCs. Additional information on the Process as well as information on the current status of RFCs is contained in the Internet Official Protocol Standards (STD 1) currently published as RFC 2200.

The designation of the core specifications from the Working Group as Proposed Standards indicates that they have entered on to the "Internet Standards Track". The objective of specifications on this "Track" is to become full Internet Standards but there are several steps along the way. As described in RFC 2026, the subsequent maturity levels on the "Internet Standards Track" are Draft Standard and Internet Standard. The Proposed Standard status of the specifications indicates a number of things, for instance:

- The specifications should be expected to have some changes before advancing to more mature phases of the Standards Process.

- Though implementation and operational experience are not required for a Proposed Standard status, independent implementation and interoperability are required to advance to the next maturity level (Draft Standard).

- Normally, specifications will remain at the Proposed Standard level for at least six months. If all the defined requirements for Draft Standard are met, they may be advanced at that point. However, it is not uncommon for specifications to "stay in grade" for longer than the required six months. Depending on the reasons for "staying in grade", new RFC(s) may or may not be published.

I believe that the Working Group needs to develop a sufficient set of specifications to provide the basis for standardized, secure network management. However, we should not "re-invent the wheel". This has been the general guidance since the formation of the SNMP Advisory Team subsequent to the 36th IETF in Montreal (June of '96). One of the results of this guidance has been the decision to make use of the current set of network management RFCs already on the "Standards Track", (RFCs 1902-1908). Using these specifications should substantially shorten the time needed to provide the sufficient set of specifications.

Where do I see the Working Group going from here? Identifying SNMPv3 implementations and promoting interoperability tests must be done before the specifications can be advanced to Draft Standard. An important part of this activity will be determining more specifically what is meant by SNMPv3 interoperability. This will likely involve developing Applicability Statement(s) for the specifications. Additionally, the Working Group has indicated support for providing a simple, understandable overview of SNMPv3 and more documentation for the transition from and the coexistance with the earlier versions of SNMP. It is also likely that some changes will be required to RFCs 1902-1908 though these are expected to be fairly small.

The Working Group met at the 40th IETF in Washington, D.C. in December '97. Besides celebrating the advancement of the core specifications to Proposed Standard, the Working Group heard implementation reports, discussed interoperability testing in the early summer, and gathered suggestions for future work and an update of the charter.

The Evolution of Architectural Concepts in the SNMPv3 Working Group

David Harrington, Cabletron

This article will review the evolution of some concepts of "An Architecture for Describing SNMP Management Frameworks". The concepts derive from earlier technical proposals, and in response to policy and technical requirements identified during the process of developing a secure SNMP, evolved into an architecture for describing SNMP frameworks.

The article will quickly review the efforts from which the concepts derive, the constraints and technical goals for SNMPv3, and then discuss the major architectural concepts and how they evolved into their current form.

It is expected that the reader will be acquainted with SNMPv1 concepts, and have read the document "An Architecture for Describing SNMP Management Frameworks". A knowledge of the SNMPv2 effort, while not required, will aid understanding.

Historical Background

In 1992-1993, the SNMPv2 Working Group developed a security model based on parties to an SNMP transaction. Known as SNMPv2p, this design was moved to historic status when the SNMPv2 Working Group decided that a proposed user-based security model was much simpler, and was more likely to be understood and deployed.

In December 1995, the SNMPv2 Working Group was deactivated, and independent coalitions were formed to study and experiment with alternative security and administrative framework designs. The two most prominent approaches, SNMPv2u and SNMPv2*, both employed a user-based security model as a simpler alternative to the party-based model. They had many points in common in the design of the administrative framework, but the two approaches had philosophical differences.

SNMPv2u supported early standardization of the security features and a minimal specification, and deferred standardization of features for managing large networks to encourage rapid deployment of simple agents.

SNMPv2* supported concurrent standardization of security and scalability features to ensure that the security design adequately addressed issues of proxy, trap destinations, discovery, and remote configuration of security, to enable effective management of medium and large networks.

In August 1996, the Security and Administrative Framework Evolution Advisory Team was formed to recommend a single approach that would resolve the differences between SNMPv2u and SNMPv2*. At the San Jose IETF meeting (December 1996), the team published its recommendation.

In March 1997, the SNMPv3 Working group was chartered to continue the work of the SNMPv2 Working Group to define a standard for SNMP security and administration.

In this article, an unqualified reference to "the Working Group" refers to the SNMPv3 Working Group.

Working Group Objectives

The Objectives of the SNMPv3 Working Group reflect many sources. Some constraints were stated by the Area Directors or the Chair or were the result of community needs; some design requirements were formalized in the charter, or inherited from the SNMPv2 Working Group; additional design objectives were developed by consensus of the active participants as the work progressed.

Working Group Constraints

The Area Directors have stressed the need for the Working Group to succeed. The Internet community needs a secure protocol for network management to make it possible to not only monitor networks, but to manage them.

The Working Group must have realistic expectations; the documents and the design do not need to be perfect. The documents must be done quickly, and the standard must be useful, functional and deployable, without constraining implementation decisions. It should be reasonably possible to move portions of the architecture forward in the standards track, even if consensus has not been reached on all pieces, and to correct problems later without destabilizing the whole proposal.

The focus must be on specifying a single standard approach in a timely manner. The work will start with the recommendation of the advisory team. The recommendation is based on reusing concepts, technical elements and documentation from SNMPv2u and SNMPv2*. The focus must be on completing the existing proposals, not on developing new proposals. RFCs 1902-1908, the SNMPv2 Draft Standard, should not be changed if that can be avoided. An initial set of documents should be ready for Working Group "Last Call" by Munich (August 1997). Work on additional output should be deferred until after that date.

Working Group Design Requirements

To be successful, the Working Group must produce a set of documents that will provide a standard for the next generation of core SNMP functions, and the proposed design must meet these requirements.

SNMP must be secure in a way that is useful for the people operating networks, especially support for secure SETs, which is the most important deficiency in SNMPv1.

The wide range of operational environments with differing management demands that comprise the Internet community must be accommodated. It must be relatively inexpensive to deploy a minimal conforming implementation, but also possible to support additional features useful for managing large networks.

SNMP should be able to be extended as new mechanisms or protocols become available or unanticipated aspects of network operation and management arise.

The documents should be clear and unambiguous.

It should be easy to setup and maintain the security and administrative configuration.

SNMP should be kept as simple as reasonably possible.

Working Group Design Objectives

A number of design decisions were made by the Working Group to keep the work focused, and some functional goals were acknowledged as desirable, but not required.

The Working Group should use as much as practical of the concepts, technical elements and documentation from the SNMPv2u and SNMPv2* activities, since implementation experience has already been gained for features of these designs.

It is highly desirable that the architecture be able to ease the transition to SNMPv3 from earlier versions, including SNMPv1, SNMPv2p, SNMPv2c, SNMPv2u, and SNMPv2*. It should also be flexible enough to support a wide range of possibilities for future frameworks.

Since new technologies may emerge, or existing technologies may prove inadequate, the architecture should make it possible to incrementally upgrade portions of SNMP, without disrupting an entire SNMP framework. To facilitate the development, testing, and deployment of alternative solutions to supplement the industry standards, the architecture should permit the integration of enterprise-specific modules.

Since there are many more agents than managers in a typical network environment, any complexity should be incurred in the manager rather than the agent. Where the overhead to support features needed for large networks must be present in all entities, even those that do not support the feature, the overhead should be minimized as much as feasible.

A standard MIB for specifying trap destinations should be part of the output of the Working Group, since this is needed by a number of SNMP-related Working Groups.

The Architectural Concepts

Many of the concepts have a long history in the secure SNMPv2 efforts, and were debated by the SNMPv2 Working Group. The concepts discussed in this article, however, will start with the recommendation of the advisory team, which included the concepts inherited from SNMPv2. This article will review the efforts of the advisory team to identify components of the multiple proposed SNMPv2 architectures which could be merged into one architecture. It will follow the evolution as concepts of SNMPv2/SNMPv3 coalesced into an architecture.

Modularity and Controlling Side-Effects

The advisory team studied the nature of the differences between the SNMPv2u approach and the SNMPv2* approach and, to a lesser degree, the other approaches which had been suggested for SNMPv2. They also studied implementation reports and their own implementation experiences with SNMPv1, SNMPv2p, SNMPv2u, SNMPv2*, and SNMPv2c to understand what issues made implementation difficult.

One of the major problems noted in the discussions regarding converging the SNMPv2u approach and the SNMPv2* was side-effects. There were many instances of an item important to one "camp", on which the second camp was willing to compromise, until it was studied further, and it was found that the compromise caused unwanted side-effects to something important to the second camp. This happened frequently, and made it difficult to find reasonable compromise solutions.

It also became obvious that the proposals could not simply be merged by taking pieces of text from A, adding them to pieces of text from B, and publishing a set of merged documents. Even when the advisory team attempted to resolve conflicts by selecting one approach over the other, the side-effects that occurred caused those simple decisions to be revisited again and again.

Software engineering techniques, namely modularity, encapsulation, tight cohesion, and loose coupling, can be used to help resolve the side-effect problem. Starting with those elements that were common to SNMPv2u and SNMPv2*, such as authentication, encryption, timeliness checking, and view-based access control, the team identified those parts of the designs which were not generally contentious. Then the team defined interfaces to hide the internal processing and data of those modules, to protect them from side-effects of changes external to the module.

This was not a very difficult task for the areas that were not highly contentious, since SNMPv2u and SNMPv2* separated user-based security from the administrative model to a large degree, replaceable components of user-based security were already modular, some implementors had added view-based access control to SNMPv1, and the use of existing components made SNMPv2c relatively simple to implement.

Consistent Terminology

It was somewhat more difficult to isolate functionality when segments of the SNMP community disagreed about the semantics of existing terms. It was sometimes possible to defer the resolution of specific issues by limiting the usage of a term to the internals of a module. Some data elements were needed by more than one module, but the actual meaning of the element changed slightly depending on which module used it. In some instances, the team could resolve the problem by calling the same item by different names in different modules, and have one common name for use in the interface between modules. This clarification of the terms helped to make debate over the issues easier.

The greatest problem was the MIB, where objects that were related to separate areas of processing were combined into one large MIB module, and objects were sometimes used in different ways by different elements of procedure, rather than reflecting the separate nature of the various processing modules that operated on the objects. By defining separate MIB modules for different processing areas, managed objects could be limited to one processing module, and resolution of issues regarding those objects could be deferred until discussion of that module was on the table. When managed objects were used by multiple processing areas, duplicate objects could be defined, and the variations in semantics caused by different usage of the objects could be identified. It was always possible that the duplicated objects could be merged later.

Separate Document Advancement

Many of the side-effects that had plagued the SNMPv2 efforts revolved around features that were considered important for network management, but not necessarily for device management. These features include proxy, informs, and remote configuration of the security and administration MIB objects. These are important issues when managing large networks, but were considered unnecessary overhead for a minimal conforming implementation. If the MIB objects and the processing for these functional areas could be isolated from those functional areas that were important for all SNMPv2 entities, it might be possible to advance some documents without waiting to resolve all the problems. This had been done for RFCs 1902-1908, and RFC 1901 (SNMPv2c) and 1909-1910 (SNMPv2u) had been published separately as well.

More importantly, the debate over proxy led to agreements that if an adequate interface could be identified and defined, then proxy could be developed later as a separate module.

Based on the precedent of RFCs 1902-1908 and the proposed deferral of proxy, the advisory team recommended a modular SNMP architecture, with fixed interfaces, and documents that could be advanced individually. This approach would use modularity, encapsulation, tight cohesion, and loose coupling to minimize the side-effects, and make independent advancement of SNMP modules possible.

All very pretty, but the devil is in the details. The SNMPv3 Working Group was chartered and the real work began.

The Architecture, Interfaces, and Implementations

The advisory team specified detailed APIs, defined or implied implementation constraints and got the split between modules wrong. They had not followed and responded to external developments while they worked in relative isolation. They did not anticipate several evolving demands on the architecture, such as non-SNMP access to authentication, encryption, and access control services; support for the widely deployed SNMPv1; or accommodation of the various flavors of SNMPv2 that were being deployed while they worked.

It is unacceptable to dictate to enterprises how they should implement their code. Using application programming interfaces was rejected, but they could be defined as application service interfaces, so they only describe the nature of the data that crosses boundaries between modules. Enterprises merely need to understand the concepts of the data that must be passed from module to module.

However, that still had the effect of dictating how an enterprise should build their implementation in specific modules - still unacceptable. “The Architecture for SNMP Management Frameworks” was changed to “An Architecture for SNMP Management Frameworks”, to show that enterprises could use a different architecture if they chose.

But that still implied constraints for those who were willing to follow this architectural model. The architecture was useful for describing the concepts, but the constraint for a modular implementation, even if only implied, was unacceptable.

An architecture could only be used to describe the concepts and the conceptual interfaces contained in the conceptual architecture - thus, “An Architecture for Describing SNMP Management Frameworks”.

Support for Diverse Environments

The environments in which SNMP must execute vary widely. Producers of simple managed devices want to keep the resources used by SNMP to a minimum. At the same time, there is a need for more complex configurations which can spend more resources for SNMP and thus provide more functionality.

The SNMPv3 architecture keeps the competing requirements of these environments in balance and allows the more complex environments to logically extend the simple environment. Whenever possible, the costs associated with allowing support for extended functionality is borne only by those who extend the functionality.

The subsystem and modular approach aids in this goal by permitting modules to define a minimal implementation approach for those environments with severe resource constraints, more complete implementation approaches for those environments where resources are less constrained, and by permitting support for multiple approaches within a given node.

SNMP Entity

An SNMP entity consists of an SNMP engine and one or more associated applications. The engine contains a Dispatcher, a Message Processing Subsystem, a Security Subsystem, and an Access Control Subsystem. Applications use the services of the engine for sending and receiving messages, authenticating and encrypting messages, and controlling access to managed objects.

```
+-------------------------------------------------------------------+
|  SNMP entity                                                      |
|                                                                   |
|  +-------------------------------------------------------------+  |
|  |  SNMP engine (identified by snmpEngineID)                   |  |
|  |                                                             |  |
|  |  +------------+ +------------+ +-----------+ +-----------+  |  |
|  |  |            | |            | |           | |           |  |  |
|  |  | Dispatcher | | Message    | | Security  | | Access    |  |  |
|  |  |            | | Processing | | Subsystem | | Control   |  |  |
|  |  |            | | Subsystem  | |           | | Subsystem |  |  |
|  |  |            | |            | |           | |           |  |  |
|  |  +------------+ +------------+ +-----------+ +-----------+  |  |
|  |                                                             |  |
|  +-------------------------------------------------------------+  |
|                                                                   |
|  +-------------------------------------------------------------+  |
|  |  Application(s)                                             |  |
|  |                                                             |  |
|  |  +-------------+  +--------------+  +--------------+        |  |
|  |  | Command     |  | Notification |  | Proxy        |        |  |
|  |  | Generator   |  | Receiver     |  | Forwarder    |        |  |
|  |  +-------------+  +--------------+  +--------------+        |  |
|  |                                                             |  |
|  |  +-------------+  +--------------+  +--------------+        |  |
|  |  | Command     |  | Notification |  | Other        |        |  |
|  |  | Responder   |  | Originator   |  |              |        |  |
|  |  +-------------+  +--------------+  +--------------+        |  |
|  |                                                             |  |
|  +-------------------------------------------------------------+  |
|                                                                   |
+-------------------------------------------------------------------+
```

Dispatcher

The Dispatcher coordinates communications between the subsystems, and differentiates between multiple co-existent modules within a subsystem. It determines to which application an incoming PDU should be directed. It coordinates with potentially multiple transport mappings.

Message Processing Modules

The message format for SNMPv2 was an area of contention because, once established, the format would be fixed and immutable, and this meant it had to be very close to perfect for all concerned. The requirements of various segments of the community were incompatible. The SNMPv2 Working Group could not reach consensus on a single format. SNMPv2 was left with no standard secure message for transferring management data between nodes.

By defining message processing in modules that can be replaced or supplemented, the burden of perfection is eliminated. The Working Group can accept the current proposal to allow the standard to advance, but the message processing, and its defined format, can be replaced or supplemented if necessary.

One concern regarding the SNMPv2 message format was its sheer size. The size of the proposed SNMPv3 message format is also large. If somebody can design a message format that is smaller and more efficient than the current design, the existing format can be superseded without necessarily changing the other conceptual modules of the architecture.

That said, changing the message format could have a very negative effect on interoperability, so the designer of a new message format is likely to face a steep uphill battle to convince the community to use the new format. The architecture allows for new message formats, but the marketplace is expected to discourage the development of new formats.

Multiple message format modules can exist simultaneously, to permit support for new, and especially for old, message formats. Message Processing Modules can be developed for SNMPv1 and various flavors of SNMPv2.

Security Modules

The starting design included a security module taken almost directly from the User-based Security Model used in SNMPv2u and SNMPv2*. This module inherited the desired ability to supplement or upgrade existing authentication and encryption mechanisms as new mechanisms became available. The design also permitted supplementing or replacing the entire User-based security module. The modules in the Security Subsystem may be controlled by a Message Processing Module.

Support for previous security formats, such as SNMPv1 communities, can be added fairly easily. For example, implementors may choose to support communities by mapping them to user-based security identifiers, or by defining a separate module for community-based security.

There have been a number of security models discussed during the evolution of SNMPv2/SNMPv3. Some models, such as the party model, require parameters that do not fit well with the requirements of the User-based model. Security processing is defined in modules, and the message processing module can call the appropriate module for a particular message. This permits support for security models with different parameter requirements, and to support co-existent security models.

Application Modules

SNMPv1 describes agents, which contain specific functionality, and managers, which execute *management applications*.

The starting design defined an engine which contained modules for message processing, message security, and local processing of PDUs. It also identified *applications* that used the engine. In keeping with traditional SNMP concepts, the applications executed only on managers.

Proxy was included in the design as a management application. Proxy has historically been viewed as an agent functionality, even though proxy agents generate requests and receive notifications and responses, which are manager functions.

The design showed Inform PDUs being issued by management applications, while trap PDUs were issued by the local processing module in the engine. To send traps, the local processing module needed a trap table, but then so did informs. The Working Group debated combining informs and traps into one module, but would this be a module in the engine (i.e. in an agent) or an application external to the engine (i.e. in a manager)?

The concept developed that *applications* were not limited to execution on a manager; an agent could be described as executing specific applications, such that all local processing is just a special case of an application. All traditional processing can be broken down to five types of applications - request generation, response processing, notification sending, notification receiving, and proxy processing.

A combination of these application modules would exist in a traditional agent; a different combination would exist in a traditional manager. Other existing implementations contain non-traditional combinations of applications, such as an implementation that emits traps, but does not accept requests.

This line of thinking resolves some of the SNMPv2 issues such as whether informs were generated by agents or only by managers, and whether only agents should have agentIDs/snmpIDs. The concepts of agent and manager become only specific subsets of the possible configurations of SNMP entities.

Additional applications can be defined that allow extensions to address new aspects of network operation and control, such as applications for distributed management.

Since SNMPv1 is defined in terms of traditional agent and manager functionality, and traditional agents and managers can be described as a combination of *applications*, then *applications* can be used to describe SNMPv1 functionality. Therefore, the architecture can be used to describe not only an SNMPv3 framework, but also the SNMPv1 framework, and possibly the various SNMPv2 proposed frameworks.

Access Control Modules

The starting design provided no ability to use SNMP authentication or access control services except for SNMP messages. The developing Distributed Management (DISMAN) Working Group designs would benefit from access to the same services, not necessarily using an SNMP message. It might also be desirable to use some services and not others. For example, a Java script executing under DISMAN control could be authenticated using a Java authentication mechanism but access SNMP managed objects. This access should be subject to the same access control as an SNMP transaction. So it would benefit DISMAN if the SNMP engine provided the access control service, even when the SNMP engine did not provide the authentication service.

While access control is commonly considered part of security, analysis of the usages showed that there was a distinct difference between the security processes applied to messages and the security processes applied for access control. Application modules need to use access control services while processing objects, but authentication and encryption are used while processing messages. A message header is used while processing message security, but the PDU is used while applying access control.

There are certainly common factors, such as identification of the principal and the level of security processing requested. These can be provided through the service interface. The distinction was sufficient to get consensus among the active participants that the MIB objects could be described in independent MIB modules, and the elements of procedure for access control could be described in a module separate from the message security procedures.

If access control is a separate module called only by applications, it can be used by non-SNMP applications running on the same system. Consistent access control is possible without the onerous operational task of maintaining parallel configurations of access control policy databases for SNMP and non-SNMP applications.

Summary

This article has reviewed the evolution of some concepts of “An Architecture for Describing SNMP Management Frameworks”. The concepts are not new; they have been refined from earlier versions of SNMP.

Hopefully, this architecture will encourage the deployment of SNMPv3 and allow continued advances in SNMP as the needs of network operation and control change over time.

Security Features of SNMPv3

Uri Blumenthal, IBM Watson Research
Bert Wijnen, IBM Watson Research

The ultimate question a security system has to answer is: "Should I permit this operation?" Naturally, in order to be able to do that, several lesser questions have to be answered first:

- Is the message specifying an operation unaltered and timely?
- Who requested the operation to be performed?
- What objects are accessed in the operation?
- What are the rights of the requester with regard to the objects of the operation?

The SNMP architecture divides the responsibility for providing the answers to the questions above between two subsystems: The *Security Module* is tasked with the first two questions, while the rest fall to the *Access Control Module*.

To deal with the "who" question, it is necessary to define that "who." Thus, the SNMP architecture introduces the term "principal." A *principal* is the "who" on whose behalf services are provided, or processing takes place. A principal can be, among other things, an individual acting in a particular role; a set of individuals, with each acting in a particular role; an application or a set of applications; and any combination thereof.

Since a principal may have to be identified both inside and outside a given security model, there is a need to have two identities: one specific for that security model that may take any shape convenient for that security model (for example, an ASN.1 OID to identify a Party, or an octet string to identify a User), and the other one, security-model-independent, that can be used outside a particular security model. This last one, called *securityName*, is a human-readable string.

Multiple security models

It is imprudent to expect that the view we have on how security is accomplished can survive unchanged forever, or even for the lifespan of the protocol. Old security requirements may go away and new ones may come up. Accommodating those new requirements may demand that not only the cryptographic algorithms but the whole approach to the problem must be changed. It necessarily means, that it is not enough to be able to replace cryptographic protocols - the security model itself may have to be replaced.

Therefore, the proposed SNMP architecture supports multiple security models. Moreover, it allows several different security models to be used simultaneously by an SNMP entity. SNMPv3 messages carry a field in the header that identifies by which security model the message must be processed.

To guarantee interoperability, one security model must be defined and implemented by every compliant SNMPv3 entity. It is called *User-based Security Model* (USM) and its model number is 3.

User-based Security Model

Here are the primary threats the USM defends against:

- An unauthorized entity may alter an SNMP message (issued on behalf of an authorized principal) in-transit. (Modification of Information)
- An attempt of an unauthorized entity to perform an operation by assuming the identify of an authorized one. (Masquerade)
- Delay or replay of the messages *to an extent greater than can occur in natural conditions of network service.* (Message Stream Modification)
- Unauthorized entities can see the contents of SNMP data exchange. (Disclosure)

It was decided that neither *traffic analysis* nor *denial of service* need to be defended against, because (a) such defense is nearly impossible to achieve; and (b) the threat is not significant enough.

Identification

A *principal* in USM is represented by a *user* and identified by *userName*. The main purpose of a user is to hold secret keys and some security-related information like what cryptographic algorithms to use, etc. The USM internal principal identifier *userName* maps to the security-model-independent identifier *securityName* by an identity transform. The *userName* is therefore a human-readable string.

The USM identifies SNMPv3 entities by their *snmpEngineID*. SNMPv3 entities can be traditional agents running on managed devices, traditional network management stations, mid-level managers, etc. The *userName* is needed for auditing and authorization purposes while the *snmpEngineID* is needed to identify the target of an SNMP operation.

When two SNMP engines communicate, one is designated to be the *authoritative* SNMP engine. When an SNMP message contains a payload that expects a response (for example a get-request, get-next-request, get-bulk-request, set-request or inform-request PDU), then the receiver of such messages is authoritative. When an SNMP message contains a payload that does not expect a response (for example, an snmpV2-trap, response or report PDU), then the sender of such a message is authoritative. It is important because (a) the keys that are owned by the principal but stored in SNMP engines are localized to the authoritative engine (see below), and (b) the timeliness indicators of the authoritative engine are, well, authoritative. (The non-authoritative SNMP engine maintains its notion of the timeliness values from the authoritative engine and updates its notion as appropriate to keep them in sync with the authoritative one).

USM MIB

USM defines a MIB for the purpose of remote configuration. While it is unreasonable to quote the whole MIB group here, we provide the structure of the *usmUserTable*, since the items we will be describing are related closely to that table.

```
UsmUserEntry ::= SEQUENCE
    {
        usmUserEngineID          SnmpEngineID,
        usmUserName              SnmpAdminString,
        usmUserSecurityName      SnmpAdminString,
        usmUserCloneFrom         RowPointer,
        usmUserAuthProtocol      AutonomousType,
        usmUserAuthKeyChange     KeyChange,
        usmUserOwnAuthKeyChange  KeyChange,
        usmUserPrivProtocol      AutonomousType,
        usmUserPrivKeyChange     KeyChange,
        usmUserOwnPrivKeyChange  KeyChange,
        usmUserPublic            OCTET STRING,
        usmUserStorageType       StorageType,
        usmUserStatus            RowStatus
    }
```

As you see, the table entry holds: a principal identifier (*usmUserName* and *usmUserSecurityName*); a reference to another entry in this table from which this entry was cloned; authentication and privacy protocol discriminants (which tell us what protocol to use); the typical SNMP table row maintenance objects (*usmUserStorageType*, *usmUserStatus*); and several objects related to remote key update (*usmUserAuthKeyChange*, *usmUserOwnAuthKeyChange*, *usmUserPrivKeyChange*, *usmUserOwnPrivKeyChange*), that we will discuss later on.

Since all the keys are *localized* (see below) to a particular SNMP engine, it is necessary to store the identifier of the SNMP engine (*usmUserEngineID*), for which the given keys for the user *usmUserName* are prepared (since they will not work with any other user, or with any other engine).

Keys

A *user* "owns" the *keys* - secrets that he shares with the remote SNMP engines he manages. These secrets are used by authentication and privacy protocols. Keys are octet strings, and usually are derived from passwords typed in by human users. USM provides for two keys per user per SNMP engine: one key for message authentication, and the other one for encryption. Since these secrets are not retrievable, there are no objects in the MIB to represent the keys directly; the cryptographic keys are not "visible."

Password to key:
Even though computers are good at dealing with random strings, especially of fixed length, humans prefer meaningful ASCII strings. Since humans represent the majority of network administrators and 100% of paying customers, their preferences were accommodated in USM design. However, the preferences of the computers that would actually run the code and do the work, weren't forgotten either.

If you take a human-readable ASCII string (what a typical user would choose as his password) and run a cryptographically strong hash function over it, you will get a pseudorandom sequence of bytes (usually of a fixed-length), just what a computer would love to use as a key. The USM password to key algorithm concatenates the password with itself many times until it forms a 1 Megabyte string before the hash function is computed. This slows down a possible dictionary attack, when an adversary puts a whole dictionary in his computer and runs the hash-function over each word from it in turn, to see whether the resulting key will match the cryptographic lock he's trying to pick.

Key localization:
In order to allow a human user to have one password, and at the same time to allow the remote engines to have their secrets cryptographically independent from each other, a key localization is employed (described in [1]). In short, it converts a user password or a non-localized key into a key, unique to a specific remote engine, using the secret non-localized key and the remote engine's publicly known identifier as the input to a cryptographically strong one-way function.

As a result, if an SNMP engine is compromised, only communications between this one engine and the users whose localized keys were stored in that engine, are compromised. All the communications between those users and other engines are still secure.

Key update:
It must be possible to distribute new sets of keys and update values of existing keys over the wire without encryption of the messages. The key update protocol is crafted in such a way that it is infeasible to determine the old key from the new key and the key update messages. This is very important, because otherwise the master key used to create new users could be compromised.

Key update uses a cryptographically strong one-way function to: (a) put an impenetrable barrier between the old and the new keys, preventing backwards movement; and (b) make the relations between the bits of the old and the new key intractable both ways (forward and backward), forcing the cryptanalyst to use only the traffic protected by a given key in the attempts to determine that key.

There is, however, a danger if a key for an engine ever falls into the hands of an adversary. From that point on, it is possible for the adversary to "drill forward." The adversary will know the current communication key for as long as the adversary monitors *all* the traffic to that engine and picks all the key update requests. In order to make "drilling forward" more difficult, it is strongly recommended that key update requests always be protected by encryption, if at all possible.

The Textual Convention *KeyChange* defines the behavior of the objects responsible for key updates.

It is assumed, that there are secrets already available on the SNMP engine whose keys we are updating. Normally, there would be a "user template" - an entry in the user table that is used for only one thing - providing cryptographic material (initial keys) and information (initial cryptographic algorithms) for creating new users. Since it was chosen not to employ Diffie-Hellman key negotiation, and encryption is optional, it is not feasible to securely deliver a secret remotely, unless the remote end already shares some secrets with us. Thus, whether the secrets were distributed off-line, or were taken from the preconfigured entries in the user table, by the time we do a key update there is always an "old" secret that we replace (modify).

Once you have your old secret and know what you want to replace it with, the steps in the TC *KeyChange* key update algorithm are:

1. Generate a random value (true randomness is preferred).
2. Compute a temporary value using the old secret and the previously generated random value as input to a cryptographically strong hash function.
3. XOR the result with the desired value of the secret (the new secret) - this is the "delta."
4. Send both the random value and the "delta" to the engine, whose secret you wish to change.
5. The receiving engine reverses the above process, and computes the new secret based on the old secret, the random value provided by the sender, and the "delta" provided by the sender.

However, it is not enough to just be able to update the keys remotely. Both the user himself and the network administrator must be able to change the keys for the given user. This is not as easy as it may seem, due to the intricacies of access control. Without making access control unmanageably complicated, it was deemed infeasible to configure it so that every user would be able to change his and only his keys, but the network administrator would be able to change anybody's keys.

Thus, there are two key update objects per each key in the user entry. One, *usmUserAuthKeyChange*, is for authentication key updates by the network administrator, and would not be within the accessible part of the MIB tree of anybody else.

The other one, *usmUserOwnAuthKeyChange*, differs from the previous object in one important detail: in addition to implementing the algorithm defined by *KeyChange* TC, before permitting the operation it checks whether the requester of the operation has the same *userName* as the "owner" of the row in which this object resides. This means, that even though a user may be granted write access to this object, he will be able to modify only his own key. Precisely as designed, and without increasing the complexity of access control.

Thus, the network administrator can configure the initial keys for a user, while the user can do subsequent key changes by himself. Of course, coordination between the users and the network administrator is required - it is easy to see that if the administrator starts changing people's keys without letting them know, nothing good would come of it.

Authentication

The purpose of USM authentication is to tell with certainty what user the message is from, what engine it is for, whether or not it was damaged en-route, and with a lesser degree of assurance, whether this message is fresh and not a replay. The authentication protocol is determined by the *usmUserAuthProtocol* object in the user table entry. Two authentication protocols are defined for SNMPv3 USM: HMAC-MD5-96, based on MD5 (RFC 1321), and HMAC-SHA-96, based on SHA-1 [2].

Data integrity and authenticity:
In order to provide integrity and authenticity, one generates a special cryptographic "fingerprint" of the message one wishes to protect, and sends that fingerprint along with the message. We call it a "fingerprint" because it is as unique for the message, as fingerprints are unique for human beings. Because of good cryptographic techniques, such fingerprints are impossible (actually infeasible) to forge. In cryptography, such a fingerprint is named a Message Authentication Code (MAC). There are several methods of obtaining (computing) a MAC.

In SNMPv3 USM, both data integrity and authenticity are ensured by using a cryptographically strong hash-function in HMAC mode (RFC 2104). HMAC is the "top of the line" method of applying keyed hash functions for authentication purposes.
In brief, the steps for the HMAC mode are:

1. Derive two temporary keys K1 and K2 from the localized user key K;
2. Compute T = Hash(K2 | Msg);
3. Compute M = Hash(K1 | T);
4. The first 96 bits of M constitute the Message Authentication Code (MAC).

HMAC is new for SNMPv3. Every compliant SNMPv3 engine *must* implement HMAC-MD5-96, and it *should* implement HMAC-SHA-96. Note, that when HMAC-MD5-96 is used, the key for it should be generated using MD5, and when HMAC-SHA-96 is used, the key should be generated using SHA-1.

Timeliness

This section deals with detecting old and replayed messages. Since the amount of material is large, and the concepts are fundamental for SNMPv3, we "elevate" the discussion of timeliness-related issues to a separate section. So how do we ensure that messages we receive are not "stale" (old) or replayed?

In general, synchronized clocks are used. Every message carries a timestamp, which can be examined upon receipt, and if the difference between the timestamp and the clock value of the receiving engine is too large, the message is rejected as old.

However, synchronizing the clocks can be a problem. In addition to that, some SNMP entities may have no battery-powered clocks (i.e. after reboot or power failure their clocks start from zero). But if it is difficult for one engine to keep track of another engine's clock, having each of them keeping track of the other is simply intolerable.

Authoritative clock:
In every secure SNMPv3 communication between the two engines, the authoritative engine has the "clock" that is believed to be "correct" and it is the responsibility of the non-authoritative engine to find out the value of that "clock" and keep track of it.

The authoritative engine maintains two *timeliness values*: *engineBoots* (how many times this SNMP engine was rebooted) and *engineTime* (how many seconds passed since the last reboot). These are not the same as "real" clock, but the similarities are obvious.

Both timeliness values are present in the USM message header and play an important role in replay detection.

The non-authoritative SNMP engine must figure out the current timeliness values of the authoritative engine it communicates with (possibly by sending an insecure GET, and verifying the obtained values with an authenticated request), and keep track of them. By providing automatic clock synchronization (see below), SNMPv3 USM reduces the overhead of explicit clock retrievals to a minimum.

Window of opportunity:
Since we realize that some time has to pass between the moments when the message was encoded and the moment we decode it, we have to have some window of validity - that is, even though the time value from the message is somewhat behind what we expect, the message is still assumed to be good, despite the delays in sending, transit and receiving. This window is not variable or configurable. Thus, USM allows each message a 150-seconds-wide window of opportunity. Any message whose "encoding time" is outside that window is deemed unauthentic. The steps in the verification algorithm are:

1. If the *engineBoots* of the received message is greater than our notion of it, we accept the message and update our notions of both *engineBoots* and *engineTime* of that SNMP engine.
2. If the *engineBoots* of the received message is less than our notion of it, the message obviously is "stale" and is discarded.
3. If the *engineBoots* is equal to our notion of it, we compare the *engineTime* of the message with our notion of *engineTime* of that engine, and depending on whether it differs by less than 150 seconds or not we either accept the message, or discard it.

Note that this allows a message to be replayed within 150 seconds of its generation time. But since this situation is no different from a "normal" packet duplication that occurs in normal network operation, network management applications must be able to cope with it. To assist with sensitive operations (like modifying something, where performing the operation immediately again can cause damage), *TestAndIncr* objects exist in SNMPv3 and it is assumed that a responsible management application will not attempt to modify anything without taking reasonable precautions, such as including one or more *TestAndIncr* objects in the SET operation. On the other hand, it is obvious that a repeated retrieval operation does not cause harm.

Clock synchronization:
We have two communicating engines and one has the authoritative clock. The other one has to (1) learn the value of the authoritative clock, (2) maintain its notion of the clock and (3) periodically verify that its notion of the clock is not too far off. In addition to that, the non-authoritative engine must record the last received time value from the authoritative engine. This saved value is called *latestReceivedEngineTime*.

Initially, the non-authoritative engine has to explicitly retrieve the time value from the authoritative engine. From that point on, clock synchronization is automatic, as long as the authoritative engine's clock does not fall behind the non-authoritative engine's clock for more than 150 seconds since the last secure communication.

Each secure message carries a time value (when it was encoded to be sent). Upon arrival, this time value is compared with the clock. If the difference is within 150 seconds, the message is timely.

If a secure message is within the time window and comes from the engine with the authoritative clock, and the clock value is greater than the *latestReceivedEngineTime* for that engine, then the notion of the time (*engineTime*) for that engine is updated. If the received *engineBoots* value from the authoritative engine is greater than the saved value, then the value is updated for all three objects: *engineBoots*, *engineTime*, *latestReceivedEngineTime*.

Replays:
Even though the following information belongs elsewhere, to be fair to the reader, we will summarize here how the "recent" replays (within 150 seconds) are dealt with.

Every message has a message identifier - an integer field *msgID*. This field should be different for every request, and it is expected that at least for a 150 second interval this will hold true for any implementation. A request originator must match the responses it receives with its outstanding requests using *msgID*, and it is clear that there cannot be two responses with the same *msgID* within a 150 second period. This is how the originator detects the replays.

A receiver, on the other hand, cannot utilize *msgID* for this purpose. The only downside of receiving reasonably "fresh" but duplicated (replayed) retrieval requests is time spent on processing them - they can cause no other harm. Requests to modify the information are more dangerous. They are dealt with adequately by the sender including a *TestAndIncr* object in a SET request. Thus, in a replayed message the object's value will not match the expected one and the operation will fail. This is described in the Message Processing Model.

Privacy

Like the previous SNMP versions, privacy is optional. Unlike SNMPv2p, only the scoped PDU is protected by a privacy blanket.

If privacy is supported by a given engine, it must support the DES [3,4] encryption algorithm. The Encryption algorithm is determined by the *usmUserPrivProtocol* object in the user table entry.

Previous versions of SNMP had a problem because the DES initialization vector (IV) was not changed from message to message. This has been fixed in the USM. The IV now differs for each encrypted message.

The encryption keys are obtained, localized, kept and updated exactly the same way as those for authentication. The objects *usmUserPrivKeyChange* and *usmUserOwnPrivKeyChange* are responsible for that. Note that if SHA-1 is used to obtain a DES encryption key, the first 128 bits of the SHA-1 output are used and the rest is discarded.

References

[1] Blumenthal, U., Hien, N., Wijnen, B., *Key Derivation for Network Management Applications*, IEEE Network Magazine 11(3), May/June 1997.

[2] National Institute of Standards and Technology (NIST), *Secure Hash Standard*, FIPS Publication 180-1, April 1995.

[3] National Institute of Standards and Technology (NIST), *Data Encryption Standard*, FIPS Publication 46-1, January, 1977; reaffirmed January, 1988.

[4] National Institute of Standards and Technology (NIST), *DES Modes of Operation*, FIPS Publication 81, December, 1980.

SNMP Versions

David T. Perkins, SNMPinfo and Desktalk Systems

Confusion continues with regards to the similarities and differences of the different versions of the SNMP protocol. Also, there are many questions regarding dependency between the SMI and the protocol versions. This article attempts to clarify the confusion and answer the questions.

IETF Terminology

The *Internet Engineering Task Force* (IETF) publishes documents that are called *Requests For Comments* (RFCs). These documents, even though called "requests for comments", are "final" versions of documents that specify standards, operational practices, opinions, humor, etc. for the Internet protocol suite. (Documents that are works in progress and made available for review are called *internet-drafts*.) The subset of the documents that specify standards are given a status of *proposed*, *draft*, *full*, *experimental*, or *historic*. A document meant to specify a standard enters the standards-track as *proposed*, and advances to *draft* before becoming a *full* standard after rigorous review, implementation, deployment, and operational experience. A document specifying a protocol, format, or procedure not yet ready for standardization is given the label of *experimental*. Documents that have been replaced by others, or whose contents are no longer relevant have status of *historical*.

SNMP Protocol Versions

The versions of the SNMP protocol are:

- *SNMPv1* (full): This is the first version of the protocol, and is defined by RFC 1157. This document replaces the earlier versions that were published as RFC 1067 and RFC 1098. Security is based on *community strings*.

- *SNMPsec* (historic): This version of the protocol added strong security to the protocol operations of SNMPv1, and is defined by RFC 1351, RFC 1352, and RFC 1353. Security is based on *parties*. Few, if any, vendors implemented this version of the protocol, which is now largely forgotten.

- *SNMPv2p* (historic): For this version, much work was done to update the SNMPv1 protocol and the SMIv1, and not just security. The result was updated protocol operations, new protocol operations and data types, and party-based security from SNMPsec. This version of the protocol, now called party-based SNMPv2 is defined by RFC 1441, RFC 1445, RFC 1446, RFC 1448, and RFC 1449. (Note this protocol has also been called *SNMPv2 classic*, but that name has been confused with community-based SNMPv2. Thus, the term SNMPv2p is preferred.)

- *SNMPv2c* (experimental): This version of the protocol is called community string-based SNMPv2. It is an update of the protocol operations and data types of SNMPv2p, and uses community-based security from SNMPv1. It is defined by RFC 1901, RFC 1905, and RFC 1906.

- *SNMPv2u* (experimental): This version of the protocol uses the protocol operations and data types of SNMPv2c and security based on *users*. It is defined by RFC 1905, RFC 1906, RFC 1909, and RFC 1910.

- *SNMPv2** (experimental): This version combined the best features of SNMPv2p and SNMPv2u. (It is also called *SNMPv2star*.) The documents defining this version were never published as RFCs. Copies of these unpublished documents can be found at the WEB site owned by SNMP Research (a leading SNMP vendor and proponent of this version).

- *SNMPv3* (proposed): This version of the protocol is a combination of user-based security and the protocol operations and data types from SNMPv2p and support for proxies. The security is based on that found in SNMPv2u and SNMPv2*, and updated after much review. The documents defining this protocol will soon be published as RFCs.

SNMP SMI Versions

The Structure of Management Information (SMI) defines the format for defining managed objects that are accessed via the SNMP protocol, the data types of objects, the format for defining events (called traps in SMIv1 and notifications in SMIv2), and contains a few administrative assignments. There are currently two versions of the SMI, which are:

- *SMIv1* (full): This version is defined by RFC 1155, RFC 1212, and RFC 1215. An earlier version defined by RFC 1065 is historic. The current version is also called the *concise* format, since the earlier version was quite verbose.

- *SMIv2* (draft): This version is defined by RFC 1902, RFC 1903, and RFC 1904. An earlier version is defined by RFC 1442, RFC 1443, and RFC 1444 and is historic. The earlier version, which has no widely recognized name, defined a few data types which are no longer supported in the current version of the SMI or SNMPv2 protocol. These data types are *BIT STRING*, *UInteger32*, and *NsapAddress*.

SMIv2 is a backward-compatible update of SMIv1, in all cases except for data type Counter64. That is, it is possible to mechanically create a definition of managed objects in the SMIv1 format from a definition in the SMIv2 format except for objects whose data type is Counter64.

There is no complete mechanical conversion from definitions of managed objects in the SMIv1 format to the SMIv2 format, since the SMIv2 format contains fields for additional information that must be provided by the designer of the definitions. Also, the *ACCESS* clause was changed to *MAX-ACCESS* and its meaning changed, and, thus, the values need to be reviewed when converting from SMIv1 to SMIv2. (You cannot simply use the same values in all cases when you translate object definitions.) Finally, the SMIv2 format contains constructs to define requirement specifications and implementation specifications not found in the SMIv1 format.

By design, the format for the definition of managed objects is independent of the protocol to access them, except for objects with data type of Counter64. That data type does not exist in the SNMPv1 and SNMPsec protocols. A conforming SNMPv1 or SNMPsec entity will generate an ASN.1 parse error when parsing a message containing a Counter64 data type. RFC 2089 defines the behavior of a conforming bilingual (and multilingual) agent that has access to objects with the Counter64 data type.

Version Usage

At this time, only the SNMPv1 protocol has widespread usage. The SNMPv1 protocol is most likely found in every managed device and management platform that supports SNMP. The SNMPsec protocol never saw commercial availability. The SNMPv2p protocol has seen limited commercial availability. Only one of the leading device vendors has made available agents supporting SNMPv2p. All indicators point to no new SNMPv2p offerings and current offerings being replaced by SNMPv2c or SNMPv3. The SNMPv2u and SNMPv2* protocols saw no significant commercial offerings. Support for SNMPv2c in commercial products has been limited, but has been building in 1997. Now that SNMPv3 has been approved to enter the standards-track and for publication, some vendors may not offer SNMPv2c and instead, skip to SNMPv3.

At this time, there is widespread use and support of both versions of the SMI. This is due in part to the policy in the IETF that new versions of RFCs must specify MIB modules in the SMIv2 format. Many commercial products that process MIB modules support both formats.

University Comment

Aiko Pras, University of Twente
Jürgen Schönwälder, TU Braunschweig

At this place readers familiar with *The Simple Times* might have expected to see the Industry Comment. This featured column was written by Marshall Rose, the creator of *The Simple Times* and the driving force behind all 16 issues of the past.

The Simple Times started back in spring 1992. The various issues of *The Simple Times* not only presented interesting articles, but also mixtures of thoughtful summaries, answers to frequently asked questions, excellent inside information and sometimes thought-provoking discussions about future directions. Since its appearance, *The Simple Times* has been very popular within the SNMP community and old issues are still being downloaded from the SimpleTimes Web server by people who want to better understand why things are the way they are. Nowadays, *The Simple Times* has nearly 5000 registered readers.

Marshall Rose left the network management area about a year ago and his farewell seemed to ring the death-knell of the *The Simple Times*. However, given the immense amount of SNMP related work still going on within the IETF, there was a need to re-active *The Simple Times* in order to document some of the evolution and to have a place where people can publish their thoughts on the future of Internet network management.

After some talks with Marshall and other people who had previously contributed to *The Simple Times*, it was decided to move the newsletter and the associated Web server to a new home. The Web server is now maintained by the University of Twente by the same people who maintain the SimpleWeb server, and the editing work is done as a joint project between the Technical University of Braunschweig (Germany) and the University of Twente (the Netherlands). We would like to take this opportunity to thank Marshall for all his help and the time he has spend to make the previous 16 issues of *The Simple Times* a reality.

The issue you are reading right now is the result of this transition and we plan to continue *The Simple Times* as the same newsletter it has been in the past. This means that we continue to rely on help and contributions from the SNMP community. As Marshall wrote in the first issue:

Our job is simply to make the trains run on time. When you like the contents, thank the other volunteers. If an issue comes out late, you know who to blame.

Standards Summary

Please consult the latest version of *Internet Official Protocol Standards*. As of this writing, the latest version is RFC 2200.

SNMPv1 Framework

Full Standards:

- RFC 1155 - Structure of Management Information (SMI);
- RFC 1157 - Simple Network Management Protocol (SNMP); and,
- RFC 1212 - Concise MIB definitions.

Proposed Standards:

- RFC 1418 - SNMP over OSI;
- RFC 1419 - SNMP over AppleTalk; and,
- RFC 1420 - SNMP over IPX.

Informational:

- RFC 1215 - A convention for defining traps for use with the SNMP.

SNMPv2 Framework

Draft Standards:

- RFC 1902 - SMI for SNMPv2;
- RFC 1903 - Textual Conventions for SNMPv2;
- RFC 1904 - Conformance Statements for SNMPv2;
- RFC 1905 - Protocol Operations for SNMPv2;
- RFC 1906 - Transport Mappings for SNMPv2;
- RFC 1907 - MIB for SNMPv2; and,
- RFC 1908 - Coexistence between SNMPv1 and SNMPv2.

Experimental:

- RFC 1901 - Introduction to Community-based SNMPv2;
- RFC 1909 - An Administrative Infrastructure for SNMPv2; and,
- RFC 1910 - User-based Security Model for SNMPv2.

MIB Modules

Full Standards:

- RFC 1213 - Management Information Base (MIB-II); and,
- RFC 1643 - Ether-Like Interface Type (SNMPv1).

Draft Standards:

- RFC 1493 - Bridge MIB;
- RFC 1559 - DECnet phase IV MIB;
- RFC 1657 - BGP version 4 MIB;
- RFC 1658 - Character Device MIB;
- RFC 1659 - RS-232 Interface Type MIB;
- RFC 1660 - Parallel Printer Interface Type MIB;
- RFC 1694 - SMDS Interface Protocol (SIP) Interface Type MIB;
- RFC 1724 - RIP version 2 MIB;
- RFC 1748 - IEEE 802.5 Token Ring Interface Type MIB;
- RFC 1757 - Remote Network Monitoring MIB;
- RFC 1850 - OSPF version 2 MIB; and,
- RFC 2115 - Frame Relay DTE Interface Type MIB.

Proposed Standards:

- RFC 1285 - FDDI Interface Type (SMT 6.2) MIB;
- RFC 1381 - X.25 LAPB MIB;
- RFC 1382 - X.25 PLP MIB;
- RFC 1406 - DS1/E1 Interface Type MIB;
- RFC 1407 - DS3/E3 Interface Type MIB;
- RFC 1414 - Identification MIB;
- RFC 1461 - Multiprotocol Interconnect over X.25 MIB;
- RFC 1471 - PPP Link Control Protocol (LCP) MIB;
- RFC 1472 - PPP Security Protocols MIB;

- RFC 1473 - PPP IP Network Control Protocol MIB;
- RFC 1474 - PPP Bridge Network Control Protocol MIB;
- RFC 1512 - FDDI Interface Type (SMT 7.3) MIB;
- RFC 1513 - Token Ring Extensions to RMON MIB;
- RFC 1514 - Host Resources MIB;
- RFC 1525 - Source Routing Bridge MIB;
- RFC 1565 - Network Services Monitoring MIB;
- RFC 1566 - Mail Monitoring MIB;
- RFC 1567 - X.500 Directory Monitoring MIB;
- RFC 1573 - Evolution of the Interfaces Group of MIB-II;
- RFC 1595 - SONET/SDH Interface Type MIB;
- RFC 1604 - Frame Relay Service MIB;
- RFC 1611 - DNS Server MIB;
- RFC 1612 - DNS Resolver MIB;
- RFC 1628 - Uninterruptible Power Supply MIB;
- RFC 1650 - Ether-Like Interface Type (SNMPv2);
- RFC 1666 - SNA NAU MIB;
- RFC 1695 - ATM MIB;
- RFC 1696 - Modem MIB;
- RFC 1697 - Relational Database Management System MIB;
- RFC 1742 - AppleTalk MIB;
- RFC 1747 - SNA DLC MIB;
- RFC 1749 - 802.5 Station Source Routing MIB;
- RFC 1759 - Printer MIB;
- RFC 2006 - Mobile IP MIB;
- RFC 2011 - SNMPv2 IP MIB;
- RFC 2012 - SNMPv2 TCP MIB;
- RFC 2013 - SNMPv2 UDP MIB;
- RFC 2020 - IEEE 802.12 Interfaces MIB;
- RFC 2021 - RMON-2 MIB;
- RFC 2024 - Data Link Switching MIB;
- RFC 2037 - Entity MIB;
- RFC 2051 - APPC MIB;
- RFC 2074 - RMON Protocol Identifier;
- RFC 2096 - IP Forwarding Table MIB;
- RFC 2108 - IEEE 802.3 Repeater MIB;
- RFC 2127 - ISDN MIB;
- RFC 2128 - Dial Control MIB;
- RFC 2155 - APPN MIB;
- RFC 2206 - Resource Reservation Protocol MIB;
- RFC 2213 - Integrated Services MIB;
- RFC 2214 - Integrated Services Guaranteed Service Extensions MIB;
- RFC 2232 - DLUR MIB;
- RFC 2233 - Interfaces Group MIB;
- RFC 2238 - High Performance Routing MIB; and,
- RFC 2239 - IEEE 802.3 Medium Attachment Unit (MAU) MIB.

Related Documents

Informational:

- RFC 1270 - SNMP communication services;
- RFC 1321 - MD5 message-digest algorithm;
- RFC 1470 - A network management tool catalog;
- RFC 2039 - Applicablity of Standard MIBs to WWW Server Management; and,
- RFC 2089 - Mapping SNMPv2 onto SNMPv1 within a bi-lingual SNMP agent.

Experimental:

- RFC 1187 - Bulk table retrieval with the SNMP;
- RFC 1224 - Techniques for managing asynchronously generated alerts;
- RFC 1238 - CLNS MIB; and,
- RFC 1592 - SNMP Distributed Program Interface (SNMP-DPI);
- RFC 1792 - TCP/IPX Connection MIB Specification; and,
- RFC 2064 - Traffic Flow Measurement: Meter MIB.

Internet Resources

- IETF Home Page
  `http://www.ietf.org/`
- The Simple Web
  `http://wwwsnmp.cs.utwente.nl/`
- The Simple Times
  `http://www.simple-times.org/`
- The SNMPv3 Page
  `http://www.ibr.cs.tu-bs.de/projects/snmpv3/`

Calendar and Announcements

IETF Meetings:

- 40th Meeting of the IETF
  December 8-12, 1997, Washington, DC, USA
- 41th Meeting of the IETF
  March 30-April 3, 1998, Los Angeles, CA, USA
- 42th Meeting of the IETF
  August 23-28, 1998, Chicago, IL, USA

Conferences and Workshops:

- Network Operations & Management Symposium '98
  February 15-20, 1998, New Orleans, LA, USA
- IEEE Workshop on Systems Management '98
  April 22-24, 1998, Newport, Rhode Island, USA
- Enterprise Management Summit '98
  August 3-7, 1998, Santa Clara, CA, USA
- Distributed Systems Operations & Management '98
  October 26-28, 1998, Delaware, USA
- Integrated Network Management '99
  May 10-14, 1999, Boston, MA, USA

Exhibitions and Trade Shows:

- NetWorld + Interop Singapore
  March 30-April 3, 1998, Singapore
- NetWorld + Interop Las Vegas
  May 4-8, 1998, Las Vegas, USA
- NetWorld + Interop Tokio
  June 1-5, 1998, Tokio, Japan
- NetWorld + Interop London
  October 13-15, 1998, London, UK
- NetWorld + Interop Atlanta
  October 19-23, 1998, Atlanta, USA
- NetWorld + Interop Paris
  November 5-7, 1998, Paris, France
- NetWorld + Interop Sydney
  November 24-28, 1998, Sydney, Australia

Publication Information

Editors	
Jürgen Schönwälder	TU Braunschweig
Aiko Pras	University Twente
Contact Information	
E-mail	`st-editorial@simple-times.org`
ISSN	1060–6068

Submissions

The Simple Times solicits high-quality articles of technology and comment. Technical articles are refereed to ensure that the content is marketing-free. By definition, commentaries reflect opinion and, as such, are reviewed only to the extent required to ensure commonly-accepted publication norms.

The Simple Times also solicits terse announcements of products and services, publications, and events. These contributions are reviewed only to the extent required to ensure commonly-accepted publication norms.

Submissions are accepted only via electronic mail, and must be formatted in HTML version 1.0. Each submission must include the author's full name, title, affiliation, postal and electronic mail addresses, telephone, and fax numbers. Note that by initiating this process, the submitting party agrees to place the contribution into the public domain.

Subscriptions

The Simple Times is available in HTML and PostScript. New issues are announced via an electronic mailing list. Send electronic mail to

`st-request@simple-times.org`

with a `Subject:` line of

`help`

if you want to subscribe to this list. Back issues are available via *The Simple Times* Web server:

`http://www.simple-times.org/`