In Proc. DSOM '95, 6th IFIP/IEEE Workshop on Distributed Systems: Operations and Management, Ottawa, Canada (Oct. 1995).

Use of Control Variables to Convey Protocol Semantics in SNMP

Ruchir Godura* and Adarshpal S. Sethi

Department of Computer and Information Sciences
University of Delaware, Newark, DE

Abstract

We propose a general mechanism for enhancing the power of the SNMP and SNMPv2 protocols that is backward compatible with both protocols. The mechanism uses a new group of control variables to convey the semantics of protocol operations in PDU exchanges between a manager and an agent. We describe in detail a new enhancement of the protocol using this mechanism that allows implementation of atomic operations without PDU size limitations.

1 Introduction

The Simple Network Management Protocol (SNMP) [1] was one of several proposals to develop a set of tools powerful enough to manage the increasingly complex computer networks that were created in the late 1980s. The exponential growth of the number of hosts on the Internet and the increasingly large size and interconnections of networks made it clear that a workable solution had to be found to manage the complexity. SNMP was initially created as a temporary short-term solution to manage TCP/IP based networks. But owing to its simplicity, which its original authors had very wisely ensured through the "Fundamental Axiom" [6], SNMP became a protocol in its own right, rather than a stop-gap measure. As with any evolving protocol, SNMP needed revisions and enhancements, the first of which came with the development of the RMON (Remote Monitoring) MIB for SNMP [8]. The RMON Management Information Base (MIB) was a major step in the development of SNMP, but its scope was limited to making SNMP viable in an Internet-based environment. There was no change made to the protocol itself.

The need to patch up certain deficiencies in the original version of SNMP led to a major overhaul exercise that started in late 1992, resulting in the development of the specifications for SNMP version 2 (SNMPv2) [2]. Although SNMPv2 made extensive additions to SNMP in terms of security features, the only new protocol operation added was `GETBULK`. There was extensive debate over the handling of creation and deletion of "conceptual" rows in MIB tables, and SNMPv2 ended by formalizing these operations in the form of textual conventions [4].

One of the original philosophies of SNMP was that it should remain a stateless protocol. Thus, an agent would process one request at a time, and have no need to carry over or "remember" information from a previous command PDU. This translated into the use of the User Datagram Protocol (UDP) for PDU transfer, and the definition of manager commands that would enable one-shot operation – i.e., there would be no *negotiation* between the manager and the agent. This

*Ruchir Godura is now with AT&T Bell Laboratories, Holmdel, NJ.

principle has served SNMP well, making it possible to write lean and efficient agents that do not hog the resources of the network entities they are managing.

However, the solution to the create-delete problem adopted by SNMPv2, namely the RowStatus textual convention (which itself is a refinement of the RMON "Polka") makes it necessary for the agents to exhibit stateful behavior. This happens because variables in a MIB row may need to be treated atomically, and some MIB tables have a width that may exceed the maximum size of SNMP PDUs. The size of SNMP PDUs is itself restricted by other factors such as the maximum size of a UDP datagram and small MTU (Maximum Transmission Unit) sizes used to limit IP fragmentation. If all the necessary variables required to create a MIB row cannot be fit into one single PDU, operating on only part of the row at a time could leave the row in an inconsistent state for an uncertain amount of time, as well as open it to a conflict with another manager trying to operate on the same row. Hence the need for a "stateful" and complex RowStatus textual convention.

The reason for this complexity is the lack of communicating power in SNMP and SNMPv2 between an agent and a manager. The only vocabulary available is `GET`, `SET`, `GETNEXT`, and `GETBULK`. Any logical operation that the manager wishes to accomplish has to be expressed as a sequence of these operations. In this paper we propose a general mechanism for enhancing the power of the protocol by adding new operations to it in a manner that is backward compatible with both SNMP and SNMPv2. This is done by using a new group of control variables that allows the protocol to exchange control information along with data between a manager and an agent. The method of control variables can be used to enhance the functionality of SNMP[1] in a variety of ways. It can be used to add operations for creating and deleting conceptual rows, to perform atomic operations by treating requests sent in multiple PDUs as a single operation, to add new operations for moving and copying tabular rows, to retrieve a complete table in a single operation, and even to implement functionality indentical to the `GETBULK` of SNMPv2 without adding a new PDU type. For reasons of lack of space, we only describe one of the above in this paper, namely performing atomic operations.

2 Using Control information

What is control information? It is simply information that is transmitted along with the data, and is meant to aid in the interpretation of the data. Currently, the control information is carried implicitly in the form of the PDU type or more indirectly through textual conventions. It is impractical to create a large set of PDU types, mainly because of the processing overhead required to encode/decode another PDU type, and also because introduction of new PDU types leads to incompatibility with existing versions of SNMP. Textual conventions are restrictive in that they are used to trigger certain functions in the agent, but there is no way to pass parameters to such functions. What we would like is a mechanism by which a manager could provide a list of variables and values to an agent and ask it to execute a particular operation using the values as arguments. If the agent had a procedure to carry out the operation, it would do so, otherwise it would respond to the manager's request in an intelligible manner.

In this paper we propose that a new class of MIB objects called *control* variables be created. These are conceptual objects in that they are not retrieved or set by a manager. They are defined in the MIB as objects, but their presence in an SNMP PDU (in the form of a variable-binding

[1]The techniques proposed in this paper are equally applicable to both SNMP and SNMPv2. For the sake of brevity, we will normally use SNMP to generically include both SNMP and SNMPv2.

pair) is meant to convey control information. Typically, the presence of a *control* variable name in a varbind list would indicate that a certain kind of operation needs to be performed on the "actual" varbinds, while the value associated with the *control* variable would indicate variations of the operation. Ensuring that control information is out of band is easy, because the agent has to simply determine which of the basic groups of objects it lies in. To make implementation easier, it is required that all *control* variables in a PDU occur before any "actual" variables.

Once it becomes possible for a manager to tell an agent what to do with a list of varbinds other than simply get or set it, a variety of operations can be implemented to enhance the functionality of the protocol. In particular, we describe a method that can implement atomic operations by overcoming PDU size limitations. This method uses control information to essentially implement a PDU reconstruction capability at the agent, to ensure that any command, no matter how long, could be presented to the SNMP protocol machine as a single unit. Thus, a long command could be broken down by the manager to conform to the PDU size limitations, transmitted over the network to the agent, and reconstructed by the agent to reproduce the entire command PDU *before* it started processing the variables. The control variables present in each PDU would provide sufficient information for the reconstruction of the original varbind list.

The advantage this provides is that, in designing other operations like row manipulations, set operations, etc., we could work with the assumption that all the data required for an operation can be supplied by the manager to the agent at once, instead of in bits and pieces. This would eliminate the necessity for complicated and sometimes opaque textual conventions like RowStatus. The details of how this is accomplished are presented in Section 4.

Control variables can also be seen as implementing virtual remote procedure calls from the manager to the agent. Parameters passed to the call are the value of the control variable itself, as well as the list of varbinds that follow the control variable. With such a powerful capability, it is easy to see how specific control variables can be defined to convey the semantics of `DELETE, CREATE, GETBULK, GETNEXT`, and any other esoteric operation a network manager might ever require.

It must be emphasized at this point that one of the more important features of this proposal is that it is backward compatible with the existing standards for *both* SNMPv1 *and* SNMPv2. Besides being compatible, it is easily integrated into an existing implementation of SNMP. This last part was verified by actually taking an implementation of SNMP (CMU-SNMPv2) and enhancing it to conform to our control variable specification.

What is being introduced is a *standardized* method of increasing the functionality of SNMP. Having a standardized approach to deal with future problems discourages the use of clever and opaque fixes by individual implementors. Though at times it may seem that efficiency is being compromised, we gain openness and interoperability.

3 The Method of Control Variables

This section describes how the MIB variables can be used to convey protocol semantics in manager–agent exchanges. The use of MIB variables to carry *meaning* rather than *data* necessitates the creation of another group of variables under the *mib-2* subtree of the set of defined objects. In addition to the standard object groups defined by RFC 1213 under MIB-II (e.g., **system**, **interfaces**, etc.) [5], we seek to introduce a new group of variables called the **Control Group**. This group is defined by making the following addition to the definition of MIB-II as given in RFC 1213 :

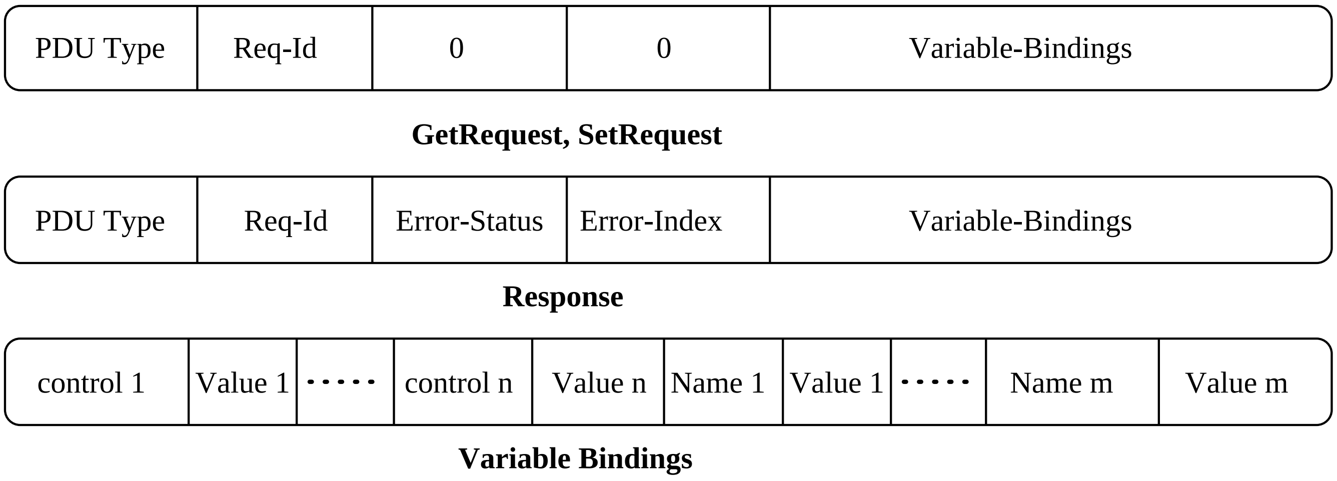


Figure 1: Structure of PDU with control variables

```
control      OBJECT IDENTIFIER ::= { mib-2 12 }
```

The variables defined under this group will be the means by which we shall attempt to incorporate more complex protocol operations into the framework of SNMP. We give below only one example of possible control variables that can be defined in this manner:

```
-- The CONTROL group
   continued OBJECT-TYPE
        SYNTAX INTEGER
        MAX-ACCESS read-write
        STATUS current
        DESCRIPTION
          " used for implementing continued
            variable lists for atomic operations"
        ::= { control 1 }
```

The control group of variables is not implemented as normal variables whose value can be read or set on some physical device. The presence of a control variable in an SNMP request PDU conveys to an agent the *protocol operation* that was desired by the manager. Currently, SNMPv2 allows only a certain set of operations, viz., `SET, GET, GETNEXT`, and `GETBULK` [3]. The agent distinguishes between PDUs containing different kinds of requests by recognizing the different PDU types that are used for different operations. For each operation type that a manager/agent wants to implement beyond the four standard operations listed above, a *control* variable would be defined in the MIB definition available to the agent and the manager. The variable *continued* defined above is used to spread a request that is too large to fit into one PDU over multiple PDUs, and have the request reassembled at the agent before actual execution of the request. This allows the multiple PDUs to be treated as a single operation that is executed atomically at the agent.

3.1 The semantics of control variables

An SNMP PDU may contain one or more *control* variables. However, they must occur at the beginning of the variable list that is contained in the PDU. Figure 1 shows the structure of a PDU with control variables in it. To keep the implementation simple, we impose the restriction that a control variable may not be preceded by any other MIB variable. A control variable may have an associated (usually integer) value that is encoded and transmitted along with the other varbinds in the PDU. The interpretation of the value of the control variable is specific to that particular operation, and is adequately described in the MIB description of the variable.

To carry out a protocol operation using control variables, the manager sends a PDU to the agent, with a list of varbinds, the first of which is the control variable defining the type of operation to be carried out. The PDU type may be a `GET` or a `SET`, depending upon the type of operation. Operations that require the MIB data to be modified would require a `SET` PDU, while operations that are of the retrieval type only would require a `GET` PDU type. In cases where the nature of the operation is not clearly specified in the textual description of the control variable, either of the two types could be used, as in the case of the *continued* variable. Each operation requires some information to be passed from the manager to the agent. The remaining variables in the request PDU comprise this information. The value of the control variable, usually defined to be an integer, may also be used to convey more specific parameters of the request.

Compatibility with existing SNMP implementations. In the event that a manager issues a control variable based request to an agent that does not implement either that variable or the entire group of control variables, the agent (rightly) generates a `noSuchObject` error for that variable. This response informs the manager that the agent does not implement that particular function, and it (the manager) must find some other way to carry out the request.

In the entire process of designing the manager-agent interaction using control variables, the greater part of the burden of memory and computing requirements is put on the manager as far as possible. The agent, keeping in mind the "Fundamental Axiom" [6], is kept as simple minded as possible. Indeed, allowing the manager-agent dialog to have a vocabulary of greater than four or five words gives us a lot of expressive power in designing efficient protocols.

4 Overcoming PDU size limitations in SNMP

One of the motivations behind this work was the problem of the limitation imposed on PDU sizes by the transport/network layer protocol used, most notably, UDP and IP. What is presented here is a method of overcoming this limitation using control variables. This method allows a manager to send multiple SNMP PDUs and have the agent treat them as a single atomic operation. Following the description of the protocol is an evaluation of its effectiveness and applicability.

4.1 The method of "continued" PDUs

The essence of the proposal is that protocol operations that have varbind lists that are too large to fit in one PDU, must be broken down into several PDUs by the manager, numbered, and sent to the agent, which reassembles the PDUs to form one single varbind list *before* processing the request. The following issues were kept in mind:

1. The structure of the PDU should remain the same.

2. Compliance should not be essential. Thus any agent/manager not wanting to implement this feature should still be interoperable with managers/agents that do support it.

3. Overheads required should be minimal.

Requirement 1 is immediately satisfied, since the use of control variables does not require the structure of the PDU to be changed. Requirement 2 is also satisfied since the nature of the control variables necessitates compatibility. Requirement 3 will be discussed at the end of the section.

4.2 Description of the protocol

Any agent or manager wishing to implement the capability to manipulate larger lists of varbinds than will fit into one PDU, atomically, must implement the *control.continued* variable in the MIB, as defined earlier.

When a manager has to break up a varbind list into more than one PDU, then the first varbind of each PDU must be the *control.continued* object type described above, along with an integer value that is computed as follows:
If the PDU is the i^{th} of m PDUs, then

```
control.continued = i * 1000 + m
```

The reason that this formula is used is that we need to pass a pair of integers with each PDU of the chain, (i, m), which indicates that the PDU is the i^{th} of the chain of m PDUs. i and m can be combined into one integer with the formula given above, which limits m to values of 999 and less. This integer number is encoded into the SNMP PDU as the value of the variable *control.continued*. At the agent end, the integer can easily be parsed to retrieve the values of i and m according to the formulas below:

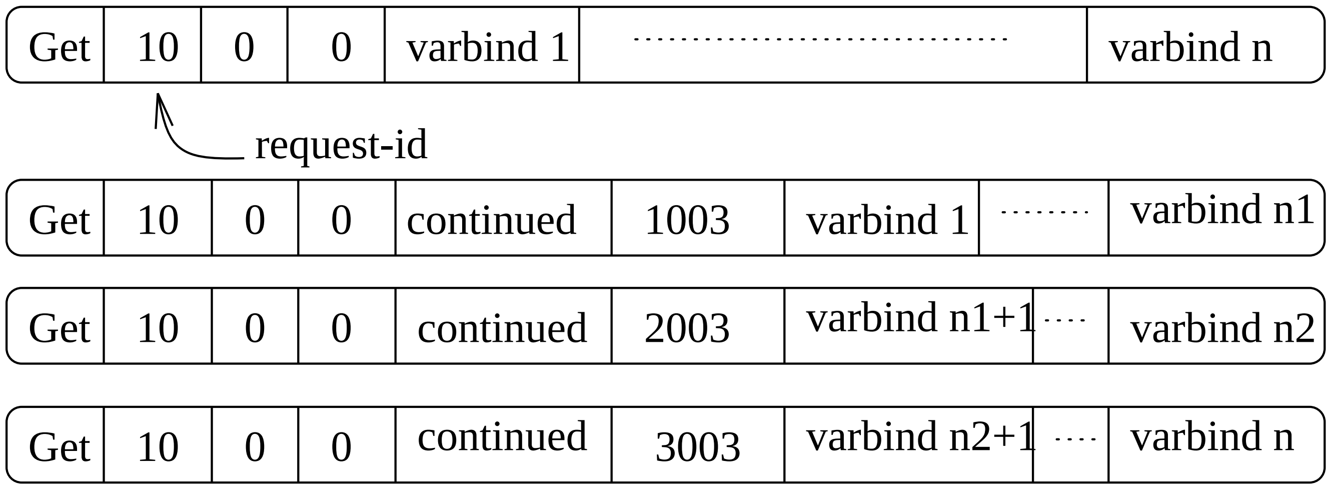
```
i = control.continued DIV 1000
m = control.continued MOD 1000
```

Besides, each of the chain of PDUs *must* have the same request-id. This is necessary not only because they are part of the same logical request, but also because the request id is the distinguishing factor that allows the agent to differentiate between "continued" PDU's having identical i and m values, and those that are part of different PDU chains. Figure 2 describes in greater detail the format of the PDU exchange between manager and agent using *control.continued* variables.

When an agent receives a PDU, it parses it and looks at the first varbind. If it is the *control.continued* variable, the agent stores the list of varbinds after parsing them, and returns an ack to the manager in the form of a PDU containing only the *control.continued* object. The PDU containing the ack has the same request-id as the request PDU. The agent stores incomplete PDU chains as lists of varbinds along with the request-id, PDU-type, and party information for the manager. As a new *continued* PDU is received, the agent checks to see if it is part of an already existing chain. If so, it is added to the chain. If it is not, then a new PDU chain is created. If the total number of simultaneously outstanding PDU chains for that party equals the maximum allowed (a value that is locally determined by the agent), then the oldest outstanding PDU chain is dropped. In the event that the new PDU is part of an already open chain, and the addition of this PDU completes the request, then the entire request is taken off the buffer storage and passed to the SNMP protocol machine to process.

The Method of Chained PDUs

The Following long request is broken down into separate PDUs by the manager :


Get | 10 | 0 | 0 | varbind 1 | .... | varbind n
request-id
Get | 10 | 0 | 0 | continued | 1003 | varbind 1 | .... | varbind n1
Get | 10 | 0 | 0 | continued | 2003 | varbind n1+1 | .... | varbind n2
Get | 10 | 0 | 0 | continued | 3003 | varbind n2+1 | .... | varbind n

The value of the *control.continued* variable is determined according to the formula :

value = i * 1000 + k

where i is the sequence number of the PDU and k is the total number.

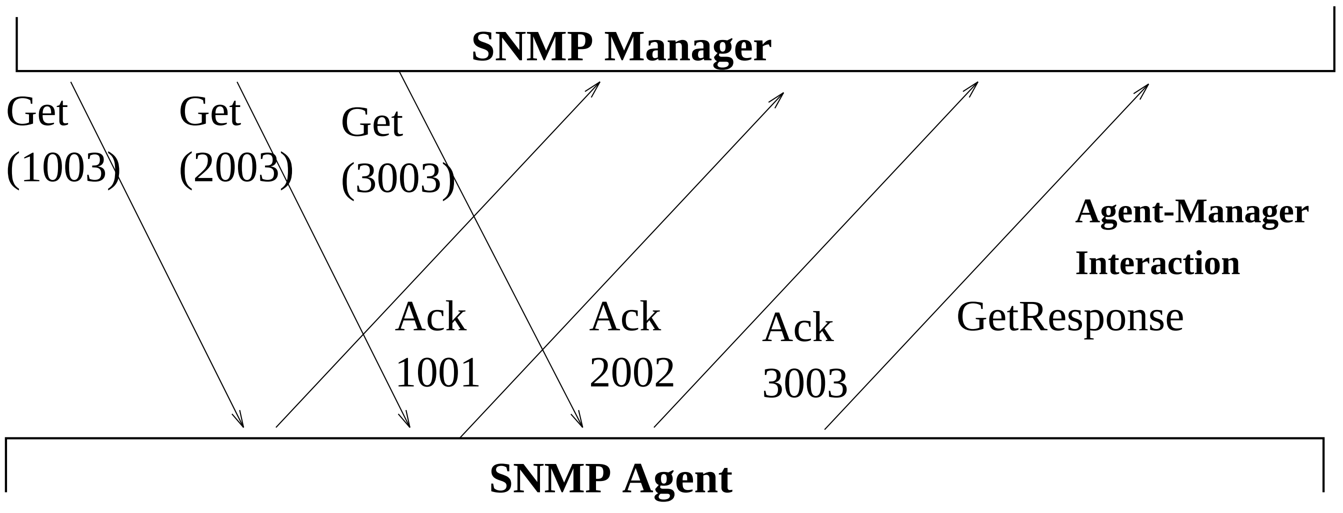


Figure 2: The method of chained PDUs

The value of the *control.continued* object in the ack also consists of two integers, encoded into one integer using the same formula described above. The first indicates the sequence number of the PDU just received, and the second indicates the total number of PDUs of that chain received so far. If the PDU does not contain a *control.continued* object as its first varbind, the varbind list is processed as one complete command. If the agent does not implement the *control.continued* object, it processes the PDU in the same way as it would process a complete request PDU, returning `noSuchObject` for the *control.continued* object. This would indicate to the manager that the agent does not implement the capability to handle longer varbind lists than will fit in one SNMP PDU.

The agent's response algorithm : Upon receipt of a PDU with the *control.continued* variable set to (i, m) (meaning that it is the i^{th} of m PDUs in a chain), and request-id I, the following actions are taken :

1. An ack for the i^{th} PDU is sent. This contains the continued object with the values (i, k), where k is the total number of distinct PDUs of that chain (or request-id I) that have been received by the agent so far. If this is the first PDU of the chain, then k is assigned the value 1, and a list structure is set up to hold the varbind lists of this new chain with request-id I. In doing so, if the maximum allowable number of "open" chains for that party is exceeded, then the oldest (timewise) chain is dropped.

2. The agent checks to see if the PDU (i, m) for request-id I has already been received. If so, then the current PDU is a retransmission from the manager, or due to duplication over the network. Whatever may be the reason, it is discarded. If PDU (i, m) has not yet been received, it is parsed to extract the variable-bindings and these are added to the list structure set up for the chain with request-id I.

3. The agent determines if the chain with request-id I is now complete. If so, it proceeds to the next step, otherwise it returns to waiting state for incoming PDUs.

4. The chain with request-id I now being complete, the agent removes the chain from its buffers, and starts processing the request using the varbind list. It builds the output packet, leaving space for a *control.continued* variable. If the processing is not over, and there is no more space left in the PDU, a *control.continued* variable is inserted, with a value equal to the sequence number of the response PDU. The first response PDU would get the value 1, the next 2, and so on. The last PDU in the response chain is indicated by a PDU with just the *control.continued* variable and an empty varbind list. If a response PDU is lost, it is the manager's responsibility to issue a `GET` request to retrieve the values of the variables in the missing PDU.
   Note: A `tooBig` error is *never* returned on a "continued" request. Also, the variables are processed in the order in which they were sent.

If the request was not a *continued* request (i.e., it did not have *control.continued* variables in it), then, if the response does not fit into one PDU, a response PDU with an error value of `tooBig` is returned (as is done by SNMPv2 standards). In the case that the request from the manager was a *continued* request, it is the manager's responsibility to keep track of the response PDUs it has received, and determine if any part of the response chain has been lost by scanning the variable list in the initial request that it sent out. In some cases like bulk retrieval (e.g., `GETBULK`), this job becomes a little more complicated, since the size of the response cannot be determined beforehand. Thus the manager cannot know how many PDUs or varbinds to expect in the response. In the case of a missing response PDU, the manager has sufficient information to frame another `GET` to retrieve the lost data. However, the last PDU in the response chain is explicit, and the response PDUs are all numbered, so missing response PDUs can be easily detected. Thus, a *continued* request cannot result in a `tooBig` error, and a non-*continued* request cannot result in a *continued* response. This is, of course, to maintain compatibility, so that a manager or agent that does not issue a *continued* request should not have to deal with a *continued* response.

Errors and retransmissions : The following unusual cases are handled:

1. PDUs arrive out of order: Each PDU carries in it sufficient information to calculate its place in the list. The agent sets up one list structure for each request id per party, and var-binds are linked into this list until all have arrived.

2. A PDU is lost: If a PDU is lost, the manager will not receive an ack for it, and may retransmit the relevant PDU. If, on receipt of this PDU, the list is complete, the request is processed.

3. The chain is never complete: A timeout mechanism in the agent deletes partial lists that have been idle for long. Besides, only a certain locally determined maximum number of "open" lists may be maintained by an agent for each party. If this number is exceeded, then the oldest one is dropped.

4. Multiple transmissions of PDUs: Only one copy of each link of the chain is kept by the agent in its list structure. However, each *continued* PDU received is replied to with an identical ack.

5. The agent runs out of memory, or, for some reason, is not able to handle the list: The agent drops the list, and returns an ack with the second integer value as a zero. Thus, if the i^{th} of k PDUs caused the agent to run out of memory, an ack, with its *control.continued* variable containing the encoding of $(i, 0)$, would be returned.

4.3 Evaluation of this protocol

- **Backward Compatibility:** The PDU structure remains the same. Implementation of the *control.continued* variable is fully optional, and normal conversation can take place between a manager and an agent with either one or both of them not having implemented this feature.

- **Processing Overhead:** Processing overhead and memory requirements are minimal, considering the fact that a connection oriented protocol is sought to be superimposed over a basically connectionless protocol, SNMP. A comparison to the RowStatus method of row creation would be appropriate, since both seek to overcome similar limitations of PDU size. Indeed, our method can be used to provide operations for row creation and deletion, once the atomic operation is available using *control.continued*, and thus can completely replace the RowStatus method. Not only does this method obviate the necessity to create and maintain "virtual" or "artificial" rows, but, at equal or less cost, extends the capability of manipulating arbitrarily long lists of objects to the other operations like get and set. The amount of memory required to maintain and hold incomplete or inactive rows, as required by the RowStatus method, is no less than would be required to hold varbind lists in our method. Besides, processing overheads are required to maintain the semantics of an incomplete row.

- **Agent Capability:** It is recommended that an agent only implement tables that are narrow enough to fit into PDUs that can be handled by it, given the memory and computational resources of the agent. Sometimes, however, even though a large enough PDU can be handled by the agent, intermediate network entities or proxy agents might require PDUs to be smaller. This is where this method of chained PDUs could be used. Besides, it may often be necessary to manipulate many rows at one time, or to manipulate a large set of scalar variables atomically, leading to a need for the capacity to handle arbitrarily long varbind lists.

- **TooBig Error eliminated:** The use of *continued* PDUs allows the agent to avoid returning the `tooBig` error. This not only allows us to design manager-agent interactions in which the response PDUs may be different from the request PDUs, but also leads to an efficient implementation of large requests like the `GETBULK` request.

- **Implementation Issue:** Designing a state machine to work for only a `rowStatus` variable, then, becomes a device dependent feature, in the sense that if a device were to support a table that had a `rowStatus` column, the state machine that would implement setting of the `rowStatus` variable to different values would have to be specific to that device. This is eliminated in the proposed method, since handling states now becomes part of the agent software, and is totally device independent.

5 Implementation of the Control Paradigm

The CMU version of SNMPv2 was used as a the starting point for the implementation of the *control group* of variables. In this section we describe the process of transforming the agent to accept control variables.

A library of functions was implemented to provide the variable list handling capability. The basic snmp daemon was left mostly unchanged. The `snmp_agent` functions were, however, modified to incorporate the protocol machine described in Section 4. In particular, `snmp_agent_parse()` and `parse_var_op_list()` have had to be modified. The file containing the MIB definitions in ASN.1, `mib.txt`, had to be modified too, to include the definitions for the new control group of variables. Besides, another library of functions had to be written to implement each of the new types of functionalities introduced.

The total size of the agent code is about 10560 lines, of which, added files `var_list.c` and `agent_extensions.c` account for about 500 lines. The total code size increases by about 5%. However, this does not reflect the increase in the memory requirements. The memory requirements are dynamic, and memory is allocated as and when it is necessary to create storage space to accommodate lists of variables. The most important aspect is that the required changes are to be made only in `snmp_agent.c` and nowhere else. The implementation of the upgraded agent boils down to recompiling the agent software with the added library. This also ensures that the "new" agent is backward compatible with the "old" agent. It accepts and responds to everything exactly the way it did earlier, except that now it can process control variables.

6 Conclusions

We have proposed a technique whereby protocol semantics can be conveyed along with the data in an SNMP PDU. The basic principle is that, with an increased vocabulary of commands, there is less "uncertainty" in the protocol operation commands. Thus, a logical operation that must be carried out by a manager can now be expressed in terms of higher level primitives, rather than the ones already available, viz., `GET, SET, GETNEXT`, and `GETBULK`. Not only does this proposal provide for a set of higher level SNMP command primitives, it also provides for an easy, backward compatible method to enhance and enlarge this set, something that might prove invaluable in the future. As such, it is an all encompassing solution to many current and (hopefully) future problems in SNMP. It discourages site-dependent, implementor-specific, clever but opaque fixes to network management problems that might arise, and develops a standardized approach to protocol problem solving, encouraging openness.

Since SNMP will be around for a significant time in the future, this work outlines an approach towards an extensible SNMP that allows protocol operations to be extended in a backward compatible manner. The open-ended flexibility that this proposal seeks to introduce into SNMP should serve its needs well into the future.

Yet, a proposal is only as good as the performance of its implementations. There is considerable scope for a thorough identification of all fixes and protocol operations that are currently being done in a roundabout way or through non-standard methods, and reintroduction of these through the medium of control variables. There is no limit to the level of abstraction that may be introduced into the primitives that control variables provide. Current mandates of leanness and simplicity of the agent may not apply in the future, and network managers wishing to experiment with putting more load on the agent will have considerable room to maneuver.

References

[1] Jeffrey D. Case, Mark S. Fedor, Martin L. Schoffstall, and James R. Davin. A simple network management protocol. Request For Comments 1157, SNMP Research, May 1990.

[2] Jeffrey D. Case, Keith McCloghrie, Marshall T. Rose, and Steven L. Waldbusser. Introduction to version 2 of the Internet-standard network management framework. Request For Comments 1441, SNMP Research, Inc., April 1993.

[3] Jeffrey D. Case, Keith McCloghrie, Marshall T. Rose, and Steven L. Waldbusser. Protocol operations for version 2 of the Simple Network Management Protocol (SNMPv2). Request For Comments 1448, SNMP Research, Inc., April 1993.

[4] Jeffrey D. Case, Keith McCloghrie, Marshall T. Rose, and Steven L. Waldbusser. Textual conventions for version 2 of the Simple Network Management Protocol (SNMPv2). Request For Comments 1443, SNMP Research, Inc., April 1993.

[5] Keith McCloghrie and Marshall T. Rose (editor). Management information base for network management of TCP/IP-based internets: MIB-II. Request For Comments 1213, Performance Systems International, Inc., March 1991.

[6] Marshall T. Rose. *The Simple Book : An Introduction to Internet Management.* Prentice-Hall, Inc., second edition, 1994.

[7] William Stallings. *SNMP, SNMPv2 and CMIP : The Practical Guide to Network Management Standards.* Addison-Wesley Pub. Co., 1993.

[8] Steven L. Waldbusser. Remote network monitoring management information base. Request For Comments 1271, Carnegie-Mellon University, November 1991.