Network Working Group
Request for Comments: 2572
Obsoletes: 2272
Category: Standards Track

J. Case
SNMP Research Inc.
D. Harrington
Cabletron Systems, Inc.
R. Presuhn
BMC Software, Inc.
B. Wijnen
IBM T. J. Watson Research

April 1999

Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)

Status of this Memo

Copyright Notice

Abstract

This document describes the Message Processing and Dispatching for SNMP messages within the SNMP architecture [RFC2571]. It defines the procedures for dispatching potentially multiple versions of SNMP messages to the proper SNMP Message Processing Models, and for dispatching PDUs to SNMP applications. This document also describes one Message Processing Model - the SNMPv3 Message Processing Model.

Table of Contents

1. Introduction

The Architecture for describing Internet Management Frameworks [RFC2571] describes that an SNMP engine is composed of:

1) a Dispatcher
2) a Message Processing Subsystem,
3) a Security Subsystem, and
4) an Access Control Subsystem.

Applications make use of the services of these subsystems.

It is important to understand the SNMP architecture and its terminology to understand where the Message Processing Subsystem and Dispatcher described in this document fit into the architecture and interact with other subsystems within the architecture. The reader is expected to have read and understood the description of the SNMP architecture, defined in [RFC2571].

The Dispatcher in the SNMP engine sends and receives SNMP messages. It also dispatches SNMP PDUs to SNMP applications. When an SNMP message needs to be prepared or when data needs to be extracted from an SNMP message, the Dispatcher delegates these tasks to a message version-specific Message Processing Model within the Message Processing Subsystem.

A Message Processing Model is responsible for processing a SNMP version-specific message and for coordinating the interaction with the Security Subsystem to ensure proper security is applied to the SNMP message being handled.

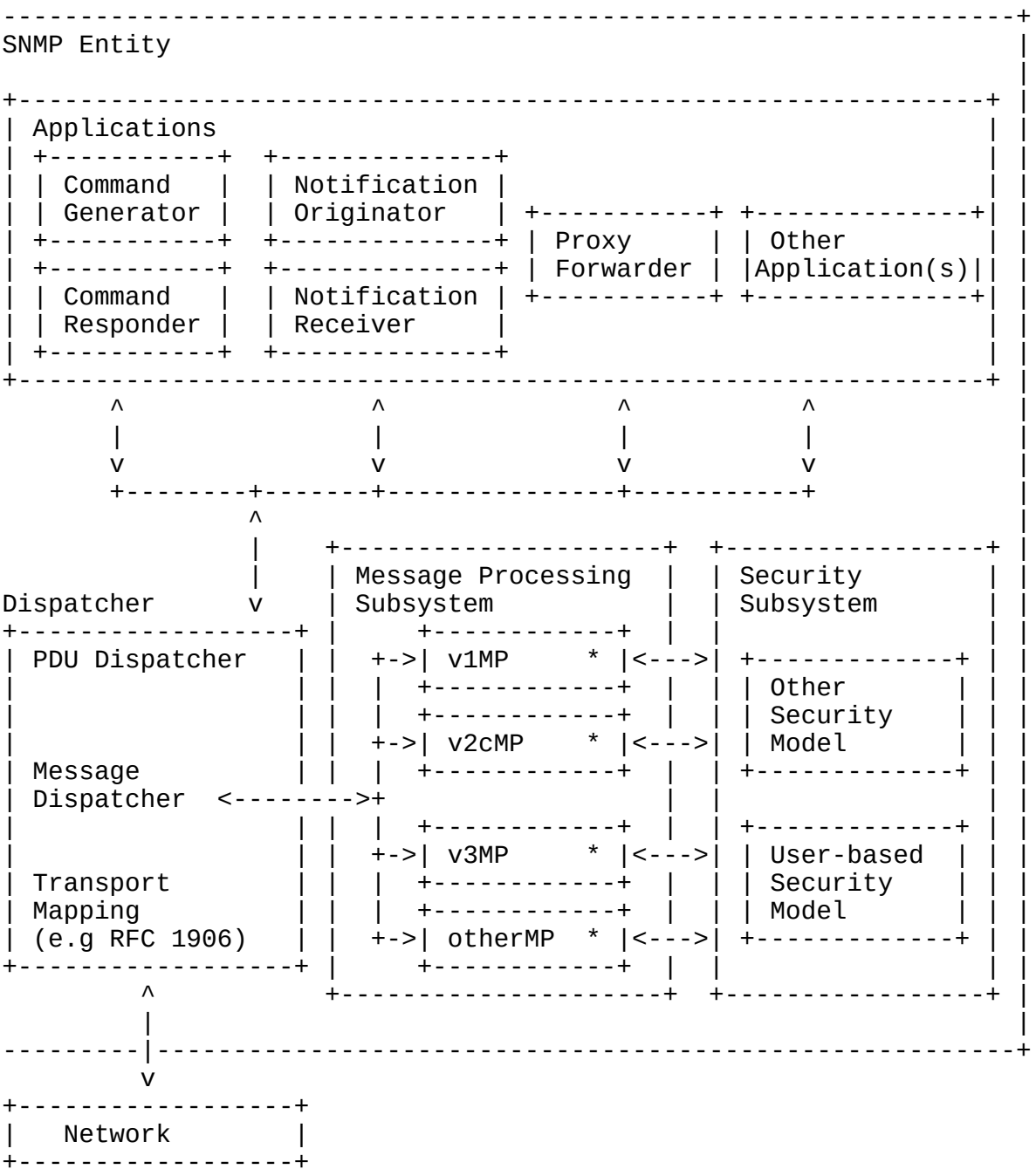
Interactions between the Dispatcher, the Message Processing Subsystem, and applications are modeled using abstract data elements and abstract service interface primitives defined by the SNMP architecture.

Similarly, interactions between the Message Processing Subsystem and the Security Subsystem are modeled using abstract data elements and abstract service interface primitives as defined by the SNMP architecture.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

2. Overview

The following illustration depicts the Message Processing in relation to SNMP applications, the Security Subsystem and Transport Mappings.



2.1. The Dispatcher.

The Dispatcher is a key piece of an SNMP engine. There is only one in an SNMP engine, and its job is to dispatch tasks to the multiple version-specific Message Processing Models, and to dispatch PDUs to various applications.

For outgoing messages, an application provides a PDU to be sent, plus the data needed to prepare and send the message, and the application specifies which version-specific Message Processing Model will be used to prepare the message with the desired security processing. Once the message is prepared, the Dispatcher sends the message.

For incoming messages, the Dispatcher determines the SNMP version of the incoming message and passes the message to the version-specific Message Processing Model to extract the components of the message and to coordinate the processing of security services for the message. After version-specific processing, the PDU Dispatcher determines which application, if any, should receive the PDU for processing and forwards it accordingly.

The Dispatcher, while sending and receiving SNMP messages, collects statistics about SNMP messages and the behavior of the SNMP engine in managed objects to make them accessible to remote SNMP entities. This document defines these managed objects, the MIB module which contains them, and how these managed objects might be used to provide useful management.

2.2. Message Processing Subsystem

The SNMP Message Processing Subsystem is the part of an SNMP engine which interacts with the Dispatcher to handle the version-specific SNMP messages. It contains one or more Message Processing Models.

This document describes one Message Processing Model, the SNMPv3 Message Processing Model, in Section 6. The SNMPv3 Message Processing Model is defined in a separate section to show that multiple (independent) Message Processing Models can exist at the same time and that such Models can be described in different documents. The SNMPv3 Message Processing Model can be replaced or supplemented with other Message Processing Models in the future. Two Message Processing Models which are expected to be developed in the future are the SNMPv1 message format [RFC1157] and the SNMPv2c message format [RFC1901]. Others may be developed as needed.

3. Elements of Message Processing and Dispatching

See [RFC2571] for the definitions of
- contextEngineID
- contextName
- scopedPDU
- maxSizeResponseScopedPDU
- securityModel
- securityName
- securityLevel
- messageProcessingModel

For incoming messages, a version-specific message processing module provides these values to the Dispatcher. For outgoing messages, an application provides these values to the Dispatcher.

For some version-specific processing, the values may be extracted from received messages; for other versions, the values may be determined by algorithm, or by an implementation-defined mechanism. The mechanism by which the value is determined is irrelevant to the Dispatcher.

The following additional or expanded definitions are for use within the Dispatcher.

3.1. messageProcessingModel

The value of messageProcessingModel identifies a Message Processing Model. A Message Processing Model describes the version-specific procedures for extracting data from messages, generating messages, calling upon a securityModel to apply its security services to messages, for converting data from a version-specific message format into a generic format usable by the Dispatcher, and for converting data from Dispatcher format into a version-specific message format.

3.2. pduVersion

The value of pduVersion represents a specific version of protocol operation and its associated PDU formats, such as SNMPv1 or SNMPv2 [RFC1905]. The values of pduVersion are specific to the version of the PDU contained in a message, and the PDUs processed by applications. The Dispatcher does not use the value of pduVersion directly.

An application specifies the pduVersion when it requests the PDU Dispatcher to send a PDU to another SNMP engine. The Dispatcher passes the pduVersion to a Message Processing Model, so it knows how to handle the PDU properly.

For incoming messages, pduVersion is provided to the Dispatcher by a version-specific Message Processing module. The PDU Dispatcher passes the pduVersion to the application so it knows how to handle the PDU properly. For example, a command responder application needs to know whether to use [RFC1905] elements of procedure and syntax instead of those specified for SNMPv1.

3.3. pduType

A value of pduType represents a specific type of protocol operation. The values of pduType are specific to the version of the PDU contained in a message.

Applications register to support particular pduTypes for particular contextEngineIDs.

For incoming messages, pduType is provided to the Dispatcher by a version-specific Message Processing module. It is subsequently used to dispatch the PDU to the application which registered for the pduType for the contextEngineID of the associated scopedPDU.

3.4. sendPduHandle

This handle is generated for coordinating the processing of requests and responses between the SNMP engine and an application. The handle must be unique across all version-specific Message Processing Models, and is of local significance only.

4. Dispatcher Elements of Procedure

This section describes the procedures followed by the Dispatcher when generating and processing SNMP messages.

4.1. Sending an SNMP Message to the Network

This section describes the procedure followed by an SNMP engine whenever it sends an SNMP message.

4.1.1. Sending a Request or Notification

The following procedures are followed by the Dispatcher when an application wants to send an SNMP PDU to another (remote) application, i.e., to initiate a communication by originating a message, such as one containing a request or a trap.

1) The application requests this using the abstract service primitive:

```
 statusInformation =              -- sendPduHandle if success
                                  -- errorIndication if failure
   sendPdu(
   IN   transportDomain           -- transport domain to be used
   IN   transportAddress          -- destination network address
   IN   messageProcessingModel    -- typically, SNMP version
   IN   securityModel             -- Security Model to use
   IN   securityName              -- on behalf of this principal
   IN   securityLevel             -- Level of Security requested
   IN   contextEngineID           -- data from/at this entity
   IN   contextName               -- data from/in this context
   IN   pduVersion                -- the version of the PDU
   IN   PDU                       -- SNMP Protocol Data Unit
   IN   expectResponse            -- TRUE or FALSE
        )
```

2) If the messageProcessingModel value does not represent a Message Processing Model known to the Dispatcher, then an errorIndication (implementation-dependent) is returned to the calling application. No further processing is performed.

3) The Dispatcher generates a sendPduHandle to coordinate subsequent processing.

4) The Message Dispatcher sends the request to the version-specific Message Processing module identified by messageProcessingModel using the abstract service primitive:

```
statusInformation =              - success or error indication
  prepareOutgoingMessage(
  IN   transportDomain           -- as specified by application
  IN   transportAddress          -- as specified by application
  IN   messageProcessingModel    -- as specified by application
  IN   securityModel             -- as specified by application
  IN   securityName              -- as specified by application
  IN   securityLevel             -- as specified by application
  IN   contextEngineID           -- as specified by application
  IN   contextName               -- as specified by application
  IN   pduVersion                -- as specified by application
  IN   PDU                       -- as specified by application
  IN   expectResponse            -- as specified by application
  IN   sendPduHandle             -- as determined in step 3.
  OUT  destTransportDomain       -- destination transport domain
  OUT  destTransportAddress      -- destination transport address
  OUT  outgoingMessage           -- the message to send
  OUT  outgoingMessageLength     -- the message length
       )
```

5) If the statusInformation indicates an error, the errorIndication is returned to the calling application. No further processing is performed.

6) If the statusInformation indicates success, the sendPduHandle is returned to the application, and the outgoingMessage is sent via the transport specified by the transportDomain to the address specified by the transportAddress.

Outgoing Message Processing is complete.

4.1.2. Sending a Response to the Network

The following procedure is followed when an application wants to return a response back to the originator of an SNMP Request.

1) An application can request this using the abstract service primitive:

```
result =
returnResponsePdu(
 IN   messageProcessingModel   -- typically, SNMP version
 IN   securityModel            -- Security Model in use
 IN   securityName             -- on behalf of this principal
 IN   securityLevel            -- same as on incoming request
 IN   contextEngineID          -- data from/at this SNMP entity
 IN   contextName              -- data from/in this context
 IN   pduVersion               -- the version of the PDU
 IN   PDU                      -- SNMP Protocol Data Unit
 IN   maxSizeResponseScopedPDU -- maximum size of Response PDU
 IN   stateReference           -- reference to state information
                               -- as presented with the request
 IN   statusInformation        -- success or errorIndication
 )                             -- (error counter OID and value
                               -- when errorIndication)
```

2) The Message Dispatcher sends the request to the appropriate Message Processing Model indicated by the received value of messageProcessingModel using the abstract service primitive:

```
result =                        -- SUCCESS or errorIndication
 prepareResponseMessage(
 IN   messageProcessingModel   -- specified by application
 IN   securityModel            -- specified by application
 IN   securityName             -- specified by application
 IN   securityLevel            -- specified by application
 IN   contextEngineID          -- specified by application
 IN   contextName              -- specified by application
 IN   pduVersion               -- specified by application
 IN   PDU                      -- specified by application
 IN   maxSizeResponseScopedPDU -- specified by application
 IN   stateReference           -- specified by application
 IN   statusInformation        -- specified by application
 OUT  destTransportDomain      -- destination transport domain
 OUT  destTransportAddress     -- destination transport address
 OUT  outgoingMessage          -- the message to send
 OUT  outgoingMessageLength    -- the message length
      )
```

3) If the result is an errorIndication, the errorIndication is returned to the calling application. No further processing is performed.

4) If the result is success, the outgoingMessage is sent over the transport specified by the transportDomain to the address specified by the transportAddress.

Message Processing is complete.

4.2. Receiving an SNMP Message from the Network

This section describes the procedure followed by an SNMP engine whenever it receives an SNMP message.

Please note, that for the sake of clarity and to prevent the text from being even longer and more complicated, some details were omitted from the steps below. In particular, The elements of procedure do not always explicitly indicate when state information needs to be released. The general rule is that if state information is available when a message is to be "discarded without further processing", then the state information must also be released at that same time.

4.2.1. Message Dispatching of received SNMP Messages

1) The snmpInPkts counter [RFC1907] is incremented.

2) The version of the SNMP message is determined in an implementation-dependent manner. If the packet cannot be sufficiently parsed to determine the version of the SNMP message, then the snmpInASNParseErrs [RFC1907] counter is incremented, and the message is discarded without further processing. If the version is not supported, then the snmpInBadVersions [RFC1907] counter is incremented, and the message is discarded without further processing.

3) The origin transportDomain and origin transportAddress are determined.

4) The message is passed to the version-specific Message Processing Model which returns the abstract data elements required by the Dispatcher. This is performed using the abstract service primitive:

```
result =                          -- SUCCESS or errorIndication
  prepareDataElements(
  IN   transportDomain            -- origin as determined in step 3.
  IN   transportAddress           -- origin as determined in step 3.
  IN   wholeMsg                   -- as received from the network
  IN   wholeMsgLength             -- as received from the network
  OUT  messageProcessingModel     -- typically, SNMP version
  OUT  securityModel              -- Security Model specified
  OUT  securityName               -- on behalf of this principal
  OUT  securityLevel              -- Level of Security specified
  OUT  contextEngineID            -- data from/at this entity
  OUT  contextName                -- data from/in this context
  OUT  pduVersion                 -- the version of the PDU
  OUT  PDU                        -- SNMP Protocol Data Unit
  OUT  pduType                    -- SNMP PDU type
  OUT  sendPduHandle              -- handle for a matched request
  OUT  maxSizeResponseScopedPDU -- maximum size of Response PDU
  OUT  statusInformation          -- success or errorIndication
                                  -- (error counter OID and value
                                  -- when errorIndication)
  OUT  stateReference             -- reference to state information
                                  -- to be used for a possible
       )                          -- Response
```

5) If the result is a FAILURE errorIndication, the message is discarded without further processing.

6) At this point, the abstract data elements have been prepared and processing continues as described in Section 4.2.2, PDU Dispatching for Incoming Messages.

4.2.2. PDU Dispatching for Incoming Messages

The elements of procedure for the dispatching of PDUs depends on the value of sendPduHandle. If the value of sendPduHandle is <none>, then this is a request or notification and the procedures specified in Section 4.2.2.1 apply. If the value of snmpPduHandle is not <none>, then this is a response and the procedures specified in Section 4.2.2.2 apply.

4.2.2.1. Incoming Requests and Notifications

The following procedures are followed for the dispatching of PDUs when the value of sendPduHandle is <none>, indicating this is a request or notification.

1) The combination of contextEngineID and pduType is used to determine which application has registered for this request or notification.

2) If no application has registered for the combination, then

   a) The snmpUnknownPDUHandlers counter is incremented.

   b) A Response message is generated using the abstract service primitive:

```
result =                            -- SUCCESS or FAILURE
prepareResponseMessage(
IN   messageProcessingModel    -- as provided by MP module
IN   securityModel             -- as provided by MP module
IN   securityName              -- as provided by MP module
IN   securityLevel             -- as provided by MP module
IN   contextEngineID           -- as provided by MP module
IN   contextName               -- as provided by MP module
IN   pduVersion                -- as provided by MP module
IN   PDU                       -- as provided by MP module
IN   maxSizeResponseScopedPDU  -- as provided by MP module
IN   stateReference            -- as provided by MP module
IN   statusInformation         -- errorIndication plus
                               -- snmpUnknownPDUHandlers OID
                               -- value pair.
OUT  destTransportDomain       -- destination transportDomain
OUT  destTransportAddress      -- destination transportAddress
OUT  outgoingMessage           -- the message to send
OUT  outgoingMessageLength     -- its length
)
```

   c) If the result is SUCCESS, then the prepared message is sent to the originator of the request as identified by the transportDomain and transportAddress.

   d) The incoming message is discarded without further processing. Message Processing for this message is complete.

3) The PDU is dispatched to the application, using the abstract service primitive:

```
processPdu(                       -- process Request/Notification
  IN   messageProcessingModel     -- as provided by MP module
  IN   securityModel              -- as provided by MP module
  IN   securityName               -- as provided by MP module
  IN   securityLevel              -- as provided by MP module
  IN   contextEngineID            -- as provided by MP module
  IN   contextName                -- as provided by MP module
  IN   pduVersion                 -- as provided by MP module
  IN   PDU                        -- as provided by MP module
  IN   maxSizeResponseScopedPDU -- as provided by MP module
  IN   stateReference             -- as provided by MP module
                                  -- needed when sending response
       )
```

Message processing for this message is complete.

4.2.2.2. Incoming Responses

The following procedures are followed for the dispatching of PDUs when the value of sendPduHandle is not <none>, indicating this is a response.

1) The value of sendPduHandle is used to determine, in an implementation-defined manner, which application is waiting for a response associated with this sendPduHandle.

2) If no waiting application is found, the message is discarded without further processing, and the stateReference is released. The snmpUnknownPDUHandlers counter is incremented. Message Processing is complete for this message.

3) Any cached information, including stateReference, about the message is discarded.

4) The response is dispatched to the application using the abstract service primitive:

```
        processResponsePdu(              -- process Response PDU
          IN   messageProcessingModel    -- provided by the MP module
          IN   securityModel             -- provided by the MP module
          IN   securityName              -- provided by the MP module
          IN   securityLevel             -- provided by the MP module
          IN   contextEngineID           -- provided by the MP module
          IN   contextName               -- provided by the MP module
          IN   pduVersion                -- provided by the MP module
          IN   PDU                       -- provided by the MP module
          IN   statusInformation         -- provided by the MP module
          IN   sendPduHandle             -- provided by the MP module
               )
```

Message Processing is complete for this message.

4.3. Application Registration for Handling PDU types

Applications that want to process certain PDUs must register with the PDU Dispatcher. Applications specify the combination of contextEngineID and pduType(s) for which they want to take responsibility

1) An application registers according to the abstract interface primitive:

```
statusInformation =           -- success or errorIndication
  registerContextEngineID(
  IN   contextEngineID        -- take responsibility for this one
  IN   pduType                -- the pduType(s) to be registered
       )
```

Note: implementations may provide a means of requesting registration for simultaneous multiple contextEngineID values, e.g., all contextEngineID values, and may also provide means for requesting simultaneous registration for multiple values of pduType.

2) The parameters may be checked for validity; if they are not, then an errorIndication (invalidParameter) is returned to the application.

3) Each combination of contextEngineID and pduType can be registered only once. If another application has already registered for the specified combination, then an errorIndication (alreadyRegistered) is returned to the application.

4) Otherwise, the registration is saved so that SNMP PDUs can be dispatched to this application.

4.4. Application Unregistration for Handling PDU Types

Applications that no longer want to process certain PDUs must unregister with the PDU Dispatcher.

1) An application unregisters using the abstract service primitive:

```
unregisterContextEngineID(
 IN   contextEngineID        -- give up responsibility for this
 IN   pduType                -- the pduType(s) to be unregistered
       )
```

Note: implementations may provide means for requesting unregistration for simultaneous multiple contextEngineID values, e.g., all contextEngineID values, and may also provide means for requesting simultaneous unregistration for multiple values of pduType.

2) If the contextEngineID and pduType combination has been registered, then the registration is deleted.

If no such registration exists, then the request is ignored.

5. Definitions

5.1. Definitions for SNMP Message Processing and Dispatching

```
SNMP-MPD-MIB DEFINITIONS ::= BEGIN

IMPORTS
    MODULE-COMPLIANCE, OBJECT-GROUP         FROM SNMPv2-CONF
    MODULE-IDENTITY, OBJECT-TYPE,
    snmpModules, Counter32                  FROM SNMPv2-SMI;

snmpMPDMIB MODULE-IDENTITY
    LAST-UPDATED "9905041636Z"              -- 4 April 1999
    ORGANIZATION "SNMPv3 Working Group"
    CONTACT-INFO "WG-EMail:   snmpv3@lists.tislabs.com
                  Subscribe:  majordomo@lists.tislabs.com
                              In message body:  subscribe snmpv3

                  Chair:      Russ Mundy
                              TIS Labs at Network Associates
                  postal:     3060 Washington Road
                              Glenwood, MD 21738
                              USA
                  EMail:      mundy@tislabs.com
                  phone:      +1 301-854-6889
```

```
                    Co-editor:  Jeffrey Case
                                SNMP Research, Inc.
                    postal:     3001 Kimberlin Heights Road
                                Knoxville, TN 37920-9716
                                USA
                    EMail:      case@snmp.com
                    phone:      +1 423-573-1434

                    Co-editor   Dave Harrington
                                Cabletron Systems, Inc.
                    postal:     Post Office Box 5005
                                MailStop: Durham
                                35 Industrial Way
                                Rochester, NH 03867-5005
                                USA
                    EMail:      dbh@ctron.com
                    phone:      +1 603-337-7357

                    Co-editor:  Randy Presuhn
                                BMC Software, Inc.
                    postal:     965 Stewart Drive
                                Sunnyvale, CA 94086
                                USA
                    EMail:      randy_presuhn@bmc.com
                    phone:      +1 408-616-3100

                    Co-editor:  Bert Wijnen
                                IBM T. J. Watson Research
                    postal:     Schagen 33
                                3461 GL Linschoten
                                Netherlands
                    EMail:      wijnen@vnet.ibm.com
                    phone:      +31 348-432-794

                   "
    DESCRIPTION  "The MIB for Message Processing and Dispatching"
    REVISION     "9905041636Z"              -- 4 April 1999
    DESCRIPTION  "Updated addresses, published as RFC 2572."
    REVISION     "9709300000Z"              -- 30 September 1997
    DESCRIPTION  "Original version, published as RFC 2272."
    ::= { snmpModules 11 }

-- Administrative assignments ************************************

snmpMPDAdmin           OBJECT IDENTIFIER ::= { snmpMPDMIB 1 }
snmpMPDMIBObjects      OBJECT IDENTIFIER ::= { snmpMPDMIB 2 }
snmpMPDMIBConformance  OBJECT IDENTIFIER ::= { snmpMPDMIB 3 }
```

```
   -- Statistics for SNMP Messages ************************************

   snmpMPDStats           OBJECT IDENTIFIER ::= { snmpMPDMIBObjects 1 }

   snmpUnknownSecurityModels OBJECT-TYPE
       SYNTAX       Counter32
       MAX-ACCESS   read-only
       STATUS       current
       DESCRIPTION "The total number of packets received by the SNMP
                    engine which were dropped because they referenced a
                    securityModel that was not known to or supported by
                    the SNMP engine.
                   "
       ::= { snmpMPDStats 1 }

   snmpInvalidMsgs OBJECT-TYPE
       SYNTAX       Counter32
       MAX-ACCESS   read-only
       STATUS       current
       DESCRIPTION "The total number of packets received by the SNMP
                    engine which were dropped because there were invalid
                    or inconsistent components in the SNMP message.
                   "
       ::= { snmpMPDStats 2 }

   snmpUnknownPDUHandlers OBJECT-TYPE
       SYNTAX       Counter32
       MAX-ACCESS   read-only
       STATUS       current
       DESCRIPTION "The total number of packets received by the SNMP
                    engine which were dropped because the PDU contained
                    in the packet could not be passed to an application
                    responsible for handling the pduType, e.g. no SNMP
                    application had registered for the proper
                    combination of the contextEngineID and the pduType.
                   "
       ::= { snmpMPDStats 3 }

   -- Conformance information *************************************

   snmpMPDMIBCompliances OBJECT IDENTIFIER ::= {snmpMPDMIBConformance 1}
   snmpMPDMIBGroups      OBJECT IDENTIFIER ::= {snmpMPDMIBConformance 2}

   -- Compliance statements

   snmpMPDCompliance MODULE-COMPLIANCE
       STATUS       current
       DESCRIPTION "The compliance statement for SNMP entities which
```

```
                    implement the SNMP-MPD-MIB.
                   "

        MODULE    -- this module
            MANDATORY-GROUPS { snmpMPDGroup }

        ::= { snmpMPDMIBCompliances 1 }

    snmpMPDGroup OBJECT-GROUP
        OBJECTS {
                  snmpUnknownSecurityModels,
                  snmpInvalidMsgs,
                  snmpUnknownPDUHandlers
                }
        STATUS       current
        DESCRIPTION "A collection of objects providing for remote
                     monitoring of the SNMP Message Processing and
                     Dispatching process.
                    "
        ::= { snmpMPDMIBGroups 1 }

    END
```

6. The SNMPv3 Message Format

This section defines the SNMPv3 message format and the corresponding SNMP version 3 Message Processing Model (v3MP).

```
SNMPv3MessageSyntax DEFINITIONS IMPLICIT TAGS ::= BEGIN

    SNMPv3Message ::= SEQUENCE {
        -- identify the layout of the SNMPv3Message
        -- this element is in same position as in SNMPv1
        -- and SNMPv2c, allowing recognition
        -- the value 3 is used for snmpv3
        msgVersion INTEGER ( 0 .. 2147483647 ),
        -- administrative parameters
        msgGlobalData HeaderData,
        -- security model-specific parameters
        -- format defined by Security Model
        msgSecurityParameters OCTET STRING,
        msgData  ScopedPduData
    }

    HeaderData ::= SEQUENCE {
        msgID      INTEGER (0..2147483647),
        msgMaxSize INTEGER (484..2147483647),

        msgFlags   OCTET STRING (SIZE(1)),
                   --  .... ...1   authFlag
                   --  .... ..1.   privFlag
                   --  .... .1..   reportableFlag
                   --              Please observe:
                   --  .... ..00   is OK, means noAuthNoPriv
                   --  .... ..01   is OK, means authNoPriv
                   --  .... ..10   reserved, must NOT be used.
                   --  .... ..11   is OK, means authPriv

        msgSecurityModel INTEGER (1..2147483647)
    }

    ScopedPduData ::= CHOICE {
        plaintext    ScopedPDU,
        encryptedPDU OCTET STRING  -- encrypted scopedPDU value
    }

    ScopedPDU ::= SEQUENCE {
        contextEngineID  OCTET STRING,
        contextName      OCTET STRING,
        data             ANY -- e.g., PDUs as defined in RFC 1905
    }
END
```

6.1. msgVersion

The msgVersion field is set to snmpv3(3) and identifies the message as an SNMP version 3 Message.

6.2. msgID

The msgID is used between two SNMP entities to coordinate request messages and responses, and by the v3MP to coordinate the processing of the message by different subsystem models within the architecture.

Values for msgID SHOULD be generated in a manner that avoids re-use of any outstanding values. Doing so provides protection against some replay attacks. One possible implementation strategy would be to use the low-order bits of snmpEngineBoots [RFC2571] as the high-order portion of the msgID value and a monotonically increasing integer for the low-order portion of msgID.

Note that the request-id in a PDU may be used by SNMP applications to identify the PDU; the msgID is used by the engine to identify the message which carries a PDU. The engine needs to identify the message even if decryption of the PDU (and request-id) fails. No assumption should be made that the value of the msgID and the value of the request-id are equivalent.

The value of the msgID field for a response takes the value of the msgID field from the message to which it is a response. By use of the msgID value, an engine can distinguish the (potentially multiple) outstanding requests, and thereby correlate incoming responses with outstanding requests. In cases where an unreliable datagram service is used, the msgID also provides a simple means of identifying messages duplicated by the network. If a request is retransmitted, a new msgID value SHOULD be used for each retransmission.

6.3. msgMaxSize

The msgMaxSize field of the message conveys the maximum message size supported by the sender of the message, i.e., the maximum message size that the sender can accept when another SNMP engine sends an SNMP message (be it a response or any other message) to the sender of this message on the transport in use for this message.

When an SNMP message is being generated, the msgMaxSize is provided by the SNMP engine which generates the message. At the receiving SNMP engine, the msgMaxSize is used to determine the maximum message size the sender can accommodate.

6.4. msgFlags

The msgFlags field of the message contains several bit fields which control processing of the message.

The reportableFlag is a secondary aid in determining whether a Report PDU must be sent. It is only used in cases where the PDU portion of a message cannot be decoded, due to, for example, an incorrect encryption key. If the PDU can be decoded, the PDU type forms the basis for decisions on sending Report PDUs.

When the reportableFlag is used, if its value is one, a Report PDU MUST be returned to the sender under those conditions which can cause the generation of Report PDUs. Similarly, when the reportableFlag is used and its value is zero, then a Report PDU MUST NOT be sent. The reportableFlag MUST always be zero when the message contains a PDU from the Unconfirmed Class, such as a Report PDU, a response-type PDU (such as a Response PDU), or an unacknowledged notification-type PDU (such as an SNMPv2-trap PDU). The reportableFlag MUST always be one for a PDU from the Confirmed Class, include request-type PDUs (such as a Get PDU) and an acknowledged notification-type PDUs (such as an Inform PDU).

If the reportableFlag is set to one for a message containing a PDU from the Unconfirmed Class, such as a Report PDU, a response-type PDU (such as a Response PDU), or an unacknowledged notification-type PDU (such as an SNMPv2-trap PDU), then the receiver of that message MUST process it as though the reportableFlag had been set to zero.

If the reportableFlag is set to zero for a message containing a request-type PDU (such as a Get PDU) or an acknowledged notification-type PDU (such as an Inform PDU), then the receiver of that message must process it as though the reportableFlag had been set to one.

Report PDUs are generated directly by the SNMPv3 Message Processing Model, and support engine-to-engine communications, but may be passed to applications for processing.

An SNMP engine that receives a reportPDU may use it to determine what kind of problem was detected by the remote SNMP engine. It can do so based on the error counter included as the first (and only) varBind of the reportPDU. Based on the detected error, the SNMP engine may try to send a corrected SNMP message. If that is not possible, it may pass an indication of the error to the application on whose behalf the failed SNMP request was issued.

The authFlag and privFlag portions of the msgFlags field are set by the sender to indicate the securityLevel that was applied to the message before it was sent on the wire. The receiver of the message MUST apply the same securityLevel when the message is received and the contents are being processed.

There are three securityLevels, namely noAuthNoPriv, which is less than authNoPriv, which is in turn less than authPriv. See the SNMP architecture document [RFC2571] for details about the securityLevel.

a) authFlag

If the authFlag is set to one, then the securityModel used by the SNMP engine which sent the message MUST identify the securityName on whose behalf the SNMP message was generated and MUST provide, in a securityModel-specific manner, sufficient data for the receiver of the message to be able to authenticate that identification. In general, this authentication will allow the receiver to determine with reasonable certainty that the message was:

- sent on behalf of the principal associated with the securityName,

- was not redirected,

- was not modified in transit, and

- was not replayed.

If the authFlag is zero, then the securityModel used by the SNMP engine which sent the message must identify the securityName on whose behalf the SNMP message was generated but it does not need to provide sufficient data for the receiver of the message to authenticate the identification, as there is no need to authenticate the message in this case.

b) privFlag

If the privFlag is set, then the securityModel used by the SNMP engine which sent the message MUST also protect the scopedPDU in an SNMP message from disclosure, i.e., it MUST encrypt/decrypt the scopedPDU. If the privFlag is zero, then the securityModel in use does not need to protect the data from disclosure.

It is an explicit requirement of the SNMP architecture that if privacy is selected, then authentication is also required. That means that if the privFlag is set, then the authFlag MUST also be set to one.

The combination of the authFlag and the privFlag comprises a Level of Security as follows:

```
authFlag zero, privFlag zero -> securityLevel is noAuthNoPriv
authFlag zero, privFlag one  -> invalid combination, see below
authFlag one,  privFlag zero -> securityLevel is authNoPriv
authFlag one,  privFlag one  -> securityLevel is authPriv
```

The elements of procedure (see below) describe the action to be taken when the invalid combination of authFlag equal to zero and privFlag equal to one is encountered.

The remaining bits in msgFlags are reserved, and MUST be set to zero when sending a message and SHOULD be ignored when receiving a message.

6.5. msgSecurityModel

The v3MP supports the concurrent existence of multiple Security Models to provide security services for SNMPv3 messages. The msgSecurityModel field in an SNMPv3 Message identifies which Security Model was used by the sender to generate the message and therefore which securityModel must be used by the receiver to perform security processing for the message. The mapping to the appropriate securityModel implementation within an SNMP engine is accomplished in an implementation-dependent manner.

6.6. msgSecurityParameters

The msgSecurityParameters field of the SNMPv3 Message is used for communication between the Security Model modules in the sending and receiving SNMP engines. The data in the msgSecurityParameters field is used exclusively by the Security Model, and the contents and format of the data is defined by the Security Model. This OCTET STRING is not interpreted by the v3MP, but is passed to the local implementation of the Security Model indicated by the msgSecurityModel field in the message.

6.7. scopedPduData

The scopedPduData field represents either the plain text scopedPDU if the privFlag in the msgFlags is zero, or it represents an encryptedPDU (encoded as an OCTET STRING) which must be decrypted by the securityModel in use to produce a plaintext scopedPDU.

6.8. scopedPDU

The scopedPDU contains information to identify an administratively unique context and a PDU. The object identifiers in the PDU refer to managed objects which are (expected to be) accessible within the specified context.

6.8.1. contextEngineID

The contextEngineID in the SNMPv3 message, uniquely identifies, within an administrative domain, an SNMP entity that may realize an instance of a context with a particular contextName.

For incoming messages, the contextEngineID is used in conjunction with pduType to determine to which application the scopedPDU will be sent for processing.

For outgoing messages, the v3MP sets the contextEngineID to the value provided by the application in the request for a message to be sent.

6.8.2. contextName

The contextName field in an SNMPv3 message, in conjunction with the contextEngineID field, identifies the particular context associated with the management information contained in the PDU portion of the message. The contextName is unique within the SNMP entity specified by the contextEngineID, which may realize the managed objects referenced within the PDU. An application which originates a message provides the value for the contextName field and this value may be used during processing by an application at the receiving SNMP Engine.

6.8.3. data

The data field of the SNMPv3 Message contains the PDU. Among other things, the PDU contains the PDU type that is used by the v3MP to determine the type of the incoming SNMP message. The v3MP specifies that the PDU must be one of those specified in [RFC1905].

7. Elements of Procedure for v3MP

This section describes the procedures followed by an SNMP engine when generating and processing SNMP messages according to the SNMPv3 Message Processing Model.

Please note, that for the sake of clarity and to prevent the text from being even longer and more complicated, some details were omitted from the steps below.

a) Some steps specify that when some error conditions are encountered when processing a received message, a message containing a Report PDU is generated and the received message is discarded without further processing. However, a Report-PDU must not be generated unless the PDU causing generation of the Report PDU can be determine to be a member of the Confirmed Class, or the reportableFlag is set to one and the PDU class cannot be determined.

b) The elements of procedure do not always explicitly indicate when state information needs to be released. The general rule is that if state information is available when a message is to be "discarded without further processing", then the state information should also be released at that same time.

7.1. Prepare an Outgoing SNMP Message

This section describes the procedure followed to prepare an SNMPv3 message from the data elements passed by the Message Dispatcher.

1) The Message Dispatcher may request that an SNMPv3 message containing a Read Class, Write Class, or Notification Class PDU be prepared for sending.

   a) It makes such a request according to the abstract service primitive:

```
statusInformation =           -- success or errorIndication
  prepareOutgoingMessage(
  IN   transportDomain        -- requested transport domain
  IN   transportAddress       -- requested destination address
  IN   messageProcessingModel -- typically, SNMP version
  IN   securityModel          -- Security Model to use
  IN   securityName           -- on behalf of this principal
  IN   securityLevel          -- Level of Security requested
  IN   contextEngineID        -- data from/at this entity
  IN   contextName            -- data from/in this context
  IN   pduVersion             -- version of the PDU
  IN   PDU                    -- SNMP Protocol Data Unit
  IN   expectResponse         -- TRUE or FALSE *
  IN   sendPduHandle          -- the handle for matching
                              -- incoming responses
  OUT  destTransportDomain    -- destination transport domain
  OUT  destTransportAddress   -- destination transport address
  OUT  outgoingMessage        -- the message to send
  OUT  outgoingMessageLength  -- the length of the message
  )
```

* The SNMPv3 Message Processing Model does not use the values of expectResponse or pduVersion.

b) A unique msgID is generated. The number used for msgID should not have been used recently, and must not be the same as was used for any outstanding request.

2) The Message Dispatcher may request that an SNMPv3 message containing a Response Class or Internal Class PDU be prepared for sending.

a) It makes such a request according to the abstract service primitive:

```
result =                          -- SUCCESS or FAILURE
prepareResponseMessage(
 IN   messageProcessingModel   -- typically, SNMP version
 IN   securityModel            -- same as on incoming request
 IN   securityName             -- same as on incoming request
 IN   securityLevel            -- same as on incoming request
 IN   contextEngineID          -- data from/at this SNMP entity
 IN   contextName              -- data from/in this context
 IN   pduVersion               -- version of the PDU
 IN   PDU                      -- SNMP Protocol Data Unit
 IN   maxSizeResponseScopedPDU -- maximum size sender can accept
 IN   stateReference           -- reference to state
                               -- information presented with
                               -- the request
 IN   statusInformation        -- success or errorIndication
                               -- error counter OID and value
                               -- when errorIndication
 OUT  destTransportDomain      -- destination transport domain
 OUT  destTransportAddress     -- destination transport address
 OUT  outgoingMessage          -- the message to send
 OUT  outgoingMessageLength    -- the length of the message
 )
```

b) The cached information for the original request is retrieved via the stateReference, including

- msgID,
- contextEngineID,
- contextName,
- securityModel,
- securityName,
- securityLevel,
- securityStateReference,
- reportableFlag,
- transportDomain, and
- transportAddress.

The SNMPv3 Message Processing Model does not allow cached data to be overridden, except by error indications as detailed in (3) below.

3) If statusInformation contains values for an OID/value combination (potentially also containing a securityLevel value, contextEngineID value, or contextName value), then

   a) If reportableFlag is zero, then the original message is discarded, and no further processing is done. A result of FAILURE is returned. SNMPv3 Message Processing is complete.

   b) If a PDU is provided, it is the PDU from the original request. If possible, extract the request-id.

   c) A Report PDU is prepared:

      1) the varBindList is set to contain the OID and value from the statusInformation

      2) error-status is set to 0

      3) error-index is set to 0.

      4) request-id is set to the value extracted in step b) Otherwise, request-id is set to 0

   d) The errorIndication in statusInformation may be accompanied by a securityLevel value, a contextEngineID value, or a contextName value.

      1) If statusInformation contains a value for securityLevel, then securityLevel is set to that value, otherwise it is set to noAuthNoPriv.

2) If statusInformation contains a value for contextEngineID, then contextEngineID is set to that value, otherwise it is set to the value of this entity's snmpEngineID.

3) If statusInformation contains a value for contextName, then contextName is set to that value, otherwise it is set to the default context of "" (zero-length string).

e) PDU is set to refer to the new Report-PDU. The old PDU is discarded.

f) Processing continues with step 6) below.

4) If contextEngineID is not yet determined, then the contextEngineID is determined, in an implementation-dependent manner, possibly using the transportDomain and transportAddress.

5) If the contextName is not yet determined, the contextName is set to the default context.

6) A scopedPDU is prepared from the contextEngineID, contextName, and PDU.

7) msgGlobalData is constructed as follows

a) The msgVersion field is set to snmpv3(3).

b) msgID is set as determined in step 1 or 2 above.

c) msgMaxSize is set to an implementation-dependent value.

d) msgFlags are set as follows:

- If securityLevel specifies noAuthNoPriv, then authFlag and privFlag are both set to zero.

- If securityLevel specifies authNoPriv, then authFlag is set to one and privFlag is set to zero.

- If securityLevel specifies authPriv, then authFlag is set to one and privFlag is set to one.

- If the PDU is from the Unconfirmed Class, then the reportableFlag is set to zero.

- If the PDU is from the Confirmed Class then the reportableFlag is set to one.

- All other msgFlags bits are set to zero.

e) msgSecurityModel is set to the value of securityModel

8) If the PDU is from the Response Class or the Internal Class, then

a) The specified Security Model is called to generate the message according to the primitive:

```
statusInformation =
  generateResponseMsg(
  IN   messageProcessingModel -- SNMPv3 Message Processing
                              -- Model
  IN   globalData             -- msgGlobalData from step 7
  IN   maxMessageSize         -- from msgMaxSize (step 7c)
  IN   securityModel          -- as determined in step 7e
  IN   securityEngineID       -- the value of snmpEngineID
  IN   securityName           -- on behalf of this principal
  IN   securityLevel          -- for the outgoing message
  IN   scopedPDU              -- as prepared in step 6)
  IN   securityStateReference -- as determined in step 2
  OUT  securityParameters     -- filled in by Security Module
  OUT  wholeMsg               -- complete generated message
  OUT  wholeMsgLength         -- length of generated message
  )
```

If, upon return from the Security Model, the statusInformation includes an errorIndication, then any cached information about the outstanding request message is discarded, and an errorIndication is returned, so it can be returned to the calling application. SNMPv3 Message Processing is complete.

b) A SUCCESS result is returned. SNMPv3 Message Processing is complete.

9) If the PDU is from the Confirmed Class or the Notification Class, then

a) If the PDU is from the Unconfirmed Class, then securityEngineID is set to the value of this entity's snmpEngineID.

Otherwise, the snmpEngineID of the target entity is determined, in an implementation-dependent manner, possibly using transportDomain and transportAddress. The value of securityEngineID is set to the value of the target entity's snmpEngineID.

b) The specified Security Model is called to generate the message according to the primitive:

```
statusInformation =
 generateRequestMsg(
 IN  messageProcessingModel -- SNMPv3 Message Processing Model
 IN  globalData             -- msgGlobalData, from step 7
 IN  maxMessageSize         -- from msgMaxSize in step 7 c)
 IN  securityModel          -- as provided by caller
 IN  securityEngineID       -- authoritative SNMP entity
                            -- from step 9 a)
 IN  securityName           -- as provided by caller
 IN  securityLevel          -- as provided by caller
 IN  scopedPDU              -- as prepared in step 6
 OUT securityParameters     -- filled in by Security Module
 OUT wholeMsg               -- complete generated message
 OUT wholeMsgLength         -- length of the generated message
 )
```

If, upon return from the Security Model, the statusInformation includes an errorIndication, then the message is discarded, and the errorIndication is returned, so it can be returned to the calling application, and no further processing is done. SNMPv3 Message Processing is complete.

c) If the PDU is from the Confirmed Class, information about the outgoing message is cached, and a (implementation-specific) stateReference is created. Information to be cached includes the values of:

- sendPduHandle
- msgID
- snmpEngineID
- securityModel
- securityName
- securityLevel
- contextEngineID
- contextName

d) A SUCCESS result is returned. SNMPv3 Message Processing is complete.

7.2. Prepare Data Elements from an Incoming SNMP Message

This section describes the procedure followed to extract data from an SNMPv3 message, and to prepare the data elements required for further processing of the message by the Message Dispatcher.

1) The message is passed in from the Message Dispatcher according to the abstract service primitive:

```
result =                              -- SUCCESS or errorIndication
  prepareDataElements(
  IN  transportDomain                 -- origin transport domain
  IN  transportAddress                -- origin transport address
  IN  wholeMsg                        -- as received from the network
  IN  wholeMsgLength                  -- as received from the network
  OUT messageProcessingModel          -- typically, SNMP version
  OUT securityModel                   -- Security Model to use
  OUT securityName                    -- on behalf of this principal
  OUT securityLevel                   -- Level of Security requested
  OUT contextEngineID                 -- data from/at this entity
  OUT contextName                     -- data from/in this context
  OUT pduVersion                      -- version of the PDU
  OUT PDU                             -- SNMP Protocol Data Unit
  OUT pduType                         -- SNMP PDU type
  OUT sendPduHandle                   -- handle for matched request
  OUT maxSizeResponseScopedPDU        -- maximum size sender can accept
  OUT statusInformation               -- success or errorIndication
                                      -- error counter OID and value
                                      -- when errorIndication
  OUT stateReference                  -- reference to state information
                                      -- to be used for a possible
  )                                   -- Response
```

2) If the received message is not the serialization (according to the conventions of [RFC1906]) of an SNMPv3Message value, then the snmpInASNParseErrs counter [RFC1907] is incremented, the message is discarded without further processing, and a FAILURE result is returned. SNMPv3 Message Processing is complete.

3) The values for msgVersion, msgID, msgMaxSize, msgFlags, msgSecurityModel, msgSecurityParameters, and msgData are extracted from the message.

4) If the value of the msgSecurityModel component does not match a supported securityModel, then the snmpUnknownSecurityModels counter is incremented, the message is discarded without further processing, and a FAILURE result is returned. SNMPv3 Message Processing is complete.

5) The securityLevel is determined from the authFlag and the privFlag bits of the msgFlags component as follows:

   a) If the authFlag is not set and the privFlag is not set, then securityLevel is set to noAuthNoPriv.

b) If the authFlag is set and the privFlag is not set, then securityLevel is set to authNoPriv.

c) If the authFlag is set and the privFlag is set, then securityLevel is set to authPriv.

d) If the authFlag is not set and privFlag is set, then the snmpInvalidMsgs counter is incremented, the message is discarded without further processing, and a FAILURE result is returned. SNMPv3 Message Processing is complete.

e) Any other bits in the msgFlags are ignored.

6) The security module implementing the Security Model as specified by the securityModel component is called for authentication and privacy services. This is done according to the abstract service primitive:

```
statusInformation =            -- errorIndication or success
                               -- error counter OID and
                               -- value if error
  processIncomingMsg(
  IN  messageProcessingModel   -- SNMPv3 Message Processing Model
  IN  maxMessageSize           -- of the sending SNMP entity
  IN  securityParameters       -- for the received message
  IN  securityModel            -- for the received message
  IN  securityLevel            -- Level of Security
  IN  wholeMsg                 -- as received on the wire
  IN  wholeMsgLength           -- length as received on the wire
  OUT securityEngineID         -- authoritative SNMP entity
  OUT securityName             -- identification of the principal
  OUT scopedPDU,               -- message (plaintext) payload
  OUT maxSizeResponseScopedPDU -- maximum size sender can accept
  OUT securityStateReference   -- reference to security state
  )                            -- information, needed for
                               -- response
```

If an errorIndication is returned by the security module, then

a) If statusInformation contains values for an OID/value pair, then generation of a Report PDU is attempted (see step 3 in section 7.1).

   1) If the scopedPDU has been returned from processIncomingMsg then determine contextEngineID, contextName, and PDU.

   2) Information about the message is cached and a stateReference is created (implementation-specific).

Information to be cached includes the values of:

```
msgVersion,
msgID,
securityLevel,
msgFlags,
msgMaxSize,
securityModel,
maxSizeResponseScopedPDU,
securityStateReference
```

3) Request that a Report-PDU be prepared and sent, according to the abstract service primitive:

```
result =                        -- SUCCESS or FAILURE
returnResponsePdu(
IN  messageProcessingModel      -- SNMPv3(3)
IN  securityModel               -- same as on incoming request
IN  securityName                -- from processIncomingMsg
IN  securityLevel               -- same as on incoming request
IN  contextEngineID             -- from step 6 a) 1)
IN  contextName                 -- from step 6 a) 1)
IN  pduVersion                  -- SNMPv2-PDU
IN  PDU                         -- from step 6 a) 1)
IN  maxSizeResponseScopedPDU    -- from processIncomingMsg
IN  stateReference              -- from step 6 a) 2)
IN  statusInformation           -- from processIncomingMsg
)
```

b) The incoming message is discarded without further processing, and a FAILURE result is returned. SNMPv3 Message Processing is complete.

7) The scopedPDU is parsed to extract the contextEngineID, the contextName and the PDU. If any parse error occurs, then the snmpInASNParseErrs counter [RFC1907] is incremented, the security state information is discarded, the message is discarded without further processing, and a FAILURE result is returned. SNMPv3 Message Processing is complete. Treating an unknown PDU type is treated as a parse error is an implementation option.

8) The pduVersion is determined in an implementation-dependent manner. For SNMPv3, the pduVersion would be an SNMPv2-PDU.

9) The pduType is determined, in an implementation-dependent manner. For RFC 1905, the pduTypes include:

- GetRequest-PDU,
- GetNextRequest-PDU,
- GetBulkRequest-PDU,
- SetRequest-PDU,
- InformRequest-PDU,
- SNMPv2-Trap-PDU,
- Response-PDU,
- Report-PDU.

10) If the pduType is from the Response Class or the Internal Class, then

    a) The value of the msgID component is used to find the cached information for a corresponding outstanding Request message. If no such outstanding Request message is found, then the security state information is discarded, the message is discarded without further processing, and a FAILURE result is returned. SNMPv3 Message Processing is complete.

    b) sendPduHandle is retrieved from the cached information.

    Otherwise, sendPduHandle is set to <none>, an implementation defined value.

11) If the pduType is from the Internal Class, then

    a) statusInformation is created using the contents of the Report-PDU, in an implementation-dependent manner. This statusInformation will be forwarded to the application associated with the sendPduHandle.

    b) The cached data for the outstanding message, referred to by stateReference, is retrieved. If the securityModel or securityLevel values differ from the cached ones, it is important to recognize that Internal Class PDUs delivered at the security level of noAuthNoPriv open a window of opportunity for spoofing or replay attacks. If the receiver of such messages is aware of these risks, the use of such unauthenticated messages is acceptable and may provide a useful function for discovering engine IDs or for detecting misconfiguration at remote nodes.

    When the securityModel or securityLevel values differ from the cached ones, an implementation may retain the cached information about the outstanding Request message, in anticipation of the possibility that the Internal Class PDU

received might be illegitimate. Otherwise, any cached information about the outstanding Request message message is discarded.

c) The security state information for this incoming message is discarded.

d) stateReference is set to <none>

e) A SUCCESS result is returned. SNMPv3 Message Processing is complete.

12) If the pduType is from the Response Class, then

a) The cached data for the outstanding request, referred to by stateReference, is retrieved, including

- snmpEngineID
- securityModel
- securityName
- securityLevel
- contextEngineID
- contextName

b) If the values extracted from the incoming message differ from the cached data, then any cached information about the outstanding Request message is discarded, the incoming message is discarded without further processing, and a FAILURE result is returned. SNMPv3 Message Processing is complete.

When the securityModel or securityLevel values differ from the cached ones, an implementation may retain the cached information about the outstanding Request message, in anticipation of the possibility that the Response Class PDU received might be illegitimate.

c) Otherwise, any cached information about the outstanding Request message is discarded, and stateReference is set to <none>.

d) A SUCCESS result is returned. SNMPv3 Message Processing is complete.

13) If the pduType is from the Confirmed Class, then

a) If the value of securityEngineID is not equal to the value of snmpEngineID, then the security state information is

discarded, any cached information about this message is discarded, the incoming message is discarded without further processing, and a FAILURE result is returned. SNMPv3 Message Processing is complete.

b) Information about the message is cached and a stateReference is created (implementation-specific). Information to be cached includes the values of:

msgVersion,
msgID,
securityLevel,
msgFlags,
msgMaxSize,
securityModel,
maxSizeResponseScopedPDU,
securityStateReference

c) A SUCCESS result is returned. SNMPv3 Message Processing is complete.

14) If the pduType is from the Unconfirmed Class, then A SUCCESS result is returned. SNMPv3 Message Processing is complete.

8. Intellectual Property

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification can be obtained from the IETF Secretariat.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this standard. Please address the information to the IETF Executive Director.

9. Acknowledgements

This document is the result of the efforts of the SNMPv3 Working Group. Some special thanks are in order to the following SNMPv3 WG members:

Harald Tveit Alvestrand (Maxware)
Dave Battle (SNMP Research, Inc.)
Alan Beard (Disney Worldwide Services)
Paul Berrevoets (SWI Systemware/Halcyon Inc.)
Martin Bjorklund (Ericsson)
Uri Blumenthal (IBM T. J. Watson Research Center)
Jeff Case (SNMP Research, Inc.)
John Curran (BBN)
Mike Daniele (Compaq Computer Corporation)
T. Max Devlin (Eltrax Systems)
John Flick (Hewlett Packard)
Rob Frye (MCI)
Wes Hardaker (U.C.Davis, Information Technology - D.C.A.S.)
David Harrington (Cabletron Systems Inc.)
Lauren Heintz (BMC Software, Inc.)
N.C. Hien (IBM T. J. Watson Research Center)
Michael Kirkham (InterWorking Labs, Inc.)
Dave Levi (SNMP Research, Inc.)
Louis A Mamakos (UUNET Technologies Inc.)
Joe Marzot (Nortel Networks)
Paul Meyer (Secure Computing Corporation)
Keith McCloghrie (Cisco Systems)
Bob Moore (IBM)
Russ Mundy (TIS Labs at Network Associates)
Bob Natale (ACE*COMM Corporation)
Mike O'Dell (UUNET Technologies Inc.)
Dave Perkins (DeskTalk)
Peter Polkinghorne (Brunel University)
Randy Presuhn (BMC Software, Inc.)
David Reeder (TIS Labs at Network Associates)
David Reid (SNMP Research, Inc.)
Aleksey Romanov (Quality Quorum)
Shawn Routhier (Epilogue)
Juergen Schoenwaelder (TU Braunschweig)
Bob Stewart (Cisco Systems)
Mike Thatcher (Independent Consultant)
Bert Wijnen (IBM T. J. Watson Research Center)

The document is based on recommendations of the IETF Security and Administrative Framework Evolution for SNMP Advisory Team. Members of that Advisory Team were:

David Harrington (Cabletron Systems Inc.)
Jeff Johnson (Cisco Systems)
David Levi (SNMP Research Inc.)
John Linn (Openvision)
Russ Mundy (Trusted Information Systems) chair
Shawn Routhier (Epilogue)
Glenn Waters (Nortel)
Bert Wijnen (IBM T. J. Watson Research Center)

As recommended by the Advisory Team and the SNMPv3 Working Group Charter, the design incorporates as much as practical from previous RFCs and drafts. As a result, special thanks are due to the authors of previous designs known as SNMPv2u and SNMPv2*:

Jeff Case (SNMP Research, Inc.)
David Harrington (Cabletron Systems Inc.)
David Levi (SNMP Research, Inc.)
Keith McCloghrie (Cisco Systems)
Brian O'Keefe (Hewlett Packard)
Marshall T. Rose (Dover Beach Consulting)
Jon Saperia (BGS Systems Inc.)
Steve Waldbusser (International Network Services)
Glenn W. Waters (Bell-Northern Research Ltd.)

10. Security Considerations

The Dispatcher coordinates the processing of messages to provide a level of security for management messages and to direct the SNMP PDUs to the proper SNMP application(s).

A Message Processing Model, and in particular the V3MP defined in this document, interacts as part of the Message Processing with Security Models in the Security Subsystem via the abstract service interface primitives defined in [RFC2571] and elaborated above.

The level of security actually provided is primarily determined by the specific Security Model implementation(s) and the specific SNMP application implementation(s) incorporated into this framework. Applications have access to data which is not secured. Applications should take reasonable steps to protect the data from disclosure, and when they send data across the network, they should obey the securityLevel and call upon the services of an Access Control Model as they apply access control.

The values for the msgID element used in communication between SNMP entities must be chosen to avoid replay attacks. The values do not need to be unpredictable; it is sufficient that they not repeat.

When exchanges are carried out over an insecure network, there is an open opportunity for a third party to spoof or replay messages when any message of an exchange is given at the security level of noAuthNoPriv. For most exchanges, all messages exist at the same security level. In the case where the final message is an Internal Class PDU, this message may be delivered at a level of noAuthNoPriv or authNoPriv, independent of the security level of the preceding messages. Internal Class PDUs delivered at the level of authNoPriv are not considered to pose a security hazard. Internal Class PDUs delivered at the security level of noAuthNoPriv open a window of opportunity for spoofing or replay attacks. If the receiver of such messages is aware of these risks, the use of such unauthenticated messages is acceptable and may provide a useful function for discovering engine IDs or for detecting misconfiguration at remote nodes.

This document also contains a MIB definition module. None of the objects defined is writable, and the information they represent is not deemed to be particularly sensitive. However, if they are deemed sensitive in a particular environment, access to them should be restricted through the use of appropriately configured Security and Access Control models.

11. References

[RFC1901] The SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Introduction to Community-based SNMPv2", RFC 1901, January 1996.

[RFC2578] McCloghrie, K., Perkins, D. and J. Schoenwaelder, "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, April 1999.

[RFC1905] The SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1905, January 1996.

[RFC1906] The SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Transport Mappings for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1906, January 1996.

[RFC1907] The SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Management Information Base for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1907 January 1996.

[RFC1908] The SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Coexistence between Version 1 and Version 2 of the Internet-standard Network Management Framework", RFC 1908, January 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2028] Hovey, R. and S. Bradner, "The Organizations Involved in the IETF Standards Process", BCP 11, RFC 2028, October 1996.

[RFC2571] Harrington, D., Presuhn, R. and B. Wijnen, "An Architecture for describing SNMP Management Frameworks", RFC 2571, April 1999.

[RFC2574] Blumenthal, U. and B. Wijnen, "The User-Based Security Model for Version 3 of the Simple Network Management Protocol (SNMPv3)", RFC 2574, April 1999.

[RFC2575] Wijnen, B., Presuhn, R. and K. McCloghrie, "View-based Access Control Model for the Simple Network Management Protocol (SNMP)", RFC 2575, April 1999.

[RFC2573] Levi, D., Meyer, P. and B. Stewart, "SNMP Applications", RFC 2573, April 1999.

[RFC2570] Case, J., Mundy, R., Partain, D. and B. Stewart, "Introduction to Version 3 of the Internet-standard Network Management Framework", RFC 2570, April 1999.

12. Editors' Addresses

Jeffrey Case
SNMP Research, Inc.
3001 Kimberlin Heights Road
Knoxville, TN 37920-9716
USA

Phone: +1 423-573-1434
EMail: case@snmp.com

Dave Harrington
Cabletron Systems, Inc
Post Office Box 5005
Mail Stop: Durham
35 Industrial Way
Rochester, NH 03867-5005
USA

Phone: +1 603-337-7357
EMail: dbh@ctron.com

Randy Presuhn
BMC Software, Inc.
965 Stewart Drive
Sunnyvale, CA 94086
USA

Phone: +1 408-616-3100
EMail: randy_presuhn@bmc.com

Bert Wijnen
IBM T. J. Watson Research
Schagen 33
3461 GL Linschoten
Netherlands

Phone: +31 348-432-794
EMail: wijnen@vnet.ibm.com

13. Changes From RFC 2272

The following change log records major changes from the previous version of this document, RFC 2272.

- Updated contact information for editors.
- Made parameter identification in prepareResponseMessage() consistent, both internally and with architecture.
- Made references to processIncomingMsg() consistent, both internally and with architecture.
- Deleted superfluous expectResponse parameter from processIncomingMsg(), consistent with architecture.

- Fixed typos.

- Removed sending of a report PDU from step 4 on page 30 on RFC 2272.

- Use "PDU Class" terminology instead of directly using RFC 1905 PDU types, in order to potentially allow use of new PDU types in the future.

- Added intro document to references.

- Made various clarifications to the text.

- The handling of the reportableFlag has been made consistent.

- The acknowledgement list has been updated.

14. Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.