SNMP Query Language

by

Wengyik Yeong
Performance Systems International Inc.
yeongw@psi.com

Technical Report: 90-03-31-1
March 31 1990.

This report supersedes NYSERNet Technical Report 89-12-29-1

Abstract

This report describes an experiment to extend the capabilities of the Simple Network Management Protocol so as to allow the selective retrieval of information from manageable network elements. To this end, a network management model is developed based on the theory of relational databases. A possible implementation of management views to present a virtual view of management information is also described.

Possible optimizations that may be made in the SNMP querying process as a consequence of the model developed are also described.

This software and its supporting documentation is the product of work partially supported by the U.S. Defense Advanced Projects Agency and the Rome Air Development Center of the U.S. Air Force Systems Command under contract number F30602-88-C-0016. The existence of this software does not necessarily reflect any position or policy of the U.S. Government, and no official endorsement of the software or the work that produced it should be inferred.

The production of this software and documentation was also partially supported by NYSERNet Inc.

1. Introduction

The Simple Network Management Protocol (SNMP) [1] provides for the retrieval of complex network management information by *walking* tables of the Management Information Base (MIB) [2]. For example, to retrieve routing information, the *ipRoutingTable* table in the MIB is *walked*. This method is somewhat crude, and may require the network manager to retrieve all the information in a broad category, routing for example, in order to identify the few items of information desired to accomplish the network management task at hand. To date however, this SNMP method of management information retrieval has proven both effective and sufficient, as demonstrated by the successful deployment of the SNMP in the Internet.

However the continued growth of the Internet, coupled with the increasing proportion of that internetwork that is becoming network manageable with more widespread deployment of the SNMP could make this method of management information retrieval problematic. As the number of network elements to be managed grows, so too does the potential for the network manager to be overwhelmed by the amount of information, mostly undesired, being retrieved.

The use of database systems and other automated tools in the Network Management Station (NMS) partially alleviates this problem by imposing some structure on the information retrieved, and possibly automating the task of searching and manipulating the information, once it is available locally on the NMS. However database systems and automated tools do not address the root of the problem: the network manager is still obliged to retrieve potentially large amounts of information, and then sift through the resulting morass for the subset desired, a process only too prone to human error. Even with the use of automated tools to identify the subset desired, network and processor resources are still being wasted retrieving unneccessary information. There is a need for network managers to be able to selectively retrieve management information based on prespecified constraints that the information retrieved must satisfy.

The desirability of making a selective retrieval capability available to network managers would seem to point to the need for the Internet to move towards a more sophisticated network management protocol than the SNMP. While this is certainly possible, and would be the obvious solution, it is argued here that such an approach would be globally suboptimal.

Network management architectures such as that found in the CMIS/CMIP [3,4] provide selective retrieval capabilities by integrating them into the management protocol. This requires that all managed network elements support the ability to evaluate complex logical expressions, which in turn requires significant processor and memory resources. The paradigm is therefore one of intelligent network elements.

Given the large numbers, and the disparate functions and complexities of network elements in the Internet, it is argued that this strategy of embedding complex network management operations in the network management protocol is suboptimal, as it requires large numbers of network elements to dedicate significant amounts of processor cycles and memory space to the network management function. It is not the primary purpose of most network elements to support the network management function. Network elements exist to provide network connectivity, not to be managed.

In contrast, the NMS usually exists in the Network Operations Center (NOC) of those responsible for the management of the network, where significant resources are being dedicated to network management already. It is argued that a better, if not optimal solution to the problem of providing network managers with selective retrieval capabilities is to have the NMS ask more intelligent questions. The burden of intelligence, and the complex, processor and memory intensive software required to support such intelligence, should be placed on the NMS, not the network elements to be managed. After all, the purpose of the NMS is to provide the network manager with the ability to manage networks, and thus it is the responsibility of the NMS to provide the processor and memory resources necessary to support more complex management capabilities also.

So it is the primary objective of this work to provide network managers with selective retrieval capabilities without requiring any additional complexity in the software of managed network elements. A model is developed that provides this selective retrieval capability by using the *Get Request* and *Get Next Request* primitives of the SNMP.

It is also desirable to implement some form of information hiding, to make the structure of MIBs transparent to the user of the NMS, the network manager. The network manager is then hidden from unnecessary details regarding the actual structure of the information available on managed network elements, allowing the network manager to concentrate on the task of actually managing the network. *Management views* implement a form of information hiding, by presenting a virtual view of the MIB to the network manager, and dealing internally with the mapping from the virtual view to the actual structure of the MIB. As management views are a natural extension of the model being developed to allow selective retrieval of information with the SNMP, they are provided for in the model also.

2. Network Management Model

To provide the selective retrieval capability described above, the network manager must first be able to define the structure of the information desired, in terms of the items of information to be retrieved. For example, the network manager desiring some routing information should be able to define an object called a ‘‘route’’, in terms of the values that make up the object ‘‘route’’: a metric, a destination, a route type, and a route age, for example.

In addition to defining the structure of the information as above, the network manager also needs to be able to constrain the values that are to be retrieved. Continuing the example of the previous paragraph, the network manager may constrain the retrieval to information structured as ‘‘route’’s which have metrics of less than 5 units in value.

To provide capabilities as described above, network management is modeled as a series of operations on a relational database [5]. Specifically, the MIB is reorganized as a collection of tables, with a selective retrieve being viewed as a database retrieve on a database server, the network element from which information is being retrieved, and resulting in an unnamed result table. Management views then naturally become an extension of this model also, becoming analogous to the database views in a relational database system.

2.1. Tables and Attributes

To model network management in relational database terms, the MIB must first be reorganized to resemble a relational database. This is accomplished by mapping the MIB into a collection of **tables**, made up of **attributes**.

An **attribute** is either an MIB primitive object type identified by a leaf in the tree of object identifiers that make up the MIB, or a numeric constant. For example, the primitive object types

{ *ipRouteMetric1* }
{ *sysDescr* }
{ *ipAdEntIfIndex* }

map into the Query Language as the attributes **ipRouteMetric1**, **sysDescr** and **ipAdEntIfIndex** respectively. Similarly, the numeric constants *0* and *1* are mapped as the attributes **0** and **1**.

An **attribute instance** is the instantiation of an attribute for some arbitrary but fixed value in its range of values. In other words, an attribute instance identifies a binding of an attribute to some specific value, among all the values that the attribute can take on. For example, the instance of **ipRouteMetric1** for the route to network 10.0.0.0 would be a binding that could be identified by the literal *ipRouteMetric1_10.0.0.0*, while instances of **ifIndex** and **sysDescr** could for example be identified by the literals *ifIndex_1* and *sysDescr_0*.

It is important to note the difference between an **attribute** and an **attribute instance**. **ipRouteMetric1** is an attribute, whereas **ipRouteMetric1_10.0.0.0** is the instance of attribute **ipRouteMetric1** that identifies the value taken on by the **ipRouteMetric1** attribute when the route under consideration is for the network 10.0.0.0. By the same token, **ipRouteMetric1_128.127.0.0** is the instantiation of attribute **ipRouteMetric1** for the route to network 128.127.0.0.

In relational database terms, attributes may be thought of as *columns* in a relational table, with attribute instances being the specific values in different rows of the table, in the column corresponding to the attribute. Continuing the example of the previous paragraph, let the relational table which **ipRouteMetric1** belongs to have rows corresponding to routes, with one route per row. Then **ipRouteMetric1_10.0.0.0** identifies the value occurring in the **ipRouteMetric1** column, for the row referring to the route to network 10.0.0.0, whereas **ipRouteMetric1_128.127.0.0** identifies the value occurring in the same column, for the row referring to the route to network 128.127.0.0.

The value associated with an instance of an attribute corresponding to an MIB primitive object type is the value of the underlying SNMP variable. For example, the value of attribute instance **ipRouteMetric1_10.0.0.0** is the value of the SNMP variable

{ *ipRoutingTable ipRouteEntry(1) ipRouteMetric(3) 10.0.0.0* }

In contrast, attributes corresponding to numeric constants only have one instance, and one value, the value of the constant itself. For example, the one and only instance of attribute **0** is the attribute instance **0**, with value 0.

A **table** is then just a collection of attributes as defined above, with the data in the table being the values of the various instances of the attributes in the table. That is, a table is defined in terms of the columns that make it up, and the data in the table are those values occurring in the columns, in different rows. The relational table representing routes that has been used in the examples above, for example, is made up of the attribute **ipRouteMetric1** and some other attributes, with data consisting of various routes.

Tables in the MIB map directly onto tables in the SNMP Query Language in order to minimize the restructuring necessary. So, for example, the **ipAddrTable** in the MIB

{ *mib ip(4) ipAddrTable(20)* }

maps as the

ipAddrTable

table in the SNMP Query Language. In general, to further minimize confusion, MIB table *X* becomes the table named **X** in the Query Language.

Primitive object types in the MIB that are immediate children of MIB groups, that is, primitive object types that are not members of MIB tables but occur directly ‘‘beneath’’ MIB groups need to be collected into tables as well. This is accomplished by collecting all such MIB object types in a group that are not also members of tables in the group into a Query Language table by themselves. For example, the primitive object types

tcpRtoAlgorithm	**tcpRtoMin**
tcpRtoMax	**tcpMaxConn**
tcpActiveOpens	**tcpPassiveOpens**
tcpAttemptFails	**tcpEstabResets**
tcpCurrEstab	**tcpInSegs**
tcpOutSegs	

are all collected into the table **tcp** relational table. However the primitive object types

tcpConnState	**tcpConnLocalAddress**
tcpConnLocalPort	**tcpConnRemAddress**
tcpConnRemPort	

are in the **tcpConnTable** Query Language table, and not in the **tcp** table as these primitive object types are not immediate children of the *tcp* group in the MIB. These primitive object types are members of the *tcpConnTable* MIB table.

While Query Language tables are made up of attributes, it is not necessarily true that all attributes belong to a Query Language table. Attributes corresponding to numeric constants do not belong to any Query Language table. However, by virtue of the way in which the MIB was reorganized into tables, all attributes corresponding to MIB object types necessarily belong to exactly one Query Language table.

2.2. Dependencies and Dependency Lists

To effect selective retrieval of management information in an efficient manner, it is necessary to embed the relationship between the object type and object instance components of an SNMP variable in the Query Language. Central to the SNMP querying process is the proper specification of the object instance of the SNMP variable: careful specification will minimize the number of queries required to obtain a given set of information. Thus the inability to take advantage of the relationship between object type and object instance will require the retrieval of all information in an MIB table or group in order to obtain the subset desired. Conversely, proper specification of object instances by using the relationship between object type and object instance will yield an efficient querying process, with as little undesired information retrieved as possible.

To this end, the concept of a dependency is introduced. An attribute **a** depends on another attribute **b** if an instance of **b** is required to specify the name of the SNMP variable underlying the instance of **a**: attribute **a** depends on another attribute **b** if the value of an instance of **b** is needed to construct the object instance portion of the SNMP variables whose value is the corresponding instance of **a**. For example, the attribute

sysDescr

depends on the attribute

0

and the attribute

ipRouteAge

depends on the attribute

ipRouteDest.

The ordered list of attributes that an attribute **a** depends on is the **dependency list** of **a**. Note the requirement for ordering: for any attribute **a**, there is exactly one dependency list associated with **a**. The dependency list of an attribute **a** is the ordered sequence of attributes whose values are required in order to construct the SNMP variable underlying an instance of **a**.

For example, the dependency list of the attribute

ipRouteMetric1

is

{ **ipRouteDest** };

the dependency list of the attribute

sysDescr

is

{ **0** };

and the dependency list of the attribute

tcpConnState

is

{ **tcpConnLocalAddress**, **tcpConnLocalPort**, **tcpConnRemAddress**, **tcpConnRemPort** }.

Those familiar with the theory of relational databases should note that the concept of a dependency list is the inverse of the **key** relationship, with individual dependencies being **superkeys**. As this model requires data relationships in the **data** --> **key** direction, and not the **key** --> **data** direction, the **dependency**, and not the **key** concept is used.

2.3. Constraints

To provide selective retrieval capabilities to the network manager, it is necessary to allow the network manager to constrain the data to be retrieved, in addition to being able to define the form of the result data. This is done by having the network manager specify **constraints** which the data to be retrieved must satisfy.

A **constraint** is recursively defined to be made up of:

- a set of **subconstraints** related by logical operators, and possibly nested.

or

- a **simple comparison** which tests the value of an attribute instance. The test must evaluate to the logical value *True* for the data to satisfy the trivial constraint that is the simple comparison.

For example:

ipRouteAge > 10

ipAdEntIfIndex = **ifIndex**

are both examples of simple comparisons while

ipRouteAge > 10

ipAdEntIfIndex = **ifIndex**

ipRouteDest > 128.145.0.0 **AND** **ipRouteDest** < 128.146.0.0

(**ifStatus** = 1 **OR** **ifStatus** = 2) **AND** **ifInUcastPkts** > 10000

are all examples of constraints. Note the two trivial constraints, each made up of a single comparison.

The network manager can therefore be viewed as defining a constraint, ultimately made up of simple comparisons related by logical operations, which the data to be retrieved must satisfy. It is the function of the query optimization process described later in this paper to determine, based on the constraint specified, the sequence of queries that have to be made to most efficiently retrieve the information desired by the network manager.

3. Communities and Views

Management views provide a virtual view of the MIB to the network manager. Management views are modeled in a way analogous to the views found in database systems in general, and serve a similar purpose.

Conceptually, a management view serves to hide the actual structure of the MIB, as specified in reference [6] from the network manager. A **network management view**, or simply **view** is the collection of tables visible to the query language: a view is the set of tables whose attributes make up the database that the user may manipulate.

Views also provide the network manager with the ability to customize the structure of the MIB to individual needs, hiding unnecessary detail and reorganizing information into a form more suitable to the needs of the network manager. The relational tables that make up the MIB for the purposes of the Query Language can be further reorganized, with attributes being dropped, attributes being added from other tables, or entirely new tables being defined.

For example, a network manager could define a new table in a management view, **myIfTable** that contained only the attributes

ifIndex	**ifSpeed**
ifInUcastPkts	**ifOutUcastPkts**
ifInOctets	**ifOutOctets**

Although in the example, all the attributes of the new table were derived from the same table in the reorganized MIB, this need not always be true. The view concept allows for the definition of view tables containing attributes from multiple actual tables, with the same attributes occurring in multiple view tables. For example, it would be possible to add the **ipAdEntAddr** attribute to **myIfTable** as defined above, and still have such an attribute be a member of another table in the view.

Note that this does not in any way violate the requirements of the section **Tables and Attributes**. The tables in a view exist solely for the convenience of the Query Language user. Views and the internal structure of management information are orthogonal. Indeed, the view of management information presented to the network manager by the Query Language, the internal representation of management information within the Query Language and the representation of management information in the network element may all be different.

To distinguish between the multiple views that may be present on an NMS, additional semantics are assigned to the SNMP community. Associated with each community is a view, which determines the structure and availability of management information within that community. Hence, the view currently in use by the network manager using the Query Language is determined by the SNMP community within which network management is occurring. This binding of views to SNMP communities also limits the Query Language so that it operates on only one view at any instant. This is in line with the database model, where a database view represents the universe of operation.

The view concept defined here is a structuring convenience, not a security feature useful to network elements wishing to implement access controls. Views, as defined here do not extend beyond the confines of the NMS, and can only restrict the availability of management information by excluding such information from the virtual representation presented to the network manager. Views cannot prevent the retrieval of management information of any sort that is made available to the NMS by the network element, regardless of whether or not such information actually occurs in the view. Views are in this sense different from the SNMP MIB views defined in reference [1]: although both are bound to the SNMP community, the former is an MIB representation mechanism, whereas the latter is an access control mechanism.

4. Optimization

In general, optimization is performed in order to realize the selective retrieval of management information as efficiently as possible. The process of retrieving management information can expressed as two steps, querying the managed element for information by means of the SNMP, and comparing the information retrieved to the user specified constraint. Any information found not to satisfy the specified constraint in the second step would be rejected, and not presented to the user.

In keeping with the motivation for selective retrieval itself, the primary objective of optimization is to minimize the number of SNMP queries that need to be generated in order to retrieve the information desired, thus minimizing the overhead incurred by managed network elements in support of network management. It is also advantageous to optimize the second step, the comparison of data with the user constraint, to the greatest extent possible so as to minimize NMS resources.

As a result of the work performed, three general areas have been identified in which optimizations may be performed:

(1) **Packet Size Maximization –** by maximizing the amount of information retrievable in a query, the number of queries that need to be generated to obtain a given amount of information is minimized. This optimization involves maximizing the number of SNMP variables whose values are requested in each outgoing query.

(2) **Constraint Simplification –** by minimizing the complexity of the logical expression against which data is to be compared, the amount of checking involved in determining if an item of data satisfies user constraints is minimized. Simplification of the logical expressions that make up the user specified constraint also allows the computation of bounds restrictions to be done more efficiently.

(3) **Bounds Restriction –** by examining the constraint, the most restrictive bounds possible are determined on the range of values that each attribute can take on. This serves to minimize the number of queries that need to be generated.

4.1. Packet Size Maximization

In retrieving information from a network element, the amount of querying to be performed depends on the amount on information desired, and is ultimately dependent on the requirements of the network manager. However, given a fixed amount of information to be retrieved, as dictated by a fixed specification by the network manager, the amount of SNMP traffic required to retrieve the information varies inversely with the number of variables that each SNMP request contains. For a given amount of information, maximizing the number of SNMP variables in each packet will maximize the amount of information that can be retrieved per packet, and thus minimize the number of SNMP requests required to retrieve all the information.

Reference [1] defines that the object instances of all SNMP variables that are derived from primitive MIB object types in the same MIB table are constructed in the same way. As such, a starting point for a packet minimization strategy would be to query for all variables derived from object types in the same MIB table in a single packet, with a fixed object instance common to all the variables in that packet. All the information in the table would be retrieved with a single sequence of packets that would *walk* the table once.

Examination of the SNMP querying process reveals that in table *walking*, the only component of the variables in the packet that changes is the object instance, barring *falling off* the table being *walked*. However, if this strategy of querying for the values of all SNMP variables derived from the same MIB table is adopted, all the object instances in the packet are identical, and therefore change in the same manner and at the same rate. So it makes no difference whether the information contained in the MIB table is retrieved by walking the table once for each variable, or once, for all the variables. The strategy of querying for all SNMP variables derived from MIB objects from the same MIB table in one packet sequence is therefore reasonable.

From examining the SNMP querying process, it would also seem that querying for SNMP variables derived from MIB object types from more than one MIB table would be difficult. Consider two MIB tables T_1 and T_2 with N_1 and N_2 rows respectively. If the querying of both tables were to be done simultaneously each packet in the sequence would have two different sets of object instances that were unrelated to each other, and would not change in the same manner. Inevitably, unless $N_1 = N_2$, the table *walk* would fall off one table before the other, necessitating logic to deal with the situation. Generalizing to m tables, the logic gets difficult. Hence, although this avenue may yield further optimizations, it is not yet completely explored in the Query Language.

Translating the above requirements into Query Language terms, the values of all attributes with the same dependency list should then be retrieved in the same packet sequence. As a practical consideration, this amounts to collecting all the attributes with the same dependency list into a table, allowing querying to be based conceptually on retrieval of entire tables at once, instead of on a per attribute level.

The grouping of attributes with common dependency lists into a table provides another opportunity for a simple optimization. With this grouping strategy, all the attributes in a table will have the same dependency list, and hence, a structure optimization can be realized by associating dependency lists with tables, and not individual attributes. For optimal performance, all attributes sharing a common dependency list should belong to the same table in the Query Language.

Note that to be able to conceptualize retrieval at the table level, all the non-constant attributes of the table's dependency list must belong to the table also, in order that these attributes' values can be retrieved together with the values of all the other attributes that depend on them. The restriction would also apply to constant valued attributes, except that the value of a constant valued attribute is trivially derived. So it is a requirement of the Query Language that all non-constant attributes that are members of a table's dependency list occur in the table also.

4.2. Constraint Simplification

Central to the operation of the Query Language is the constraint specified by the network manager. The constraint is examined for the purpose of optimization in the area of bounds restriction, determining the range of values of each attribute that will be retrieved from the network element. After the retrieval of information from the network element, the constraint is examined again to exclude any information retrieved that should be rejected because the information does not satisfy the constraint.

Due to the importance of the constraint, it follows that it is best that the constraint be manipulated into a convenient form which lends itself to easy examination, with a minimum of computing necessary on each examination. For the purposes of the Query Language, the most convenient form is one where the constraint consists of a collection of expressions related by a minimum of logical operators. To accomplish this elementary laws of boolean logic, as described in references [7,8] are used.

In order to minimize the number of logical operators found in the constraint, and thus simplify the computation needed in examining the constraint, **NOT** operators are removed by application of the DeMorgan Laws, the Law of Double Negation, and by variable substitution:

$\overline{(\mathbf{a}\ \text{AND}\ \mathbf{b})}$	$\bar{\mathbf{a}}$ OR $\bar{\mathbf{b}}$	DeMorgan
$\overline{(\mathbf{a}\ \text{OR}\ \mathbf{b})}$	$\bar{\mathbf{a}}$ AND $\bar{\mathbf{b}}$	DeMorgan
$\bar{\bar{\mathbf{a}}}$	**a**	Double Negation
$\bar{\mathbf{a}}$	$\mathbf{c} = \bar{\mathbf{a}}$	Variable Substitution

Variable substitution of **c**, the inverse of **a** is possible because every logical comparison has an inverse:

operation	inverse
$=$	$\neq$
$\neq$	$=$
$>$	$\leq$
$<$	$\geq$
$\geq$	$<$
$\leq$	$>$

By taking advantage of these manipulations of boolean logic, it is possible to eliminate the **NOT** operator completely from the constraint.

It should be noted that { **NOT**, **AND** } and { **NOT**, **OR** } are both functionally complete sets of operators as defined in reference [8]. Hence, it is possible to eliminate either the **AND** or the **OR** operator, at the expense of keeping the **NOT** operator. However the elimination of either the **AND** or the **OR** operators will result in more complex expressions

$$\mathbf{a\ AND\ b} \qquad \overline{\overline{\mathbf{a}}\ \mathbf{OR}\ \overline{\mathbf{b}}}$$
$$\mathbf{a\ OR\ b} \qquad \overline{\overline{\mathbf{a}}\ \mathbf{AND}\ \overline{\mathbf{b}}}$$

Hence, since eliminating the **NOT** simplifies the constraint to a greater extent, it is the **NOT** operator that is eliminated.

Also, an optimization can be realized by noting that in a disjunction, if any term of the disjunction evaluates to the logical value *True*, the entire disjunction takes on the *True* value. Thus, if the constraint were to be converted to Disjunctive Normal Form (DNF) [9], satisfaction of any DNF term would satisfy the constraint.

Note that this optimization is not always applicable: satisfaction of one of multiple DNF terms derived from the user constraint is a sufficient, but not necessary, condition to satisfy the entire constraint. Thus to determine if information satisfies the user specified constraint, the *entire* constraint must be examined, not just a DNF term, if this optimization is applied. The optimization is useful when performing optimizations in the bound restriction area. As it turns out, this optimization is *required* to allow the performance of any optimizations in the bound restriction area at all.

4.3. Bounds Restriction

The primary means of retrieving complex network management information from the MIB by using SNMP is to *walk* portions of the MIB. The purpose of performing the bounds restrictions optimizations is primarily to reduce the size of the MIB that needs to be *walked*, so as to minimize the number of queries that need to be generated to do the *walking*.

The only attributes whose values contribute to the number of queries that need to be generated are those attributes that occur as dependencies. Other attributes, that do not occur as dependencies, do not by definition contribute to the construction of the object instances of SNMP variables. The value ranges of such attributes that do not occur as dependencies therefore do not affect the querying process in any way. Hence in performing the bounds restrictions optimizations, only the attributes that occur as dependencies are considered; the application of the bounds restrictions optimizations to any other attributes would not contribute anything to the goal of minimizing queries.

The first optimization possible in bounds restriction is the straightforward one of finding the most restrictive range of values possible that can be taken on by a dependency for all dependencies in existence, given the constraint at hand. This is best explained by the algorithm

for each table
    for each dependency of table
        for each simple comparison in constraint
            if simple comparison allows value range to be restricted further
                restrict value range further

Note that this algorithm breaks down if the constraint implies disjoint value ranges for a particular dependency because it assumes that each dependency has only one range of legal values. This is not necessarily true. For example, considering the **ifTable** table, if the constraint was

ifIndex > 4 **OR ifIndex** < 2

the algorithm would fail. In general, any constraint that contains disjunctions that imply multiple, disjoint, value ranges for a single dependency will cause the algorithm to fail.

However an optimization was developed in the previous section to extract the DNF terms from a constraint, which would solve the problem of disjoint value ranges. By applying the optimization to extract DNF terms, each constraint considered by the algorithm will be free of disjunctive terms, thus guaranteeing that there will only be one value range for the constraint under consideration. Therefore it is required that the optimization of the previous section be used if any optimizations are to be done in restricting bounds at all.

In applying the optimization to extract DNF terms from a constraint however, an implicit assumption is made that all the value ranges implied by the different DNF terms *are* disjoint. This is no more true than the original assumption that each dependency implies only one range of values. For example, considering the **ifTable** again, the constraint

ifIndex > 3 OR **ifIndex** < 7

implies two ranges $\{4,\infty\}$ and $\{-\infty,6\}$ that overlap to form the single range $\{-\infty,\infty\}$. Hence there is a need for a second optimization, or a post-optimization optimization, which examines all the value ranges generated by the algorithm above, and combines those that overlap.

Although the *walking* of tables is the primary means of management information retrieval in the Query Language, it is necessary to recognize situations when the SNMP *Get Request* is more suitable. Otherwise resources would be wasted performing table *walking* when a *Get Request* would have sufficed.

When a constraint is specified such that all attributes of a table can be instantiated only from information in the constraint, then *Get Request*s should be used to retrieve the information in the table. In more familiar terms, *Get Request*s should be used to retrieve information when all the object instances of all the SNMP variables needed to do the information retrieval can be constructed only from information in the constraint. This situation only occurs when each value range in existence after the post-optimization of the previous paragraph only implies the retrieval of one value of each attribute in the tables. Hence this is the necessary and sufficient condition for the use of *Get Request*s in preference to *Get Next Request*s for the retrieval of information.

5. Conclusions

As part of this work, an implementation of the SNMP Query Language was performed in order to validate the ideas described in this paper. This implementation is now complete, and is available for general use. Experience gained from implementing, and subsequently using, the SNMP Query Language indicates however that while it is certainly possible to provide a selective retrieval capability on top of the SNMP, doing so is not without problems.

The goal of minimizing the amount of unusable information retrieved is realized only to a limited extent, because of the inherent dependence of the SNMP querying process on the object instance of the SNMP variables found in SNMP queries. Efficient querying is realized only if members of the dependency lists of the various tables occur in the constraint. That is, the retrieval of information of no interest to the network manager is minimized only if certain attributes are constrained. The performance of the selective retrieval process therefore varies widely, depending on the nature of the constraint specified by the network manager.

Given the relative simplicity and light-weight of the SNMP, this shortcoming is to be expected. The shortcoming represents a tradeoff between added complexity in the network management protocol, and the ease with which more sophisticated network management functions may be implemented. Essentially, to what level of mediocrity should the simpler network management operations such as straightforward retrieval be dragged down to by the requirements of the more complex operations such as selective retrieval? The SNMP resolves the tradeoff in favor of the simpler management operations, so that the simpler operations are unencumbered while more sophisticated operations incur a proportionately larger amount of overhead when used.

This work has attempted to demonstrate that it is possible to extend the SNMP in order to provide selective retrieval capabilities to the network manager. It is believed that this objective has been attained.

6. Acknowledgements

The many contributions of Mr. James R. Davin of the MIT Laboratory for Computer Science are gratefully acknowledged. His comments and suggestions were invaluable in developing the Query Language, and he served as an excellent sounding board for ideas in the early days of Query Language development.

NYSERNet Inc. and Performance Systems International Inc. are both gratefully acknowledged for allowing the author to spend the time necessary to develop the Query Language, and then allowing it to be freely available. In particular, no small amount of gratitude is owed to Mr. Martin Schoffstall, formerly Vice President of Technology for NYSERNet Inc. and currently Vice President of Performance Systems International Inc. for putting up with the author through the months when the ideas for what became the Query Language were developed, and while the Query Language was being implemented.

Bibliography

[1] *The Simple Network Management Protocol*, J. D. Case, M. S. Fedor, M. L. Schoffstall, J. R. Davin. Internet Request for Comment (RFC) 1098, April 1989.

[2] *Management Information Base for Network Management of TCP/IP-based internets*, K. Z. McCloghrie, M. T. Rose. Internet Request for Comment (RFC) 1066, August 1988.

[3] *Information Processing Systems -- Open Systems Interconnection -- Management Information Service Definition -- Part 2: Common Management Information Service*, ISO/IEC JTC1/SC21. International Standards Organization Draft International Standard (DIS) 9595-2, August 1988.

[4] *Information Processing Systems -- Open Systems Interconnection -- Management Information Service Definition -- Part 2: Common Management Information Protocol*, ISI/IEC JTC1/SC21. International Standards Organization Draft International Standard (DIS) 9596-2, October 1988.

[5] *A relational model for large shared data banks*, E. F. Codd. Communications of the ACM, Volume 13 No. 6.

[6] *Structure and Identification of Management Information for TCP/IP-based internets*, M. T. Rose, K. Z. McCloghrie. Internet Request for Comment (RFC) 1065, August 1988.

[7] *Sets of Independent Postulates for the Algebra of Logic*, E. V. Huntington. Transactions of the American Mathematical Society, 1904.

[8] *Introduction to Switching Theory and Logic Design*, F. J. Hill, G. R. Peterson. John Wiley and Sons, 1968, 1974, 1981.

[9] *Computability, Complexity and Languages*, M. D. Davis, E. J. Weyuker. Academic Press, 1983.