Appeared in the *Proc. of 14th National Conf. on Communications, NCC-2008*, IIT Bombay, 1-3 Feb 2008.

Network Management Traffic Optimization

C. Jagadish[1], S. Thanga prakash[1] and Timothy A. Gonsalves[2]
[1]Midas Communication Technologies Pvt. Ltd., Chennai, India
[2]Computer Science and Engineering Department, IIT Madras, India
Email: {cja@midascomm.com, thanga@midascomm.com, tag@tenet.res.in}

Abstract - **Network Management Systems (NMS) are of paramount importance to manage large and complex telecom networks from a central place. To provide interoperability, these NMS systems are usually based on standards. Simple Network Management Protocol (SNMP) is the most widely deployed management standard. SNMP is simple but has limitations in terms of bandwidth overheads. High object identifier (OID) overhead, single request single response, no filtering capability are the typical limitations of SNMP which make the response time of a poll request high, especially over a low speed network. In this work, we propose a few extensions to SNMP to optimize the traffic and response time with minimal changes in the managed devices. From the implementation and experiments carried on corDECT networks, we show that there is considerable reduction in bandwidth requirements and response time for the management operations especially over dialup and low speed links.**

I. Introduction

Data and telecom networks are becoming increasingly complex in terms of both geographical spread and heterogeneity. To aid interoperability, centralized NMS is based on network management standards. Among the various management standards, SNMP [1] is the most widely deployed. Owing to this, one cannot rule out the usage of SNMP in small and medium range data and telecom networks.

The messages supported by SNMPv1 are get-request, set-request, getNext-request and trap. Additionally, SNMPv2c supports getBulk-request and inform-request [2]. Both SNMPv1 and SNMPv2c are weak in terms of security, while elaborate security mechanisms have been built into SNMPv3 [3]. One of the major limitations of SNMP often criticized in literature is its traffic overheads and consequent large response times. Traffic overhead is mainly due to *OID overhead* and *the lack of filtering capability*. Owing to these limitations of current SNMP getNext and getBulk services, it is usually necessary to retrieve all the rows of a Management Information Base (MIB) table even though only some of them are needed.

Apart from this the SNMP protocol is based on *single request and single response*. Traditional SNMP managers use stop-and-wait mechanism for request-responses. Due to this, response times and latencies become high for large table retrievals. Some JAVA managers implement multiple parallel threads. However, for performance reasons there is an upper limit on the number of parallel threads. The majority of the SNMP traffic consists of getNext/getBulk requests [4]. The next request in case of a getNext or getBulk request depends on the previous response. Due to this reason it is not possible to have parallel threads for getNext/getBulk requests. All this makes SNMP quite inefficient especially over a low speed link.

In this paper we propose a few simple extensions to the SNMP protocol which makes it suitable for low speed links. By making these extensions both at manager and agent, we find that there is considerable reduction in both bandwidth requirements and response times for management operations over dialup and other low speed links.

In section 2 we give a brief survey of the SNMP traffic optimization methods in the literature. In section 3 we describe the proposed techniques and protocol extensions. Section 4 covers our implementation. Experimental results and discussion are presented in section 5. The paper concludes with conclusions and scope for future work in section 6.

II. Background

There is a rich literature on mechanisms to optimize SNMP response time and bandwidth overheads. We classify optimization techniques as “techniques to reduce the number of request-responses”, “techniques to reduce packet overheads”, “techniques to effectively use the network bandwidth” and “techniques to reduce the amount of data retrieved (filtering)”.

A. *Techniques to reduce the number of request responses*

In SNMPv1, to retrieve a table *getNext* message is provided. To improve the performance over *getNext,* SNMPv2 has *getBulk*. The efficiency of retrieving bulk data using SNMP getBulk depends on the maximum supported packet size. The maximum packet size depends on the implementation of agent and manager. Although the maximum message size supported by SNMPv2c and SNMPv3 is 2^{31}-1 bytes, it is seen that in practice the maximum SNMP message size is less than 1500 bytes [4].

B. *Techniques to reduce packet overheads (OID Optimization)*

One of the major limitations of SNMP is the OID naming overhead. The length of an OID reflects the depth of the attribute in the MIB tree. There are a number of techniques proposed for OID optimization such as ObjectID Delta Compression (ODC) [5] and ObjectID Prefix Compression (OPC) [6]. The average compression obtained in retrieval of MIB-II data is nearly 25% using OPC [7]. The OID suppression technique proposed by EOS suggests retrieval of all columns of a row [6]. This becomes inefficient when the number of columns in a table is high and only a small set is relevant to the manager. Both algorithms require implementation of new PDU and compression algorithms at both agent and manager.

C. *Techniques to effectively use the network bandwidth*

One research work [8] proposes splitting of a table retrieval walk query into multiple threads of sub-walk queries to optimize the response time, where each thread retrieves a portion of the table. The shortcomings of this approach are addressed by another proposal by introducing a new *getsubtree* request [9]. This allows the agent to return multiple related responses for a single request. However the manager is forced to retrieve the entire subtree, irrespective of the actual rows of interest.

D. Techniques to reduce the amount of data retrieved

Various techniques were published for traffic optimization by filtering. In a mobile/intelligent agent proposal [10], a JAVA based mobile agent migrates to the device, for retrieving bulk data with a predefined filter. The mobile agent collects the responses through multiple getNext queries and returns a JAVA object containing responses that match the filtering criteria. This technique naturally requires the device to have support for JAVA execution and mobile agent implementation. In the paper [11] the author proposes, a new PDU *getRows* an extension to getBulk, where the request PDU carries the filter criteria. The response carries only the rows matching the filter criteria. The author shows significant improvement over getBulk and getNext. The paper focuses more on the filtering aspect and less on the single-request-single-response limitation affecting low speed links.

One proposal for filtering could be defining a MIB for filtering. The manager sets the required filter criteria before making the *getNext / getBulk* request. The response will take into account the filter set. Here the limitation is that if multiple managers are there then separate filters should be available for each manager and the manager should remember to clear the filter after the *getNext/getBulk* request is completed. Also, as SNMP is UDP based a *get* has to be done to ensure that the previous filter *set/reset* is successful.

III. Proposed techniques and protocol extensions

This section describes the proposed techniques and extensions to SNMP protocol.

A. Get/set response time optimization based on sliding window approach

In the proposed solution, a single threaded manager sends *N* independent get/set-requests to the agent without waiting for response, where *N* is the window size. Subsequently, on receipt of each response, the next request is sent. The manager thus always maintains upto *N* outstanding requests. In turn the agent will always have a request for processing. Requests are sent to the device in the order of their arrival.

One important aspect is *time-out handling*. Normally, timeouts are kept high and are increased exponentially. In this approach if one response gets delayed or lost then the effective window size reduces to *N*-1. We have optimized this with the consideration that though the underlying protocol is UDP, as normally there is only one path between the manager and the device reordering of packets is not expected. Based on this, if we receive response for sequence number *K*+1 before response for *K,* then we assume that the response for *K* is lost. However this will work even if packets come out of order. At this time we simulate timeout for the request *K*. In case if two duplicate responses are received (with the same sequence number) then the earlier response is accepted and the later one is rejected. *This requires no change at the device.*

B. The getNext/getBulk optimization

Often the requirement is to get the rows of a table in a given range, rather than getting a count number of rows as provided by max-repetitions of getBulk request. We introduce a new message called getRangeTable, a simple extension to SNMP getNext/getBulk message. The packet structure includes lower and upper range fields. Conventions, such as default lower range as the first instance in the table, and default upper range as the last instance in the table are defined.

The agent implements this as a series of getNext/getBulk requests locally for the given range. It sends the responses individually in multiple responses. The manager receives these multiple responses and processes them sequentially. This reduces one-side latency, as only one request is sent and responses are received back to back. The responses are chained with a sequence number called link id starting from 1. The last response contains negative value of the last sequence number indicating the end. The manager can detect the loss of a response by watching the link ids. In case of loss of a response, manager retrieves the lost packet by a getNext/getBulk request using the previous link id response packet. On receipt of each response packet, timeout is set for receiving the next response packet. In case of a timeout, the manager increments the retry count and reissues the getRangeTable request with the lower range updated with the instance number of the last received response packet.

The getRangeTable is very useful when the table index is a meaningful instance attribute for the table as in case of "*subscriber table*" where subscriber number is the index, "*equipment table*" where equipment id is the index etc. A typical query could be, "get the list of subscribers having subscriber numbers from 1 to 500". With this single query, upto 500 chained responses are sent from the agent back to back to the manager, one response for each available subscriber.

C. Filtering Support – getFilteredTable

This is an extension to the getRangeTable. Here in addition to the range input a generic filter input is added to the packet. In the getFilteredTable request agent interprets both the range input and the filter criteria. The filters are simple but powerful having provision for both logical and arithmetic operators. Multiple filter criteria are combined with logical operators. Similar to getRangeTable, agent simulates continuous getNext/getBulk requests internally for the given range and sends the filtered responses back to back. The packet loss handling and timeout handling at the manager is similar to the one explained for getRangeTable. This gives substantial improvement in both traffic and response time. Some typical requests of this type are "get the routing entries from the router having the destination port as interface *3"*, "get the list of wireless subscribers between 1 to 5000 having receive signal strength below the threshold value 30dbm" etc.

D. Enterprise OID Optimization

Normally each product development company registers for a Private Enterprise Number (PEN) with IANA [12] and places its proprietary MIB under the Enterprise branch of MIB. The PEN allotted for Midas communication [13] is 3794. The MIB for the corDECT product of Midas starts at 1.3.6.1.4.1.3794.1.

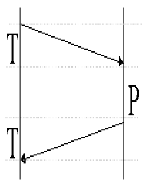

Fig. 1 Round Trip Time (R)

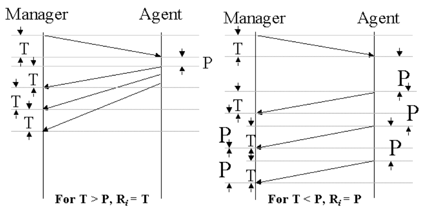

Fig. 2 Improved Response Time (R_i)

The corDECT system is a wireless telecom switch based in DECT technology. Standard SNMP MIB is not available for most of the MIB needed for corDECT. From the usage statistics we see that more than 95% of MIB access happens to corDECT Enterprise specific MIB. From this we see that each OID access has a constant overhead of 8 bytes (one byte for first two oids, 2 bytes for 3794) i.e. "1.3.6.1.4.1.3794.1". The table OID in corDECT MIB has the following additional sub-oids "group.table.entry.attribute.instance", similarly a group has the following additional sub-oids "group.atribute.instance".

Effectively the length of a table OID = 13 bytes and length of a group oid is 11 bytes. To reduce this overhead we re-registered the corDECT MIB under "iso" branch of the SNMP MIB i.e., "iso.corDECT" having OID 1.39. Effectively the corDECT MIB is registered twice in the MIB with two different module names, which is allowed. With this the agent responds to both 1.39 as well as to 1.3.6.1.4.1.3794.1. This is to ensure compatibility with the external standard managers. With this, our EMS manager can access the corDECT MIB with prefix 1.39 instead of 1.3.6.1.4.1.3794.1. With this, the number of bytes needed for a table OID = 6 bytes and for group OID = 4 bytes.

This gave us an effective constant reduction of 7 bytes per OID, which is 54% optimization for tables and 63% for groups. Also, from the statistics [14] it is seen that major processing time at the agent and manager are for OID parsing and processing. With this optimized OID, we have seen considerable reduction in processing time. *This is done only with an additional MIB loading and with no code change at both manager and agent.*

E. Computation of expected gain in response time

We denote the propagation delay of a request from manager to agent by T_m, the processing time of the request at the agent by P and the propagation delay for response from agent to manager by T_a. Then the response time or Round Trip Time (R) of a request is given by $R = T_m + P + T_a$

For the sake of simplicity, let us consider the link speed in both directions and the packet size of request and response to be the same. By this the propagation delay of request and response becomes equal. Thus, $R = 2T + P$. This is illustrated in Fig-1.

Fig-2 shows Improved Response Time (R_i) under two conditions $R_i = T$ where T >P and $R_i = P$ where T < P. We can

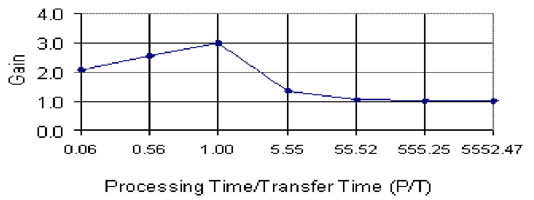

Fig. 3 Gain vs. P/T

Table-1 Techniques used and possible optimizations

Original message Type	Technique used (new message type)	Possible optimizations: Traffic	Response time	Processing Time at Agent
get/set	Sliding window	No	Yes	No
	Enterprise OID Optimization	Yes	Yes	Yes
getNext / getBulk	Multi Responses (getRangeTable)	Yes	Yes	No
	Filtering (getFilteredTable)	Yes	Yes	No
	Enterprise OID optimization	Yes	Yes	Yes

prove that the *improved response time of the sliding window is the same as that of the multiple responses*. The gain improvement is equal to the ratio of R and R_i, Gain Improvement $GI = R / R_i$. In other words the gain improvement depends on the ratio of processing time (P) and transmit time (T). This ratio is referred as PTR. Gain increases as the PTR approaches 1 and reduces as PTR digresses from 1.This is shown in Fig-3.

IV. Implementation

Ease of implementation and minimal modifications to the current SNMP implementations are the key considerations of this proposal. No changes are needed at agent for the sliding window optimization or OID optimization. For the getNext and filter optimizations, we added three new "wrapper PDU" types GetRangeTable-PDU, GetFilteredTable-PDU and MultiResponse-PDU encapsulating the existing SNMP PDU. The new PDU contains an additional header on top of the existing SNMP PDU. This support is needed both at the manager and at the agent.

For easy implementation we developed a proxy at the agent side that receives the new request PDU, extracts the encapsulated SNMP request PDU and additional information i.e., lower range, upper range and filter details from the header. *The proxy validates that all the requested varbinds belong to the same table else it returns an error.* Then the proxy queries the actual agent in a loop for the given range and sends the responses back to back, which match the filter criteria, using MultiResponse-PDU. The responses are chained with incremental link ids. The last response contains the negative value of the last sequence number. The proxy can be placed on a standalone server outside the device. The ASN details for the new GetRangeTable-PDU, GetFilteredTable-PDU and MultiResponse-PDU are given below. We see that the overhead for GetRangeTable-PDU is 21 bytes and that of GetFilteredTable-PDU is 43 bytes, which are sent only once. The overhead with each MultiResponse-PDU is 12 bytes compared to standard SNMP response.

```
Extended-SNMP-PDUs ::= CHOICE {
        getRangeTable getRangeTable-PDU,
        getFilteredTable getFilteredTable-PDU,
        multiResponse multiResponse-PDU,
        snmpPDU PDU }
-- PDUs
```

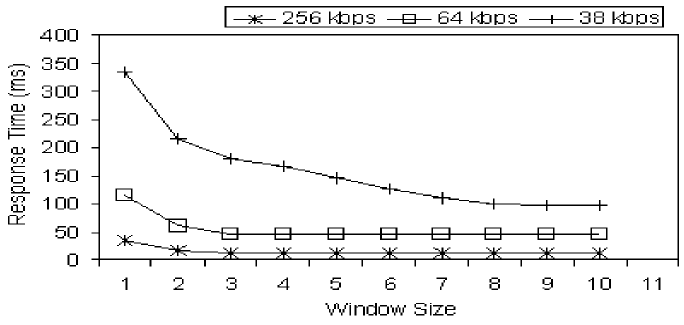

Fig. 4 Sliding window response time
(a) low bandwidth links

```
getRangeTable-PDU        ::= [9] IMPLICIT GetRangeTable-PDU
getFilteredTable-PDU     ::= [10] IMPLICIT GetFilteredTable-PDU
multiResponse-PDU        ::= [11] IMPLICIT MultiResponse-PDU

GetRangeTable-PDU ::= SEQUENCE {
    request-id INTEGER,
     lower-Range OCTECT STRING,
     upper-Range OCTECT STRING,
    snmpPdu  PDU }-- getNext / getBulk

GetFilterTable-PDU ::= SEQUENCE {
    request-id INTEGER,
    lower-Range OCTECT STRING,
    upper-Range OCTECT STRING,
    filterList     FilterList,
    snmpPdu  PDU }  -- getNext / getBulk

MultiResponse-PDU ::= SEQUENCE {
     request-id INTEGER,
     linkId  INTEGER,
    snmpPdu  PDU } -- SNMP Response PDU

FilterList ::= SEQUENCE (SIZE (1..max-bindings)) OF FilterCondition

FilterCondition ::= SEQUENCE {
    attribute  ObjectName,
    operatorValue INTEGER,
    value ObjectSyntax }
```

V. Experimental Results and Discussion

We have taken various performance readings with the new agent and manager for different network bandwidths ranging from 38 kbps to 10 Mbps. A dialup connection was used to establish a 38 kbps link. For *N* x 64 connectivity we used a router and DSL modem pair at both ends of the connection. The modems are configurable for various link speeds in steps of 64 kbps. Ethernet is used for 10 Mbps link. Readings are taken for different SNMP messages (standard and proposed). Each request message has 10 attributes. The same experiment set is repeated for various network bandwidths. The aim of the experiment was to measure the round trip delay, processing time at the agent, bytes transmitted and bytes received. Table-1 summarizes different techniques used and possible optimizations.

A. *Performance of get/set optimization*

Fig-4 shows the average *get/set* response time with sliding window for different window sizes. It is seen that the response time decreases with increase in window size up to an extent and then stabilizes. Let us call this as the Stabilized Window Size (SWS). We see that for the dialup line SWS is 9. In the *N* x 64 kbps setup we see that SWS is 3 for 64 kbps link and increases

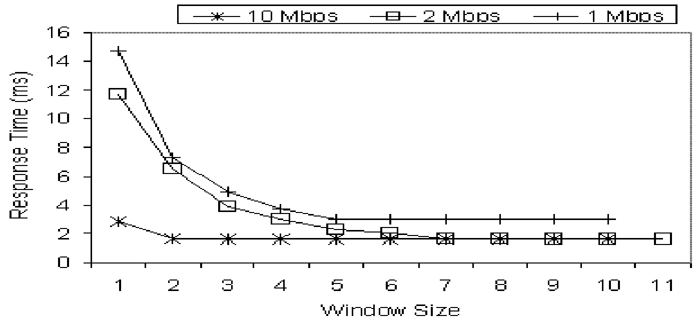

Fig. 4 (b) High bandwidth links

gradually up to 8 for 2 Mbps link. SWS for 10 Mbps LAN again drops to 3. With the sliding window the response time obtained with 2 Mbps link is equal to the response time with 10 Mbps link. This is because for higher speed links the agent processing time becomes the bottleneck rather than the network latency.

From the theoretical calculations we see that SWS should be ≤ 3 for networks of all speeds. However, due to additional delay experienced in the real networks, the SWS increases upto 9, this is explained in section *E*. The proposed mechanism has better response time upto 3 times for dialup, 6 times for 2 Mbps link and 1.5 times for 10 Mbps link compared to the stop and wait protocol. This improvement is more than the improvement computed theoretically, which is explained in section *E.*

B. *Performance optimization for getNext/getBulk*

Fig-5 shows the improvement in the response time with getRangeTable compared to standard SNMP getNext. The OID optimization gives further improvement in the response time. It has additional advantages like only the required range of entries are retrieved from the table.

C. *Performance optimization for getNext/getBulk with filtering*

This is an improvement over the getRangeTable having a generic filter using the getFilterTable request. Readings are taken considering different percentages of filtered output of 5000 entry subscriber table at various link speeds and the same is shown in the Fig-6. The first column indicates the round trip time with the existing standard SNMP with no filtering.

D. *Enterprise OID Optimization*

As discussed earlier almost all management operations on corDECT system are on the enterprise MIB. The average size of packet with and without enterprise OID optimization is measured for different numbers of OID. Fig-7 shows the sizes of optimized

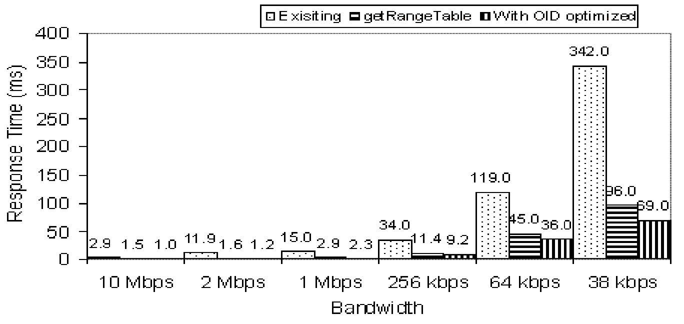

Fig. 5 Response time with “Multi-Responses”

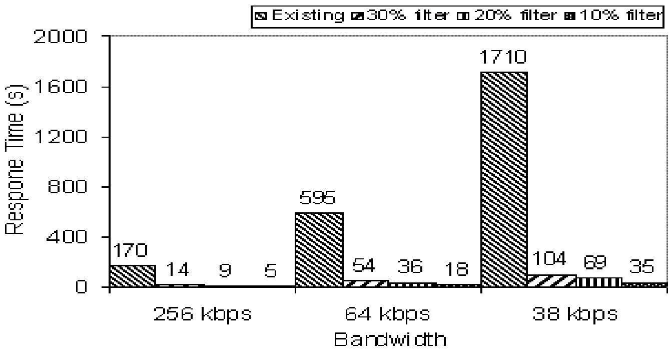

Fig. 6 Response time with different % of filters

and normal (non-optimized) packets. Table-2 shows the improvement in agent processing time with OID optimization.

E. Expected gain in response time vs. actual gain obtained

Fig-8 shows the comparison of expected response time and actual response time. From the figure we see that the pattern of actual gain matches with the expected gain but the actual gain is higher than the expected gain. This extra gain is obtained due to the network delay in the real network.

In our analytical calculation for the *R*, the propagation time of request, response packets and the processing time at agent are considered. *However in reality the actual propagation time is higher than the expected propagation time due to the network latency i.e., processing delay at the routers/switches in the network path.* Normally this network latency is significant for the first packet in the queue. When packets are sent back to back the network latency overlaps with the propagation delay and gets nullified. In stop-and-wait mechanism each request and response experiences the network latency, *while in the proposed mechanism only the first request and first response experiences this network latency*. Due to this we see that the actual gain obtained is higher than the gain expected theoretically.

VI. Conclusions

We have proposed techniques for optimization of network management traffic. This includes sliding window for get/set, multiple responses for geNext/getBulk, along with filtering and enterprise OID suppression techniques for optimization of network management response time and traffic. The proposed techniques are implemented and performance readings are taken. Analysis is done comparing the expected gain and the gain seen from the implementation. We see that the actual gain is higher than the expected gain due to the network latency. It is seen that the gain is high especially for low speed networks, favouring the needs of rural deployment. The work can be extended to do a further study to optimize the retrieved values. The implementation done for corDECT systems is ready for field deployment.

Table-2 Improvement in agent processing time

OID optimization	Agent processing time (ms)
Disabled	1.40
Enabled	0.95

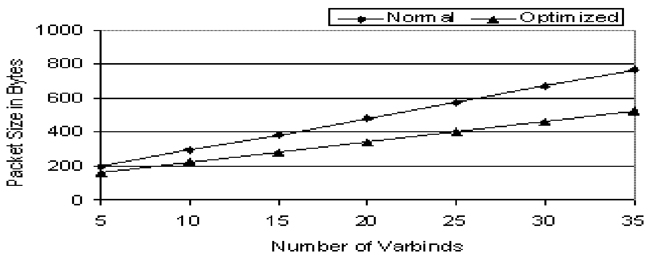

Fig. 7 Packet size comparison

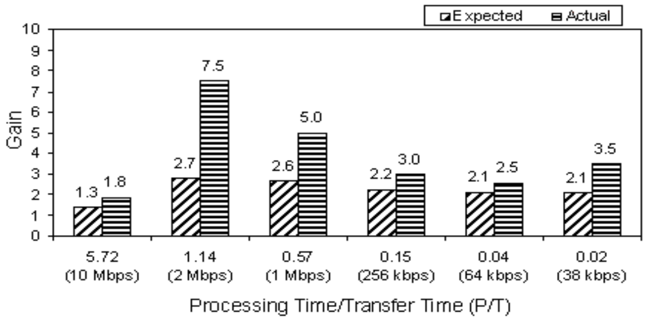

Fig. 8 Expected and Actual gain improvement

References

[1] W. Stallings, *SNMP, SNMPv2, SNMPv3 and RMON 1 and 2*, 3rd ed., Addison Wesley, 1999.

[2] R. Presuhn, J. Case, K. McCloghrie, M. Rose, S. Waldbusser, Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP), STD 52, *RFC 3416*, Dec 2002.

[3] U. Blumenthal, B. Wijnen, "User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)", RFC 3414, Dec 2002

[4] J. Schönwälder. SNMP Traffic Measurements. Internet Draft *draft-irtf-nmrg-snmp-measure-00.txt*, International University Bremen, May 2006.

[5] J.Schoenwaelder, SNMP payload compression, *draft-ietf-nmrg-snmp-compression-01.txt,* Apr 2001.

[6] S.McLeod, D.Partain, and M.White, SNMP object identifier compression, *draft-ietf-eosoidcompression-00.txt,* Apr 2001.

[7] A. Pras, T. Drevers, Rvd Meent, and D. Quartel, Comparing the Performance of SNMP and Web Services-Based Management, *IEEE electronic Transactions on Network and Service Management* 1(2), Nov 2004.

[8] M Rose, K. McCloghrie, J Davin, Bulk table retrieval with SNMP, *RFC 1187*, Oct 1990.

[9] R. Sprenkels, J.P. Martin-Flatin, Bulk transfers of MIB data, *The Simple Times* Volume 7 Number 1, Mar 1999.

[10] D. Gavalas, D. Greenwood, M. Ghanbari, M.O. Mahony, Advanced network monitoring applications based on mobile/intelligent agent technology, *Computer Communications*, Vol. 23 (2000) pp. 720-730, Apr 2000.

[11] Yen-Cheng Chen and Io- Kuan Chan, SNMP GetRows – An Effective Scheme for Retrieving Management Information from MIB Tables, *International Journal of Network Management*, Volume 17 (2007) pp. 51-67, Jan 2007.

[12] IANA, *Homepage of Internet Assigned Number Authority(IANA)*, http://www.iana.org.

[13] Midas Communications Technologies (P) Ltd, http://www.midascomm.com.

[14] Qiang Gu and Alan Marshall, Network management performance analysis and scalability tests: SNMP vs CORBA, *IEEE/IFIP Network Operations and Management Symposium*, Seoul, Korea, Apr 2004.