The Simple Times™

THE QUARTERLY NEWSLETTER OF SNMP TECHNOLOGY, COMMENT, AND EVENTS
VOLUME 7, NUMBER 1 MARCH, 1999

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

In this Issue:

The Simple Times is available as an online journal in HTML, PDF and PostScript. New issues are announced via an electronic mailing list. For information on subscriptions, see page 22.

Bulk Transfers of MIB Data

Ron Sprenkels, University of Twente
Jean-Philippe Martin-Flatin, EPFL

Since the original days of SNMP back in early 1988, the requirements for managing IP-based networks like the Internet have changed considerably. An important change is that the total amount of management information that needs to be transferred has increased greatly. Not only did the size of traditional MIB data grow, for example IP routing tables and TCP connection tables, but also new types of management information appeared, for instance accounting tables, which tend to be bulky. The widely deployed SNMP version 1 was not designed for transferring large amounts of data. The overall latency of such transfers can be quite high and the way in which the SNMP messages are encoded for transmission over the network is not particularly efficient. The new version 3 of the SNMP protocol, while improving on other issues like security and access control, does not improve the transfer of large amounts of MIB data sufficiently, even though it provides a `get-bulk` operation.

In this article, we look into ways of making bulk transfers of MIB data between SNMP agents and managers more efficient. We consider a bulk transfer to be the transfer of several hundreds of kilobytes of MIB data in a single logical transaction. For bulk transfers, our objectives are:

- to reduce the end-to-end latency (i.e., the total time to transfer a set of management data between an agent and a manager, including marshalling, unmarshalling and network transfer);
- to reduce the network overhead (i.e., the ratio between the amount of bytes transferred over the network and the actual management information); and
- to improve the retrieval of SNMP MIB tables (by both reducing latency and network overhead for the particular case of table retrieval).

These objectives share a common goal: to improve the scalability of network management in the IP world.

Improving scalability has become necessary because both the number of systems to be managed, as well as the amount of management information per system has increased.

This article is structured as follows. First we discuss what we consider the three main problems with bulk transfers: latency, network overhead and table retrieval. Next we discuss three different approaches to solve these problems. The first approach aims to be a small evolutionary change to the current SNMPv3 framework, requiring minimal changes to existing SNMP manager and agent implementations. As such, this approach is envisaged to be useful in the short term. The second approach uses a mixture of SNMP and other protocols. The third approach discusses alternative protocols and encodings, abandoning the SNMP protocol and associated BER encoding altogether. This approach will therefore take longer to design, implement and deploy, and is envisaged to be useful in the longer term. This approach also serves as food for thought, and is intended to solicit discussion on future Internet management frameworks and protocols.

Problem #1: Latency

Currently, retrieving large amounts of MIB data involves a high number of PDU exchanges over the network. When using the `get-next` operator, the retrieval of large tables with many rows requires at least one `get-next` operation per table row. If a table row does not fit into a single message (due to message size constraints) even more operations per row are needed.

RFC 1187 describes an algorithm that speeds up the retrieval of an entire table by using multiple threads in parallel where each thread retrieves only a portion of the table. To make this work, one needs a manager which supports multiple threads and which has knowledge about the distribution of instance identifiers in the table. Note that the algorithm does not reduce the total number of request/response PDU exchanges. Instead, it is more efficient in terms of latency because several threads gather data simultaneously. The price for achieving reduced latency with multiple threads is bursty SNMP traffic, which can cause overload problems on the agent side.

If the algorithm described in RFC 1187 is not used, then each `get-next` operation must be completely finished before the next one can start. Things get worse in case packets get dropped within the network, since retransmission timers have to expire and retransmissions must succeed before the retrieval process can continue.

The situation improves with the introduction of the `get-bulk` operator. However, the response to a single `get-bulk` operation still has to fit into a single UDP packet. In theory, UDP can handle packets of nearly 64 KBytes. In practice, the maximum packet size will be much smaller. For the hundreds of kilobytes of MIB data we are considering here, even the use of `get-bulk` results in a large overall delay.

Each request/response exchange (be it `get-next` or `get-bulk`) involves at least a network round trip delay time, possibly time-out and retransmission delays, and probably also other protocol stack overhead delays (e.g. marshalling and unmarshalling of data, context switching). In summary, the overall latency of a bulk transfer is high because of the large number of PDU exchanges involved and their synchronous nature.

Problem #2: Network Overhead

Network overhead is the proportion of the network bandwidth used for management which is thus unavailable for the transport of user data. Particularly in the case of bulk transfers, which deal already with large amounts of data, it is import to keep that overhead low. All currently active SNMP frameworks (SNMPv1, SNMPv2c and SNMPv3) are inefficient in terms of the number of bytes needed to transfer MIB data over the network. We identified three causes for this inefficiency: the Basic Encoding Rules (BER), the OID naming scheme and what we will call the "`get-bulk` overshoot" problem.

BER encoding is well known to be fairly inefficient in terms of network overhead. Mitra [1] and Neufeld and Vuong [2] describe this issue in detail. At the time BER was chosen for SNMP, network overhead was not considered to be a main issue; the reason BER was selected was because it was readily available and simple to implement. Since alternative encoding rules exist nowadays, it is feasible to reduce network overhead by selecting another set of encoding rules. These new rules, however, should not increase latency too much due to additional encoding/decoding times.

If we look at the OIDs of the objects involved in a bulk transfer, we observe a high degree of redundancy. For the objects in a table, we see multiple occurrences of identical portions of OIDs: all OID prefixes up to the column number are identical, as are the instance identifier postfixes of all entries of a single table row. Because of this, redundant information is transferred, resulting in a higher network overhead than strictly needed.

The `get-bulk` operator also adds to the network overhead, since the manager, which does not know the size of the table to be retrieved, has to guess a value for the `max-repetitions` parameter. Using small values for `max-repetitions` may result in too many PDU ex-

changes. Using large values, however, may result in an "overshoot" effect: the agent returns data that does not belong to the table the manager is interested in. This data will be sent over the network back to the manager, just to be discarded.

Problem #3: Table Retrieval

The problems with table retrieval discussed here are holes in tables, table consistency and `get-bulk` overshoot consequences.

Retrieving table objects is more complex than retrieving other objects. This is due to the fact that the SNMP frameworks have no notion of tables, but only of conceptual tables. The difference between these two concepts is important for tables that have rows in which some columnar objects do not exist; in other words, for tables that allow their row entries to have "holes" in them. Consider the case where a manager wants to retrieve a table by performing repeated `get-next` operations. In most cases the manager uses for each row of the table a single `get-next` operation. The `get-next` PDU contains a list of OIDs, one for each column of the row; the value of these OIDs is usually taken from the response of the previous `get-next` operation. If there is a hole in the table, the `get-next` operation returns the elements of the next row, except for the column in which there is a hole. For this column the `get-next` operation returns the next available object in the MIB tree, which is the columnar object for the next table row that does have a value in that column (we will not discuss what happens if none of the remaining table rows has a value in that column). As a consequence, the manager is faced with a set of columnar objects that do not belong to the same row anymore; it has the cumbersome task of finding out what objects belong to which rows and where the holes are. In short, reconstructing the actual table, including determining where the holes in the table are and where the table ends, is a time-consuming task.

Another problem is that the manager has no guarantee that it will retrieve a table in a consistent state. This is particularly true for large tables, because the retrieval of such tables involves a large number of PDU exchanges, which take a considerable amount of time. If in the meantime some table elements are changed by the agent, the manager ends up with an inconsistent view of the table.

Finally there is a problem which we call "`get-bulk` overshoot." When `get-bulk` is used to retrieve a table, object values may be returned that do not belong to the table of interest. If, for example, a `max-repetitions` value of 50 is used, and the table contains only 10 additional elements, 40 elements will be returned that are not really needed. In this case, the agent processed information, retrieved object values from the instrumentation and used resources, just to have the manager discard the information. This can add up to quite an amount of wasted resources.

Now that we have outlined some problems with bulk MIB data transfers, we will discuss three approaches to solving them.

Approach #1: Extending SNMP

In the first approach, we seek to make small evolutionary additions to the SNMP frameworks. As a result, any changes or additions to the current protocol should be easy to implement with limited changes to existing implementations, thus protecting the investment in current implementations of the frameworks.

An Additional Transport: SNMP over TCP

Adding an SNMP over TCP transport mapping (in addition to the preferred transport mapping over UDP) is probably the most lightweight addition that can be made to the SNMP frameworks, and it already solves some of the problems outlined earlier. The immediate effect of moving from UDP to TCP is that the UDP limitation of the maximum SNMP message size of nearly 64 KBytes or less disappears. TCP has a window mechanism that allows several chunks of data to be in transit in parallel. This removes the cause for additional round trip time latencies when a table row does not fit into a single UDP message, or when the requested number of repetitions for a `get-bulk` request does not fit into a single UDP packet. As a result, overall latency will decrease and table consistency will improve. A downside is that large buffers are required on both the manager and the agent to store the large SNMP messages. This can be a serious problem for agents in embedded environments.

Several issues should be investigated:

- To prevent the need for large buffers, a scheme could be defined where many related smaller messages are sent over the same TCP connection.
- As a result of using TCP as a transport, both agents and managers get a task they did not have before: managing their TCP connections. Possible strategies are to close each TCP connection immediately after use, to keep and manage a pool of established TCP connections, or to close TCP connections after some period of inactivity.
- SNMP manager and agent implementors should have the option to decide on a per-operation basis whether to use UDP or TCP. When a manager expects to retrieve little data from a large set of

agents, UDP is the best choice. Conversely, when large tables or MIB subtrees are expected to be retrieved from a small set of agents, TCP is a better choice.

In the early days of SNMP, when a well-known UDP port was reserved for SNMP (161), the same TCP port number was also allocated to SNMP. As a result, the reservation of a well-known port for SNMP over TCP is not an issue.

In 1994, the University of Twente had (temporarily) a prototype of SNMPv2p running over TCP. Recently, Schönwälder and Deri modified the Linux CMU SNMP library and the UCD-SNMP software to transport SNMP traffic over TCP. These experiments suggest that the extension of an existing SNMP implementation to support TCP should be relatively straightforward.

New Encoding Rules and/or Compression

An issue that affects both latency and network overhead is the way the management information is encoded for transmission over the network. We have two different ways (that can be used in conjunction) to reduce the network overhead with respect to plain BER: replacing BER by a different set of encoding rules and adding compression.

There are two different types of encoding schemes: schemes that use a definite form for the length field and schemes that use an indefinite form. Definite-form schemes require the whole of the message to be in a buffer, because they insert the length of the message in front of the message. Indefinite-form schemes do not put a length field in front of a message. Instead, they mark the end of an encoded ASN.1 element by a special byte. Hence, an indefinite-form scheme does not require the complete message to be buffered and it can encode on the fly.

We first note that all versions of SNMP mandate the use of the definite form of BER. Replacing BER by a different encoding scheme therefore requires a new protocol version and is thus a major change.

The ISO has defined several alternatives to BER. PER encoding (Packed Encoding Rules) has approximately 30% shorter encodings, at the expense of a small increase in encoding time. PER allows the use of the indefinite form, so no large encoding buffer is needed. Lightweight Encoding Rules (LER) decrease overall latency by ensuring quick encoding and decoding. However, network overhead is adversely affected, because the encodings can be much longer than those generated by BER. Distinguished Encoding Rules (DER) use the definite form only. They slightly improve encoding time over BER while having a minimal impact on network overhead compared to BER. Finally, Canonical Encoding Rules (CER) use the indefinite form like PER, but are less demanding in terms of encoding time.

We initially thought we should move from a definite-form encoding scheme to an indefinite-form encoding scheme in order to avoid large buffers. We later realized that the SNMP version 3 (SNMPv3) message header can include an authentication digest, which is computed over the whole PDU. As a result, we must buffer the entire PDU before transmitting it anyway if authentication is used. Therefore, switching to alternate encoding rules does not really prove advantageous over BER in the general case.

SNMPv3 allows to add encryption envelopes to SNMP messages. This feature can not only be useful for its intended purpose, which is encryption, but it can also be exploited to achieve data compression. By adding an encryption algorithm that in fact compresses the message, the size of the messages that are transmitted over the wire decreases. Defining compression as an encryption algorithm allows to add compression to SNMPv3 without making any changes to the protocol. However, since there is no `noAuthPriv` security level in SNMPv3, one has to use authentication in order to take advantage of compression.

Using compression relieves us of the need to abandon BER, in order to replace it with a new more efficient encoding scheme. It leaves the installed base of implemented and debugged BER encoding and decoding software in place. Any standard compression algorithm can be used like e.g. DEFLATE (RFC 1951), for which stable, debugged implementations are readily available.

Additional Protocol Operation: get-subtree

We advocate introducing a new SNMP protocol operation to be used for the retrieval of complete MIB sub-trees, since neither `get-next` nor `get-bulk` are efficient for that purpose. Note that retrieving an entire table, an entire table column or a part of a table column are all special cases of a MIB subtree. We define the `get-subtree` operation to retrieve all objects below a particular node in the MIB tree. By allowing the operation not only to retrieve a single subtree, but also to retrieve multiple subtrees with a varbind list, the operation becomes even more powerful. It can then be used to retrieve selected columns of a complete table or selected columns within a range of rows of a column.

Examples of the usage of this operation include retrieving the entire interface table (ifTable), retrieving the operational status of all interfaces in the ifTable, retrieving both the operational and the administrative statuses of all interfaces in the ifTable, and retrieving the state and remote address of all TCP connections to a particular local address/port combination. For each of these examples, the information is requested in a single

protocol operation.

The amount of data returned for a single `get-subtree` operation can be quite large; this has two implications. First, the `get-subtree` operation will be most useful when used over TCP. The strict message size limitations of the UDP transport would immediately break the advantages of this new operation. Second, even when using TCP as a transport, it will generally not be feasible for agents to have memory buffers to store huge response message. Further, since it might take some time for the agent to collect the MIB data, other requests may have to wait some time before a single-threaded agent will process them. Therefore, a mechanism is needed that allows the agent to return multiple related `response` messages for a single `get-subtree` request. The TCP transport will take care of any required retransmissions and it will keep the responses in order. The TCP transport will also provide a window that allows multiple responses to be in transit concurrently.

In summary, the main advantages of the `get-subtree` protocol operation are:

- A table can be retrieved using a single protocol operation. As a result, latency can be minimal: a single round trip time for the protocol operation, plus what is added by the TCP transport for connection setup, segmentation, retransmissions, etc.
- The agent implementation collects from its instrumentation only the values of the requested objects; there is no `get-bulk` overshoot anymore.
- The absence of `get-bulk` overshoot also means that no network overhead is generated for objects that are not of interest anyway.
- The single operation property gives the agent the best chance of returning a consistent view of the table to the manager, much better than with a large number of separate `get-next` or `get-bulk` operations.
- It is no longer necessary to guess `max-repetitions` values.
- The `get-subtree` operation can be an extension to both SNMPv1 and SNMPv3. No new message format is needed, only a new PDU type. This qualifies `get-subtree` as a relatively small evolutionary step.

An agent implementation of `get-subtree` collects and returns each of the subtrees specified in its varbind list simultaneously, that is, row by row for a table. This ensures an efficient retrieval of table rows from the instrumentation and it minimizes the risk of getting inconsistencies within a single row.

The problem with holes in tables discussed previously still exists. The reconstruction of the conceptual table remains the task of the manager. Only the retrieval and transport over the network is greatly simplified by this new protocol operation.

Approach #2: Hybrid Solutions

In this approach we present a solution which uses a combination of SNMP and other protocols.

Bulk File MIB and FTP Client MIB

Stewart proposed to solve the bulk transfers problem by using SNMP together with the File Transfer Protocol (FTP). His proposal consists of two MIB modules. The first MIB module (`CISCO-BULK-FILE-MIB`) specifies how an SNMP agent stores a user-defined set of MIB data into a local file. The second MIB module (`CISCO-FTP-CLIENT-MIB`) can be used to upload local files to an FTP server using the FTP protocol. An SNMP agent implementing both MIB modules can be instructed to save a specified (large) amount of local MIB data into a file and upload that file to a particular FTP server. We will now describe these MIB modules briefly.

The `CISCO-BULK-FILE-MIB` defines three tables. The `cbfDefineFileTable` defines the name of the file, how it is stored and what encoding format will be used. One or more entries in the `cbfDefineObjectTable` are associated to a row in the `cbfDefineFileTable`. The entries specify what local MIB objects should be put in the file upon creation. A complete MIB table can be specified in a single entry in the `cbfDefineObjectTable`. A manager initiates the creation of the actual file by doing a `set` operation on the `cbfDefineFileNow` object. This results in a new entry in the `cbfStatusFileTable` which keeps track of the progress of the file creation.

The storage type of a bulk file can either be permanent, volatile or ephemeral, where the latter indicates that data exists only in small amounts until it is read. This storage type, when used in combination with the `CISCO-FTP-CLIENT-MIB`, prevents the need for a buffer large enough to hold the complete file.

There are three options for the format of the data files: BER encoded, binary and human-readable ASCII. The BER encoded format is identical to an SNMP varbind list. The binary format consists of tags and data fields. There is a tag to set a standard OID prefix, a tag for a single object, and some tags to encode tables. Tables are encoded with little OID redundancy: for each entire row only the common instance portion of all the OIDs in that row is encoded. The binary format uses a proprietary encoding scheme for the ASN.1 primitive types `INTEGER`, `OCTET STRING` and `OBJECT IDENTIFIER`. The ASCII format is a mechanical translation of the

binary format; translation rules for tags and values to ASCII are given in the MIB specification.

The `CISCO-FTP-CLIENT-MIB` has a single table. An entry in the `cfcRequestTable` table specifies a local file that is to be uploaded to a specified FTP server, either in binary or in ASCII mode, using a specified user name and password. A manager can initiate a file transfer from the agent to an FTP server by setting the `cfcRequestEntryStatus` to `active`. The progress and result of the transfer can be monitored by reading the `cfcRequestOperationState` and `cfcRequestResult` objects. The manager can abort an ongoing transfer by setting the `cfcRequestStop` object.

In summary, this solution to the bulk transfer problem requires agents to implement two MIBs and the manager to configure entries in several MIB tables to initiate and control bulk transfers. This means that bulk transfers are treated totally different from normal accesses to MIB data. For this reason, security needs to be considered separately for these transfers. For example, there is no mechanism in place which authenticates or encrypts management data while in transit over the network. The hybrid solution described here also requires an FTP server on the manager side. This means that management data retrieved via a bulk transfer is processed very differently from management data retrieved via SNMP since it becomes available in a file on an FTP server.

Approach #3: Other Protocols and/or Encodings

The most important value of the SNMP management frameworks lies in the large amount of existing MIB specifications. These are not only the standard MIB specifications developed within IETF working groups, but also proprietary, vendor-specific MIB specifications. The MIB specifications are valuable because they represent detailed knowledge of what is relevant management information for a large variety of networking devices, protocols, network elements, transmission medias and so forth. This value exists regardless of the protocol used to move information around or the way the information is encoded while in transit. Therefore, both the SNMP protocol and the encoding rules can be replaced by something else. Some possibilities for such replacements and their properties are examined in this section.

MIME

From a software engineering point of view, management data is just another example of structured data. The Internet community has a standard way for transferring structured data called MIME, which is essentially a "bag and tag" scheme to transfer data. In order to transfer management data using MIME, we need to define a MIME type for it (the tag) and we need to define how data of that type is structured inside the bag. Because of the overhead it introduces, MIME is only suited for transferring bulk data. Furthermore, by means of a transfer encoding, MIME allows content to be transparently compressed while in transit. We will now discuss three options for the MIME type and its encoding.

- Option #1 is to define a new MIME tag for putting a BER-encoded SNMP message, which is normally sent over UDP, inside the MIME body. Using the multi-part feature, multiple related SNMP messages can be put into a single MIME envelope.

- Option #2 is to define a new MIME tag for putting management data encoded in ASCII into the MIME body. An ASCII representation for each SNMP protocol operation and for all SMI data types must be defined. Using an ASCII encoding scheme has the advantage that it is usually easier for programmers to read, understand and process data in a human-readable format.

- Option #3 is to represent SNMP protocol operations and SMI data types using an XML Document Type Definition (DTD). If XML gets good acceptance in industry, we will see many programmers with experience of using XML. Encoding management data in XML means that many programmers will have the knowledge and skills required to build management applications. This is a big advantage compared to the current situation where programmers need to have at least some basic knowledge about ASN.1 and BER.

Wrapping management data in MIME types has the advantage that several existing protocols can be used to move MIB data around. This includes SMTP (a store and forward protocol) or HTTP (a request/response protocol). We will look a bit closer at HTTP now.

HTTP

The well-known HTTP protocol is a good candidate for transferring MIME-encapsulated management data. We identified three reasons for that. First, the protocol was designed to transfer MIME data. Second, there currently is a clear trend in industry to embed HTTP servers for management purposes in networking equipment. So the chances of HTTP being accepted by industry as a protocol for management seem good. Third, HTTP is based on TCP, so it is suitable for the transfer of bulk data, as we outlined earlier.

There are some downsides to using HTTP for management as well; we will name three. First there is the feature richness of HTTP. HTTP has numerous options and features that are valid and useful for its intended purpose, which is to be used as a document transfer protocol in the World-Wide Web. However, for the transfer of management data, many of those features will not be useful or usable. Conforming implementations of the protocol must include all of these features. As a result, the HTTP implementations in network devices will be needlessly big and complex.

Second, since the development and standardization of HTTP will remain focused on its original purpose, future versions of the protocol might have characteristics that are unwanted for a management protocol. Also, for the same reason, it will probably be difficult to get new features that are desirable for the use as a management protocol into HTTP.

Finally, the security mechanisms proposed and used in conjunction with HTTP do not directly map to the security mechanisms defined in SNMPv3. This means that either some mappings need to be defined or that there will be different security mechanisms (authentication, privacy, access control) for accessing the same MIB data via SNMP or HTTP.

Conclusions

In this article, we looked at bulk transfers of MIB data. The current SNMP management frameworks are not very efficient for bulk transfers. The three main problems identified in this paper are latency, network overhead and table retrievals. We discussed three different approaches to speed up bulk transfers of MIB data.

We first looked at solutions within the SNMP framework. We believe that within the boundaries of the current SNMPv3 framework and with relatively little effort and small changes, the problems can be solved to a large extent. Latency can be significantly decreased by using TCP as a transport and by introducing a new `get-subtree` protocol operation. Network overhead can be decreased by compressing the payload of an SNMP message. Table retrieval can be improved by applying the new `get-subtree` operation to conceptual tables.

Second, there are possible hybrid solutions. We presented a solution proposed by Stewart. A downside to this solution might be that it treats bulk transfers as a separate, special issue, and still requires all of the normal SNMP framework and protocol stack to be in place. Furthermore, a whole new set of security problems will be the result of such an approach. Other hybrid solutions are probably also possible, but are not discussed in this article.

The third approach is to replace SNMP with another protocol. By using a protocol that runs over TCP, bulk transfer latency can remain low. By using compression on the encoded management information, network overhead can be kept low. If a mainstream technology is used for representing management information, e.g. XML, building management applications will no longer require skills specific to network management.

The first solution aims to be a small evolutionary step with respect to the current SNMPv3 management framework. It is relatively easy to implement and keeps the current implementations largely intact. This protects investments in current SNMP technology. The second solution is probably also fairly easy to implement, but has some architectural and security-related downsides that make it in our view less attractive than the first one. The third solution is not covered in as much detail as the first two. It will take quite some work to further define that solution. Because it breaks so radically with the current SNMP framework it will be more difficult to get it implemented and deployed. As such, it is intended to serve as food for thought for the long-term future of Internet management.

Acknowledgments

The ideas presented in this article emerged during a two-day meeting that took place in November 1998 in Lausanne, Switzerland. The following people were involved:

- L. Deri, University of Pisa
- J.P. Martin-Flatin, EPFL
- A. Pras, University of Twente
- J. Schönwälder, Technical University Braunschweig
- R. Sprenkels, University of Twente
- B. Wijnen, IBM T.J. Watson Research

This meeting resulted in the creation of the Network Management Research Group (NMRG) of the Internet Research Task Force (IRTF). Contact information and additional documentation can be found on the NMRG Web page at http://www.ibr.cs.tu-bs.de/projects/nmrg/.

References

[1] N. Mitra, *Efficient Encoding Rules for ASN.1-Based Protocols*, AT&T Technical Journal, 73(3):80-93, 1994.

[2] G. Neufeld, S. Vuong, *An overview of ASN.1*, Computer Networks and ISDN Systems, 23:393-415, 1992.