Remote ***MIB item*** look-up service

Aiko Pras, Szabolcs Boros and Bert Helthuis,
Centre for Telematics and Information Technology
University of Twente,
P.O. Box 217, 7500 AE Enschede, The Netherlands,
email: {pras | boros}@ctit.utwente.nl

Abstract: Despite some deficiencies, the Internet management framework is widely deployed and thousands of Management Information Base (MIB) modules have been defined thus far. These modules are used by implementors of agent software, as well as by managers and management applications, to understand the syntax and semantics of the management information that may be exchanged. At the manager's side, MIB modules are usually stored in separate files, which are maintained by the human manager and read by the management application. Since maintenance of this file repository can be cumbersome, management applications are often confronted with incomplete and outdated information. To solve this "meta-management" problem, this paper discusses the design of a remote look-up service for MIB item definitions. Such service facilitates the retrieval of missing MIB module definitions, as well as definitions of individual MIB items. Initially the service may be provided by a single server, but other servers can be added at later stages to improve performance and prevent copyright problems. It is envisaged that vendors of network equipment will also install servers, to distribute their vendor specific MIBs. The paper describes how the service, which is provided on a best effort basis, can be accessed by managers / management applications, and how internally servers inform each other about the MIB modules they support.

Keywords: Internet management, Management Information Base, MIB module, meta-management

1 Introduction

It is already 13 years back that the first definition appeared of the Management Information Base (MIB) for the Internet [1]. This definition was replaced in 1991 by the MIB-II [2], which defines 170 objects for the TCP/IP protocol stack and is now a full IETF standard. Since than a large number of RFCs, defining 180 different standards track MIB modules, appeared. The total number of items defined in these modules is around 12500 (in this article the term "item" will be used to denote object types, object identifiers, notifications, textual conventions, etc.). Next to the MIB modules defined in RFCs, many working groups within the IETF are currently working on updated or additional MIB modules, defined in more than 200 internet drafts[1].

MIB modules are not only defined by the IETF, but also by vendors of network equipment. The website of mibCentral [3], for example, hosted in May 2001 for twenty of the top vendors around 2350 MIB modules. According to that website, the total number of items defined by these modules is more than 200000. To get an idea of the number of MIB modules that might be defined worldwide, it is important to know that 9700 organisations already obtained an enterprise ID from IANA [4]. Under the assumption that the number of modules / objects defined by a top vendor is ten times more than that defined by an average organisation, the total number of MIB modules defined worldwide might be around 100000 and the total number of MIB items might be around 10 million. Although these numbers are maximum values and real networks will support just a small fraction, it is still likely that managers of real networks will be confronted with hundreds of different MIB modules. This number is not fixed, but grows whenever new devices get connected to the network.

[1] Some of these internet drafts update existing MIBs; others are new but will probably never be standardized.

This high and changing number raises several problems. One of these problems is that several management applications, like for example MIB browsers, need to know the definitions of all MIB items that they are confronted with. For this purpose these applications load a number of MIB module files, which are usually stored on the manager's system. It is the responsibility of the human manager to maintain this repository of MIB module files. This maintenance task, which can be seen as "management of the management system" (meta-management), is not always straightforward. After an application has indicated that the definition of a certain item is missing, the manager has to determine in which MIB module the item should be defined. To find the missing MIB module file, the manager may have to search the disks of his management system as well as his documentation CDs. If the search is without success, the manager may have to send emails to the equipment vendor, post questions in news groups, perform searches at FTP and web sites[1], etc. After the manager has found the correct MIB module, it may be necessary to transform the file into the proprietary format understood by the management application. In some cases the MIB module contains syntactical errors, which must be corrected by the manager before the file can be used by conversion tools and management applications.
Another problem of having millions of MIB item definitions, is that it may be difficult to find the right one. If the manager wants, for example, to know which objects provide information on the load of his systems, the manager could perform a `grep` on the MIB module files stored in his local repository. The chance is realistic, however, that his local repository does not contain the definitions for the most appropriate objects to measure load. As a result, the manager may not be able to determine system load reliably.

To solve the meta-management problem of maintaining the local MIB module repository, this paper discusses the design of a remote look-up service for MIB item definitions. The service allows the retrieval of complete MIB module definition files, as well as the definitions of individual MIB items. The service can be called by management applications that require the definition of certain MIB items, as well as human managers who are searching for specific management information.

The structure of the paper is as follows. Section 2 discusses the characteristics and main design decisions of the MIB item look up service. The interactions between the management systems, which will be called clients, and the systems that store the MIB item definitions, which will be called servers, are presented in Section 3. Section 4 discusses the messages which are exchanged between server systems, including their parameters and the way they are processed. Conclusions and recommendations are given in Section 5.

2 Design decisions

A first requirement is to facilitate a step-wise introduction of the MIB item look-up service. It should be possible to start the service from a single server, such as the Simpleweb [5], and add other servers at later stages. Although it seems technically possible to store all existing MIB modules on a single server (the amount of disk space needed to store all modules is probably less than 10 Gbyte), there are several advantages of running multiple servers: the load will be shared, the response times reduced and the resilience improved.
Next to these performance advantages, the introduction of multiple servers may also be interesting to create "specialized servers". The IETF, for example, may operate his own server to distribute the MIB modules that have been developed by the working groups within the IETF. Likewise, an equipment vendor may operate a server to distribute his proprietary MIB information; vendors of management software may have special servers that function in conjunction with their own software, etc. By introducing specialised servers, it is no longer necessary that

[1] On average 30 people per day use the 'pointers to enterprise specific MIBs' page on the Simpleweb [5].

every server stores every known piece of MIB information. A complication of specialised servers, however, is that servers should now be able to forward requests from one server to another; ways in which this can be achieved will be presented in Section 3.
An additional advantage of having multiple servers, is that it becomes possible to deal with copyrighted MIB information that may not be distributed from servers owned by other organisations. Note that copyrights can also be used by vendors to ensure that MIB information will only be down-loaded from their own servers; the statistics obtained from these servers may be strategic to determine which MIB modules should be implemented in future devices and which not.

Despite the fact that there are a couple of good reasons to introduce multiple servers, the authors do not expect that there will ever be a large number of servers. In particular the authors do not believe that there will ever emerge a complete hierarchical directory of MIB module servers, like for example the DNS, that use the OID naming tree to structure the directory. Such hierarchical system seems to rigid and requires to many organizations to participate. A system of loosely couple servers, each being free to serve whatever MIB module they like, seems to be a better alternative.

The basic architecture of the MIB item look-up service is shown in Figure 1. This figure shows five servers, which are loosely coupled via a mesh structure. The idea is that servers only connect to “friend” servers. It is up to the owner of a server to decide who are the “friends” and who are not; the negotiation to determine who might be a “friend” is performed off-line (email, telephone) and will not be discussed in this paper. A “friend” relationship is commutative, but not transitive. “Friends” can be hard coded in the server’s configuration files.

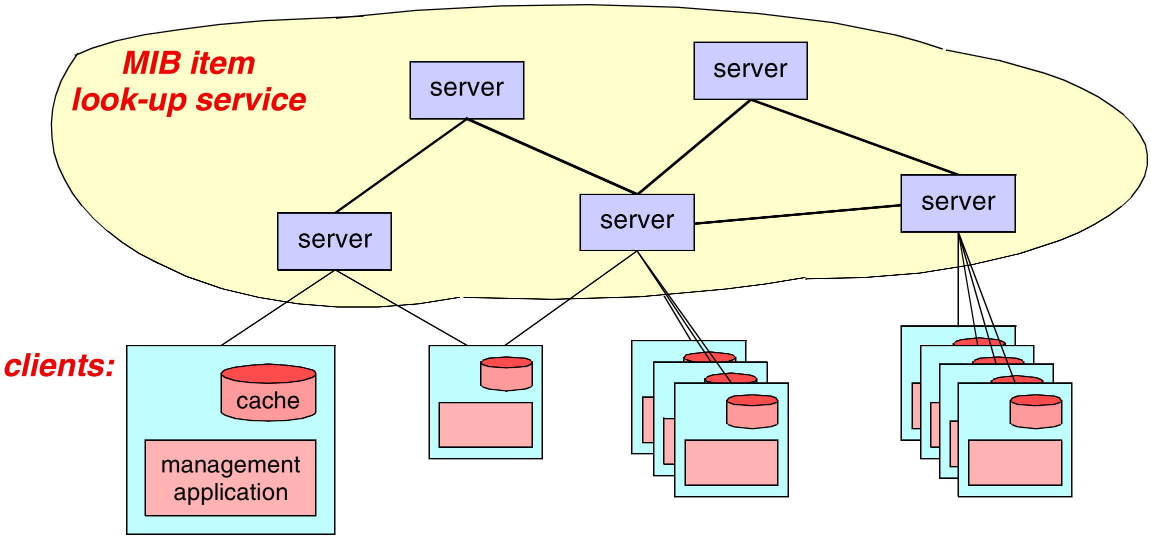

Figure 1: Architecture of the MIB item look-up service

The architecture supports two kind of messages: *data messages*, which contain the actual MIB information, and *administrative messages*, which are used by servers to inform each other about the MIB information that is supported. Administrative messages may only be exchanged between servers; data messages can be exchanged between servers, as well as between client and server. Clients, which are the actual management systems, may be connected to one or more servers.

Another design decision was to create a *best effort* service, and not a guaranteed service. The concept of best effort services is well accepted within the Internet, and is already an important step forward compared to the current situation, which was described in Section 1. To ensure that failures of the look-up service will not degrade the operation of management applications below the level of today, every management system should cache retrieved MIB modules for a while. The information within the cache will be used in case of failures of the look-up service, as well as to improve performance.

Not only the availability of the service, but also the quality of the provided information will be best effort. This implies that the service may not always return the best possible information; it is, for example, conceivable that the service returns no information or only outdated information. The reason behind this decision, is that it seems impossible to collect every MIB module that has been defined thus far and include it in the service; this would require that every organisation that defined a MIB module will be prepared to participate by either installing a server, or by giving his MIB modules to other organizations, who will put it on servers elsewhere. Best effort is also sufficient: the goal was to simplify the meta-management problem of maintaining the repository of MIB module files; a situation in which the service is able to deliver 99% of the desired information, is a step in the right direction and therefore acceptable.

3 Client-server interactions

The MIB item look-up service can be used by clients, which are the systems that run management applications, to obtain the definition of one or more MIB items. MIB items can be object types, object identifiers, notifications, textual conventions, module identities, group statements, compliance statements, etc. [7]. A client can specify the item in which he is interested by using the fully qualified numeric object identifier, or the symbolic name. In case multiple items satisfy the request (the ifTable, for example, is defined in the MIB-II [2] as well as in the Interface MIB [8]), all items will be returned. The client can refine the search by using switches to indicate that he is only interested in the most recently defined item, or in the item that has the highest level on the standards track. In addition to MIB items, the client may also request complete MIB modules or, for example, the imports of a module.

The *data messages* that are exchanged between client and server make use of the HTTP protocol. The information within the HTTP PDUs can be HTML or XML encoded. HTML encoded information can be displayed in traditional web browsers and may be helpful for human managers to find specific objects. XML encoding is easier to parse by computer programs and is therefore better suited for management applications. The XML format can be the one that is under development by the IRTF Network Management Research Group (NMRG) [14], as well as some proprietary format that can directly be used in conjunction with management applications written in perl, php, python, tcl etc.

In case a server receives a request for a MIB item that is not stored on the server's local system, but the server is aware of another server that does have the item, the server may either fetch the item from the other server, or return to the client the name of that other server. In the last case the client should re-issue the request to that other server.

A prototype server, called the *SimpleMIB look-up service* [6], is currently running at the Simpleweb [5]. To retrieve, for example, the description of the ifTable in perl, the parameters *?module=IF-MIB&object=ifTable&encoding=perl* should be used. The implementation of the server uses the libsmi library [9], which was developed by the Technical University of Braunschweig.

4 Server-server interactions

The *data messages* that were discussed in Section 3, can also be used by servers to obtain MIB items from other servers. In most cases servers will only request complete MIB modules, and store these modules on their local system. In addition to *data messages*, servers may also exchange *administrative messages* to inform their “friends” which MIB modules they are aware of. Although “friend” relationship is commutative and not transitive, MIB module information may still propagate through the entire network of servers, because it is at the discretion of every server to decide which part of his knowledge will be forwarded, and which not. To a certain extent *administrative messages* are comparable to the *distance vectors* that are exchanged by certain routing protocols: distance vector PDUs are only exchanged between neighbours, but the routing information within these PDUs may be used to inform others [10].
In order to avoid loops, servers won’t propagate MIB item information originated by them, nor information that was already forwarded by them.

The remainder of this chapter will explain which administrative messages exist (Section 4.1), how these messages are processed (Section 4.2) as well as a number of usage scenarios (Section 4.3)

4.1 Message formats

Three kind of administrative messages exist:

- *i_know* messages,
- *update* messages,
- *what_do_you_know* messages.

i_know and *update* messages convey information concerning one or more MIB modules. For each MIB module, the following information is included:

- The source server of the MIB module. This server will also be called *authoritative* server. Note that multiple servers may claim to be the authoritative server (Section 4.2 discusses how to handle in that situation).
- Timestamp, which denotes the local clock of the authoritative server at the moment the authoritative server distributed this information.
- MIB module name and numeric identity. This information can directly be derived from the MIB module definition.
- Last update of the MIB module. This information will be obtained from the “last updated” clause in the MIB module definition.
- OIDs and symbolic names of the top level node(s) within the module. Note that it is sufficient to send just the top level OID(s), and it is not necessary to send the complete lists of all OIDs

```
<i_know state_number = 1>
    <module name = IF-MIB id = 1.3.6.1.2.1.31 last-updated = 200006140000Z>
        <source>www.simpleweb.org/ietf/mibs/oidd/</source>
        <timestamp>200104050000Z</timestamp>
        <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>
        <timeout> 2592000</timeout>
    </module>
    <module name = ...>
        ...
    </module>
    ...
</i_know>
```

Figure 2: Example of an i_know message

defined by the MIB module. This is comparable to the *top-level registration* mechanism defined by the AgentX protocol [11].
- Time-out value of this module. This value indicates how long other servers may cache this information. Note that new messages (which have a later timestamp) may override this value.

Administrative messages are transferred over the HTTP protocol and are XML encoded. Figure 2 shows an example of the encoding of an *i_know* message.

Since the size of *i_know* messages may be considerable, servers may also inform “friends” of recent changes by sending *update* messages. As opposed to *i_know* messages which include complete information, *update* messages include just the information differences (delta). To maintain synchronization, every *i_know* and every *update* message includes a *<state_number>* (see the first line of Figure 2). Each server maintains his own *<state_number>*, and increments this number whenever the state changes because of the addition or removal of a MIB module definition. As opposed to sequence numbers which are used in many other algorithms, subsequent messages may carry the same state number (provided the state did not change).

i_know message may be sent whenever a server boots up, or whenever a server is requested to forward the complete list of MIB modules that it is aware of. The second event is triggered by the reception of a *what_do_you_know* message; such message is usually sent by a server that boots up, or by a server that lost synchronization due to a lost HTTP message or some other malfunctioning.

4.2 Message processing

To get an understanding of how MIB module information is exchanged between friends, this section discusses how a server that received an *update* message processes that message. Because of paper size restrictions, processing of *i_know* and *what_do_you_know* messages will not be discussed; the interested reader is referred to the website of the Internet Next Generation project [12].

First the server checks the *<state_number>* contained within the message (see Figure 2), to determine if the server is still synchronised to his “friend”. If this is the case, the server removes

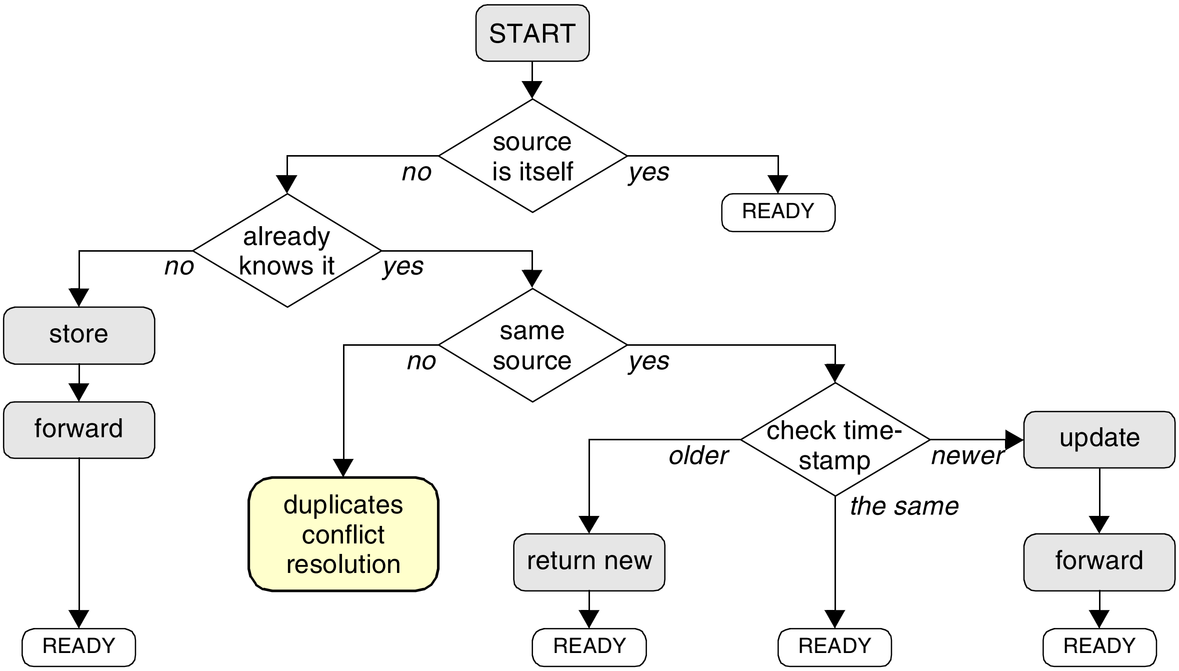

Figure 3: General overview of the update message processing

the outer wrapper of the message and inspects the information of each MIB module one by one. The processing of individual MIB modules is illustrated in Figure 3, and will be discussed below.

- The server checks the *<source>* field (see also Figure 2) of the received MIB module information, to determine if it is itself the source of this information. If that is the case, processing of this MIB module stops to avoid infinite looping of the information.
- If the information comes from another source, the server checks its database to determine if it has already an entry for this MIB module.
- If the database does not yet have an entry, the server may store the information in the database and queue it for later forwarding to other "friends". The decision whether or not to store and forward the information, depends on local policies and may, for example, be based on the amount of free disk space or the "friend" relationships. Note that storing the information that is contained within *update* messages is not the same as storing the actual MIB module definitions; if the server wants to store the actual MIB modules, it should use the normal data messages that were discussed in Section 3.
- If the server's database contains already an entry for this MIB module, it checks whether the *<source>* within the newly arrived message is the same as the one in the database. If they are not the same, multiple servers claim to be the authoritative server and further checking becomes necessary (see Figure 4).
- In case the *<source>* within the newly arrived message is the same as that in the database, the server checks the *<timestamp>* within the message to determine of the received information is more recent than that within the database. To facilitate this check, the server stores for each module the timestamp as set by the authoritative server.
  - if the *<timestamp>* within the newly arrived message is earlier than that within the database, the server notifies the friend that sent this outdated information about the existence of newer information concerning this module;
  - if the *<timestamp>* is the same, the information need not further be processed;
  - if the *<timestamp>* within the newly arrived message is the newer than that within the database, the information within the MIB module has to be updated and queued for later forwarding.

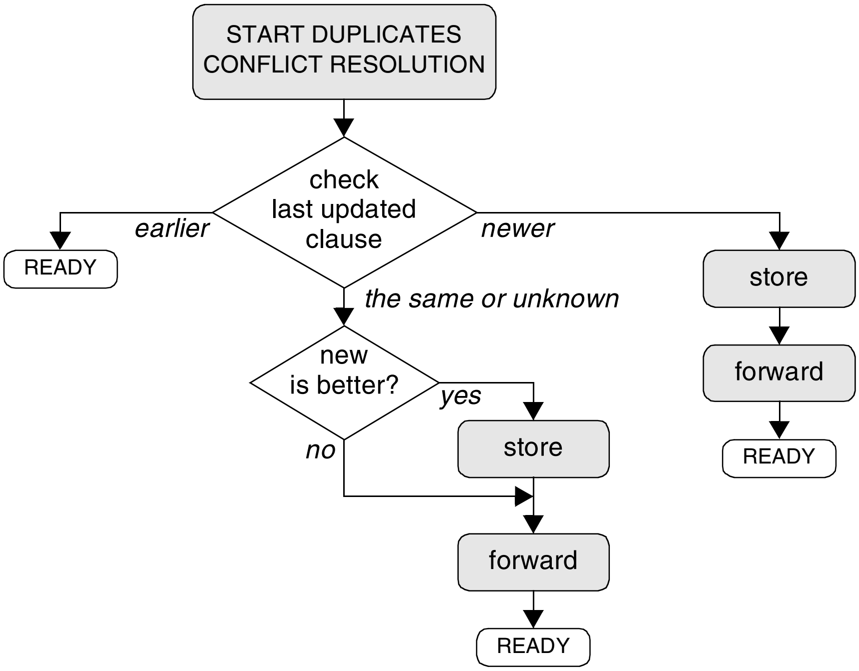

Figure 4: Duplicates conflict resolution

Figure 4 shows how a server can resolve cases in which multiple sources claim to be the authoritative server. First the server compares the *<last-updated>* field within the newly arrived message to the information contained within the database. The purpose of this comparison is to determine if the advertised MIB module is older, newer or the same as the one in the database. In case it is newer, the server may store the information in its database and forward it to other "friends". If the module is older, processing stops to avoid the propagation of a presumably outdated information. If the advertised module has the same age as the one in the database, further processing depends on the server's pre-configured policies. If one of the authoritative servers is his friend, the server may decide to favour that information. A policy may also be to store all information, in an attempt to improve the resilience of the look-up service. Note that SMIv1 [13] MIB modules do not contain *last-updated* clauses, which means that it is not possible to determine the "age" of such modules.

4.3 Usage scenarios

This section discusses some usage scenarios for normal as well as error cases. The cases will be illustrated by means of time-sequence diagrams.

One of the cases that can occur during normal operation, is that a server boots up. This case is shown in Figure 5; server A boots up and sends an *i_know* message to his "friend" B. This message includes a *<state_number>* (A1), and the list of MIB modules server A is aware of (mod1 and mod2). Server A knows that it has just booted up and may therefore be out of sync with server B; it issues therefore a *what_do_you_know* message. After reception of that message, server B responds with an *i_know* message. This response includes the state of server B (B6), as well as the MIB modules B is aware of (mod1 and mod4). In this specific example server A did not know about mod4; under the assumption that the definition of mod4 is stored on server C, server A may now issue a normal data command (see Section 3) to fetch that module from C. Note that server C need not be a "friend" of A.

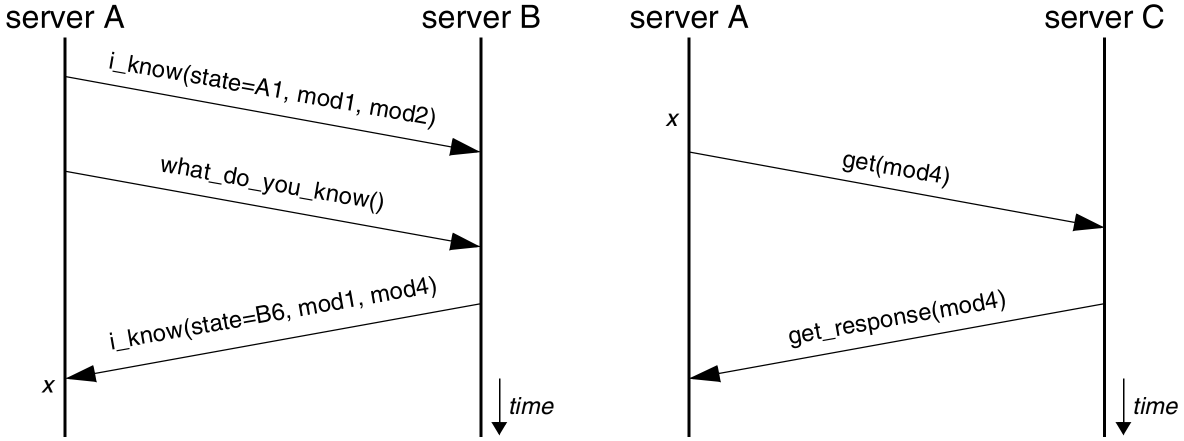

Figure 5: Server boots up

Error situations can occur due to the malfunctioning of the underlying transport channel, or due to the malfunctioning of servers. Figure 6 shows an example of a possible error situation. Server A sends an *update* message to his "friend", server B, with a *<state_number>* A32. Assume that the *update* message gets lost in the network. Also assume that shortly afterwards server A learns about two new MIB modules: mod5 and mod6. Server A increments his state_number, and issues a new *update* message, carrying this new *<state_number>* (A33). Server B receives this *update* message, and detects that the *<state_number>* is higher that the expected one: B would allow the value it already has (A31), or one higher (A32). Server B therefore concludes that it is out of sync with A, and asks server A for the complete list of all MIB modules A is aware of. For this purpose, B uses the *what_do_you_know* administrative message. Server A answers with an *i_know* message, including all the MIB modules it knows about and with a *<state_number>* equal to the one of the previous update message (A33).

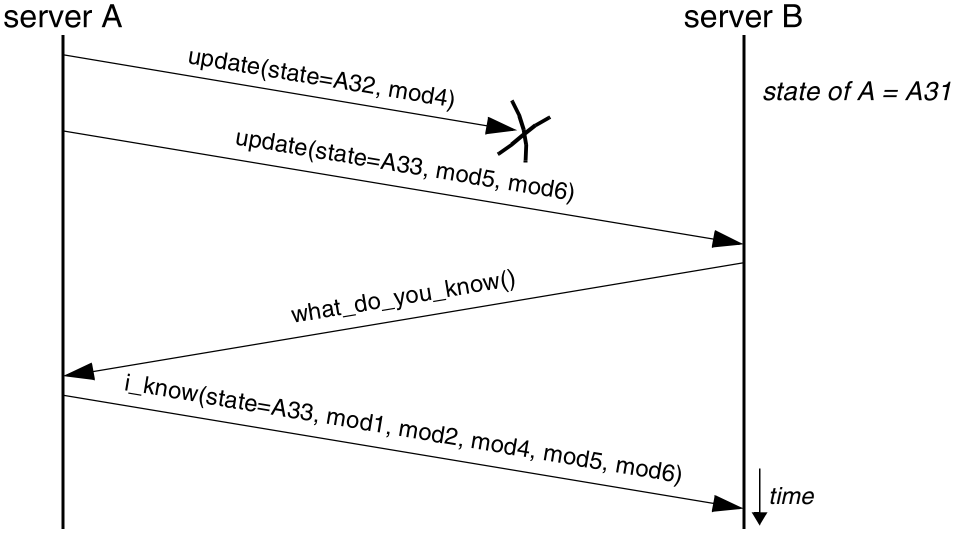

Figure 6: Server B loses synchronisation

5 Conclusions and further work

This paper discussed the design of a remote look-up service for MIB item definitions. Such service may be useful to solve one piece of the ‘meta-management’ problem. The solution presented in this paper allows the automatic retrieval of missing MIB module files, and is therefore a step forward compared to the current situation, in which human managers have to spend valuable time to solve that problem themselves.

The look-up service should be provided on a best effort basis, and introduced piecemeal. Initially the service can be provided by a single server; Section 3 discussed how human managers / management applications can access that server using HTTP. A prototype of such server has already been implemented and is running on the Simpleweb [6].
To improve performance, new servers may be added at later stages. Also vendors of network equipment may install servers; such servers may be limited in the sense that they can only distribute their own, vendor specific, MIB modules. It is expected that the total number of servers remains low; it is not envisaged that there will ever emerge a hierarchical directory structure like the DNS.

This work has been discussed within the IRTF Network Management Research Group (NMRG) [14], and will be documented into two internet drafts (one for the client-server interactions, the other for the server-server interactions). These internet drafts will include many details that could not be discussed within the context of this paper; the assignment of state numbers, for example, is tricky and also the procedures to recover from errors and deal with security require quite some text.
An interesting idea that was raised during the discussions within the IRTF NMRG, was to associate new material, which is not contained within current MIB module definitions, to the items defined in such modules. An example of such material could be japanese text for the description clauses of MIB items; such text will make it easier for japanese network managers to do their work. Another example of new material, could be pieces of text that explain how a certain MIB item relates to a certain Command Line Interface (CLI) object; such text would make it easier for managers who are familiar with CLI terminology to start using MIBs.

In the mean time work continues to develop client-side software for management applications like NetSNMP [15] and Scotty [16]. An interesting question is whether it will be possible to implement such software as a daemon that can interface via a common API to all management applications that run on the management system. Such daemon would remove the need to include in every management application the specific code that parses MIB module files.

Acknowledgements

This paper is based on the work performed within Work Unit 2 of the Internet Next Generation project [12]. This project is part of the dutch Gigaport programme, and sponsored by the Telematica Instituut (TI). The following organizations are member of the Internet Next Generation project: the Centre for Telematics and Information Technology (CTIT) of the University of Twente (UT), Ericsson EuroLab Netherlands BV (ELN), KPN Research and the TI. We would like to thank all members of the project, as well as the members of IRTF Network Management Research Group (NMRG), with whom we discussed our ideas and who provided valuable feedback.

References

[1] K. McCloghrie, M. Rose: RFC 1066 - “Management Information Base for Network Management of TCP/IP-based internets”, August 1988

[2] K. McCloghrie, M. Rose: RFC 1213 - “Management Information Base for Network Management of TCP/IP-based internets: MIB-II”, March 1991

[3] mibCentral: http://www.mibcentral.com/

[4] IANA - Private Enterprise Numbers: ftp://ftp.isi.edu/in-notes/iana/assignments/enterprise-numbers

[5] The Simpleweb: http://www.simpleweb.org/

[6] The SimpleMIB look-up service: http://www.simpleweb.org/ietf/mibs/oidd/

[7] K. McCloghrie, D. Perkins, J. Schönwälder: RFC 2578 - “Structure of Management Information Version 2 (SMIv2)”, April 1999

[8] K. McCloghrie, F. Kastenholz: RFC 2863 - “The Interfaces Group MIB”, June 2000

[9] Libsmi project page: http://www.ibr.cs.tu-bs.de/projects/libsmi/

[10] R. Perlman: Interconnections: Bridges, Routers, Switches, and Internetworking Protocols, Addison Wesley Professional Computing Series, ISBN: 0201634481, 1999

[11] M. Daniele, B. Wijnen, M. Ellison, D. Francisco: RFC 2741 - “Agent Extensibility (AgentX) Protocol”, january 2000

[12] Internet Next Generation project: http://ing.ctit.utwente.nl/WU2/

[13] M. Rose, K. McCloghrie: RFC 1155 - “Structure and Identification of Management Information for TCP/IP-based Internets”, May 1990

[14] IRTF Network Management Research Group (NMRG): http://www.ibr.cs.tu-bs.de/projects/nmrg/

[15] NetSNMP project page: http://net-snmp.sourceforge.net/

[16] Scotty project page: http://wwwhome.cs.utwente.nl/~schoenw/scotty/