# Distributed MIB Object Information Service

*S. Boros, B. Helthuis, A. Pras*

Computer Science Department, University of Twente,
P.O. Box 217, 7500 AE Enschede, The Netherlands,
email: boros@cs.utwente.nl

*Abstract*

With current SNMP manager systems, every possible MIB module specification should be stored within the manager's system to allow such a system to display management information in the correct format and to provide the manager with a description of the information. To relief manager systems from storing all possible MIB module specifications a priori, this paper introduces a MIB object information service. Such service allows SNMP manager applications to retrieve information regarding entire SNMP MIB modules as well as individual objects. The distributed nature of the service is beneficial to avoid copyright problems and to increase responsiveness.

## 1 Introduction

The Internet standard network management framework defines the structure of management information (SMI), a management information base (MIB), and a communication protocol between management agents and management stations (SNMP). The objects being managed via SNMP are grouped together in hierarchical, virtual tree like structure called MIB. Objects in the MIB are organized as described by the Stucture of Management Information [1].
In order to manage SNMP devices, network management applications need to have access to (a sometimes large) collection of MIB definition files, which have to be 'compiled' into the application. This means parsing and validating all required MIB files and loading them into an internal data structure, after which they can be used for the actual work of the application.
The MIB Object Information Service is lookup service for MIB module definitions. The idea is to relieve the management application (and its designer) of the burden of all the MIB module definition handling code by providing a small and simple interface for looking up symbols (managed objects, type definitions, macro definitions) by name or by numerical ID.
A rough calculation indicates that there exist over 150 MIB modules (maintained by IETF working groups, IANA/ICANN) and about 13000 managed objects therein. If we consider also MIB modules of vendors and individual persons the numbers grow far higher. This amount of MIB data can be too much for a single server to handle. By distributing the service over several interconnected servers the load placed on individual servers can be reduced. This will eventually improve responsiveness. Having vendors to implement the service on their own servers in order to serve information about their own MIB modules can solve copyright related issues.

The chapters of this paper are organized as follows:
Chapter 2 presents the single server model. Chapter 3 presents the distributed service, describing the participating entities and the messages that are exchanged between these entities. An example of message processing is also given. In Chapter 4 conclusions are drawn and possible further work is identified.

## 2 The single server model

The single server architecture of the MIB Object Information Service (Figure 1.) shows a number of SNMP management applications, which are connected in the role of clients to the central server. Access to the service can be gained in one of the following ways:

- linking a library to the application and performing function calls.
- forking an external program to do the work and using the resulting data.
- manual lookup of definitions via a command line application
- HTML-formatted lookup via a Web browser

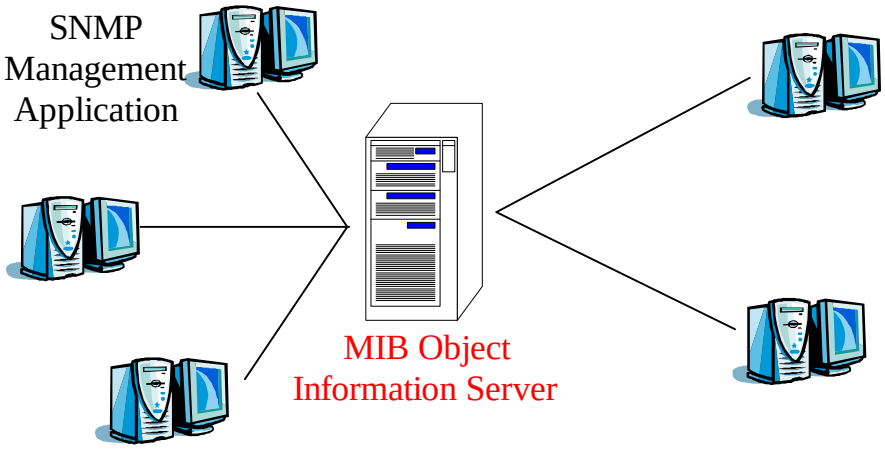


*Figure 1. The single server architecture of the MIB Object Information Service*

Currently a management application can query the server [3] for the following information:

1. search a Symbolic Name or a Fully Qualified Numeric Object Identifier. The result will be either empty or a list of one or more matches;
2. ask for the list of MIB modules the server supports. The result will be a list of 0 or more module names (strings);
3. ask for the entire MIB module;
4. ask for the Imports of a module;
5. ask for the Identity description of a module.

Output encodings are in perl, xml, php, python and tcl. Plans for future encodings include: java, html and bytestream types.

Internally the structure of the output is independent of the encoding type choosen to represent the output. The structure describes the order, relations and hierarchy of individual 'entities' in the returned data structure, whereas the encoding type says how the elements are written to the output stream. The structure is always the same, whatever output encoding type is requested.

## 3 Distributed MIB Object Information Service

Consider the case of a client querying his well-known server about information on a MIB module, a MIB object etc. If the server does not support that MIB module, object etc. the client has two possiblities: either to give up (leaving unsatisfied network managers behind) or to look for another server that supports that MIB module, object. The second possibility requires from the client (an SNMP management application) extra effort.

To avoid such situations (ie. increase responsiveness) the distribution of the service over several interconnected servers becomes necessary.

### *3.1 Entities*

Within the Distributed MIB Object Information Service each server includes the functions that were needed within the single server architecture, as well as new functions needed to communicate with other servers.

The communication between MIB Object Information Servers is accomplished via HTTP messages. Every server is required to implement an HTTP server, to fulfill HTTP GET and POST messages. In this perspective, the internal structure of a server contains an HTTP server, an HTTP client as well as the lookup engine described in Chapter 2.

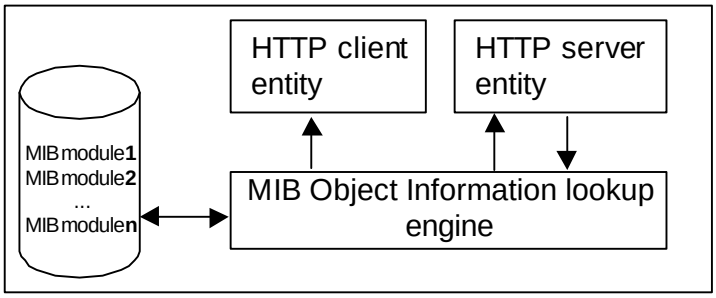


*Figure 2. Internal structure of a MIB Object Information Server*

There is a neighbouring relation ("friendship") defined between servers as follows: ServerB is called the friend of ServerA if the operators of the two servers decide to have a bilateral agreement concerning the exchange of information about supported MIB modules/objects. This relationship is commutative.

Messages aimed at the exchange of information concerning the supported MIB modules/objects (the meta-information about MIB modules/objects) are called administrative messages. The servers will exchange administrative messages only with their friends. The information will still propagate through the whole network of MIB Object Information Servers given the paradigm of the "friend of my friend". Note, however, that this is not the same as the transitivity of the "friendship" relation.

A different class of messages are the messages that carry actual MIB information. These messages can be exchanged between any two MIB Object Information Servers. In this case one of the servers acts in the role of a client, as described in Chapter 2.

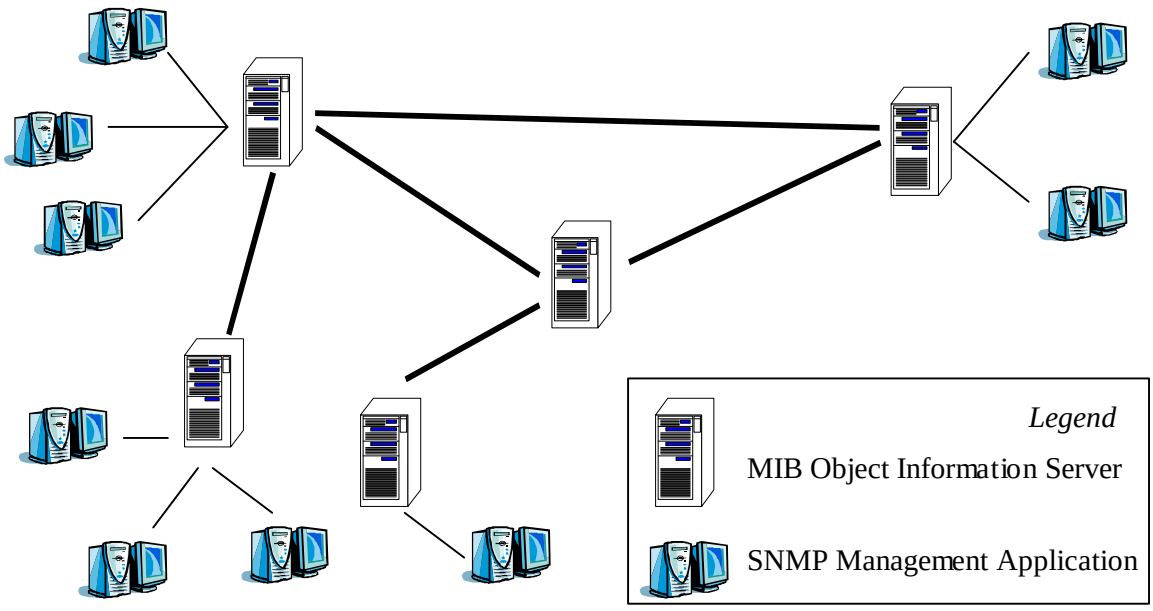


*Figure 3. Distributed MIB Object Information Service Architecture*

### *3.2 Exchanged information*

The messages that are exchanged between MIB Object Information Servers can be classified into two groups:

1. administrative messages

2. messages that cary MIB object information

By administrative messages a server informs its friends about the MIB modules and objects that it supports (ie. it can provide information about the modules and the objects within).

For the messages that carry actual MIB object information, one of the servers takes the role of the client of the service and approaches the other server via the HTTP protocol. The entity playing the role of the client can issue any of the queries mentioned in Chapter 2, and can choose to cache the answers for later use.

In this paragraph we are going to define the administrative messages transferred between two servers. For every advertised MIB module, the following information can be extracted from the message and it is to be stored on the peer server:

- MIB module name and numeric identity
- last update of the MIB module
- OIDs and symbolic names of the top level node(s) within the module
- time-out value of the objects of this module
- the source of this information

This information is conveyed through the update(…) and i_have(…) administrative messages, which are described in detail in the next section. The server that owns the master version of this MIB module/objects is called the authoritative server for the module. The authoritative server determines the values as follows.

Numeric OIDs and symbolic names of the top-level nodes of the modules are extracted parsing the module. The MIB module's name is also determined by parsing the module. The last update value and the numeric identity are extracted from the MODULE-IDENTITY field; therefore this information is only available in case of SMIv2 modules [2]. For SMIv1 MIB modules, these values default to null. The time-out value (in seconds) for the objects is set by the authoritative server to a preconfigured value. Given the type of information that is being exchanged, a period of several days, weeks or even months is suggested. The source is the URL of the authoritative server.

### *3.3 Messages*

Currently there are three administrative messages defined:

- i_have(seqnr, timestamp, mib_module1, mib_module2, ...)
- update(seqnr, timestamp, mib_module1, mib_module2, …)
- what_do_you_have()

An i_have(...) message is sent whenever a server boots up, or whenever it is requested to send a complete list of MIB objects/modules it supports and knows about. Triggering of this second event is done by sending a what_do_you_have() message to this server. This is usually done by servers that boot up (to get informed about the current set-up) or by servers that lost synchronization due to malfunction or lost HTTP messages.

The most used messages by a server is supposed to be the update message, which gives the difference since the last message was sent. The sequence numbers of the update messages and the i_have messages are different.

To convey the information between friends the data is XML encoded and sent on the wire embedded into HTTP messages. The following example shows the message body of an HTTP message that carries a complete i_have(…) message:

```
<i_have seqnr = 1 timestamp = 200104050000Z >
    <module name = IF-MIB id = 1.3.6.1.2.1.31 last-update = 200006140000Z>
        <source>www.simpleweb.org/ietf/mibs/</source>
        <timeout> 2592000</timeout>
        <TLnode name = interfaces id = 1.3.6.1.2.1.2 ></TLnode>
        <TLnode name = ifMIBObjects id = 1.3.6.1.2.1.31.1 ></TLnode>
        <TLnode name = ifConformance id = 1.3.6.1.2.1.31.2 ></TLnode>
    </module>
</i_have>
```

### *3.4 Message processing*

In this paragraph the processing of an update(…) administrative message is presented. The message is sent by Server1 (S1) to its friend Server2 (S2).

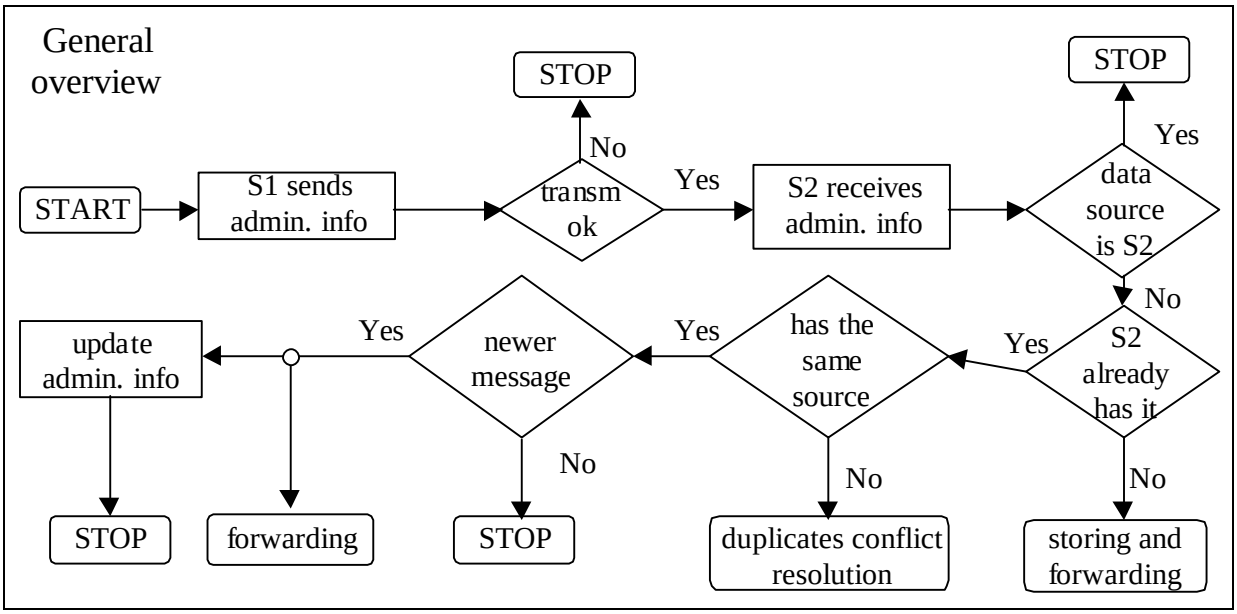


*Figure 4. General overview of the update message processing*

First Server1 sends the information about MIB modules/objects that it supports, or forwards information received from a different server. If the transmission of the HTTP message that carries this message is error-free, the message is delivered to S2. At this moment the processing on Server2 starts. The following steps are made:

- S2 decides if the source of the message differs from itself, otherwise it has to stop to avoid the infinite looping of the packets;
- if the source is different from itself, S2 will check if it already owns this information;
- if not, it may store and forward the information;
- otherwise S2 checks the source of the old information. If the the source of that information is different from the source of the information in the newly arrived packet, then further checkings become necessary for S2 to decide what to do with the duplicate information;
- in case that the source of the already stored information and the source of the information under processing is the same, S2 has to check if the message is really newer (ie. it is not a message that was lost in the network and it is delivered late). For this, S2 keeps for all his friend servers track of the sequence number of the last message and timestamp. If the received message has a lower sequence number than expected, the message will be discarded and the processing stops. In case we adopt the delta updates method, the sequence

numbers coming from any neighbouring server have to be succesive. That means, if the sequence number of the newly arrived message is not the sequence number of the last message from the same server plus one, there was an error in the transmission, S2 lost synchronization with this neighbour of its. This requires S1 to resend all the information that it is aware of, not only the difference from the last message[1]. If the message proves to be the expected one, all the administritave information already stored concerning the same MIB objects/module, has to be updated and the information may be forwarded to the friend servers.

The next picture shows how a server deals with duplicate information. Server2 has to decide if the MIB module is older, newer or the same as the information it already has. In case it is newer, Server2 still has to decide whether to store it or not. If the module is older that the one already stored on Server2 then the processing stops, avoiding the propagation of a presumably outdated information. If the module is the same as the one already stored (this is the case with all SMIv1 MIB modules, as one cannot decide the "age" of the module), the information may either be stored and forwarded, or just forwarded. These decisions are taken based on some upper level policies.

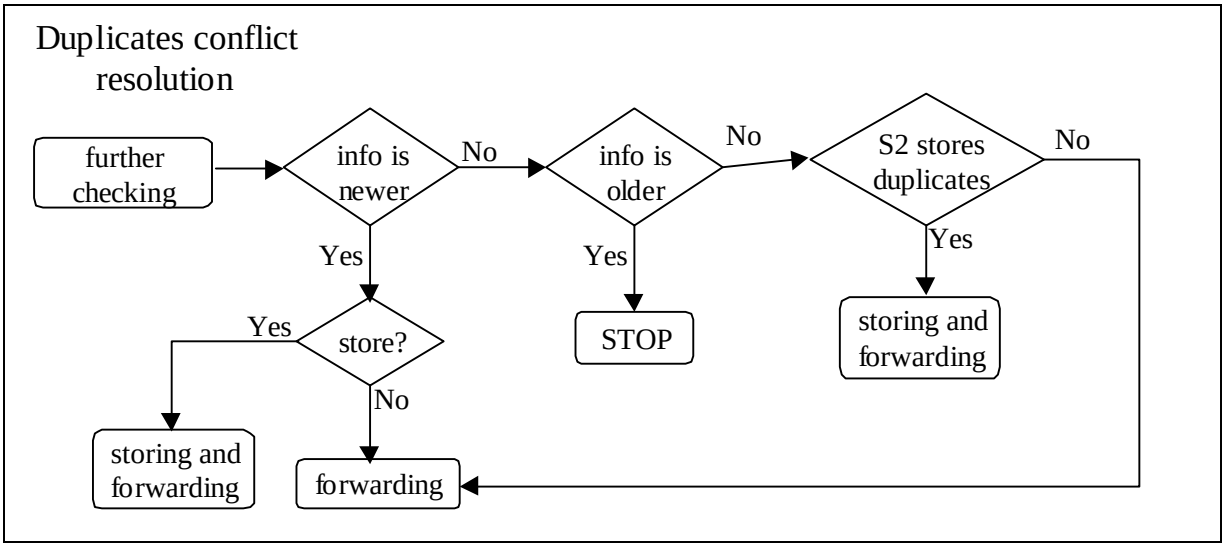


*Figure 5. Duplicates conflict resolution*

Storing and forwarding of the received administration information on Server2 involves the following steps: first Server2 has to decide if it wants to store this information (a reason not to do that may be insufficient disc space). Also a decision should be made whether the information should be forwarded or not. Forwarding the information means sending the same, unmodified data on the wires to all "friends".

[1] S2 notifies S1 to resend all information via a what_do_you_have() message.

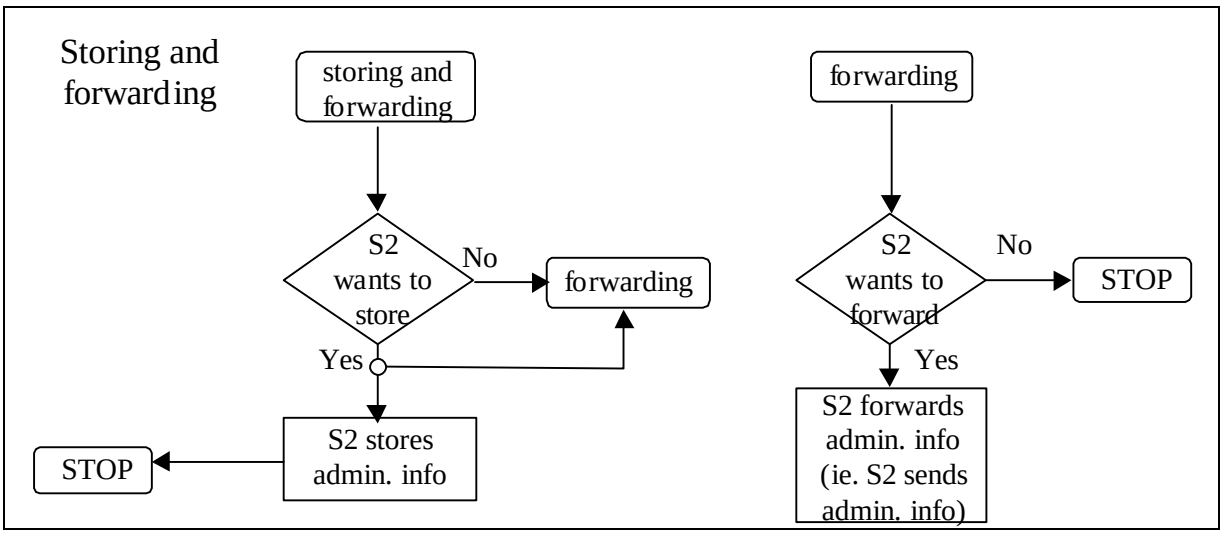


*Figure 6. Storing and forwarding process of the update message*

## 4 Conclusions and future work

For the single server model a web interface of the service is available at http://www.simpleweb.org/ietf/mibs/. It is implemented using the libsmi [4] library developed by the Technical University of Braunschweig. The process of implementing a prototype of the distributed service is carried out at this moment.

There are several questions that are still to be answered, such as: clarifying the semantics of the update messages, analyze security issues.

## Bibliography

[1] William Stallings, "*SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*", Addison Wesley Longman, Inc. 1999

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

[3] http://www.simpleweb.org/ietf/mibs/

[4] http://www.ibr.cs.tu-bs.de/projects/libsmi/