An Efficient Polling Layer for SNMP

M. Cheikhrouhou, J. Labetoulle
Institut Eurécom, Corporate Communications Department
BP 193 – 06904 Sophia-Antipolis Cedex – France
{*Morsy.Cheikhrouhou, Jacques.Labetoulle*}*@eurecom.fr*

Abstract

We propose the introduction of a Polling Layer that allows the efficient instrumentation of the management information models for the use of monitoring-intensive management applications. The polling layer allows to optimize the number of polling packets sent to the SNMP agents to collect management data. We describe the architecture of the polling layer that allows the management applications to specify their exact requirements for polling operations. We show how the knowledge of these requirements, combined with an efficient use of SNMP packets, lead to an important gain in the generated polling packets. This gain gets more important under heavy monitoring load. Finally, we show the advantages of the polling layer over different other approaches and we discuss its tradeoffs.

Keywords

SNMP, instrumentation, monitoring-intensive applications.

1. Introduction

The Network Management community is faced to an increasing shift in functionality requirements in the Network Management Systems (NMS). It is no longer sufficient to monitor the statuses of the physical components of the network and to display a red-colored icon when the number of errors increases on some device's network interface. Instead, the focus is currently shifted towards service and application-level management [1, 2, 3] which is closer to the network user perception than the component-level management. In addition, the network administrators are increasingly demanding integrated information models capable of describing the managed network in a high-level way (e.g. the Common Information Model [4]). The principle is that the network administrator is interested in managing a logical abstract entity regardless of where and how to extract the management information related to that entity .

Both of those factors are expected to impose an increased load on the monitoring activity of the NMS. If the monitoring of physical components requires only sparse polling to be performed now and then, the application- and service- targeted monitoring requires a lot of data to gather and with more frequent update. There are more data to collect because each entity in the network information model need to have all of its attributes up-to-date each time this entity is solicited by a management application. The polling frequency need to be high enough not to miss relevant changes or degradations in the application's behavior or in the server's availability.

This sensible increase in the monitoring traffic on the managed network could be intolerable. The management traffic should be kept to minimum compared to data traffic especially in the case of fault diagnosis or intensive network activity.

To anticipate this evolution, we developed a Polling Layer that optimizes the number of polling packets sent over the network. Management applications can dynamically request multiple variables to be polled while the polling layer ensures that only the minimal number of polling queries are sent over the network. In the case that multiple management applications are running in the same management system, these applications can start and stop the polling of any management data without caring about what data are being polled by the other applications.

Our Polling Layer is developed to be used for the Simple Network Management Protocol (SNMP) [5] which is by far the most used management protocol in corporate networks and in the Internet community in general. Unless a new management protocol and instrumentation is agreed on to replace SNMP (which is far to be accomplished at least in the near future), there is no other standard that allows to collect management data with less cost than SNMP.

The paper continues in Section 2. with an enumeration of the requirements that the polling layer has to fulfill. Section 3. presents the polling layer architecture that allows to fulfill its requirements. It also details the different optimization mechanisms. Section 4. presents a performance assessment by considering the number of polling packets sent with and without using the polling layer. It shows an important gain of polling packets as the polling activity gets more and more intensive. Section 5. provides a qualitative discussion of the polling layer, mainly by comparing it to other related approaches. Finally, Section 6. concludes the paper.

2. Requirements for the Polling Layer

The shift from a component-level management information model to a service-level model will affect the requirements of the management applications running upon an NMS. Obviously, a service-oriented management function would require more data to collect from the SNMP agents than a simple component-oriented application. However, other factors have to be considered also:

Monitoring-Intensive Applications Increasingly, management applications are supposed to be more aware and predictive about the status of the network services. Therefore, they are becoming more and more monitoring-intensive, i.e, requiring more data to collect with increased precision. Therefore, the polling layer has to make efficient use of polling packets. This can be achieved by including multiple variables within the same polling packet whenever this is possible.

Multiple Management Functions In an NMS that offers many functions, it may happen to have several management functions requiring the same management data to be collected from the network. Hence, an optimized polling layer should be able to combine polling requests coming from different management applications if these polling requests concern the same data to be collected.

Intensive Polling It is possible to have polling peaks during certain periods. This may happen when many management applications are running at the same time, or when the network administrator is performing an intensive management activity to diagnose a certain misbehavior of the network services and applications. The polling layer should be able to handle these peaks in the polling activity and to keep the generated monitoring traffic as low as possible.

Polling Characteristics The polling characteristics may differ according to the usage to be made of the polled value. Management applications should be able to specify the "quality" of the polling they need for each of the variables to monitor. Different usages of variable values may require different polling characteristics. The polling characteristics will be detailed later in section 3.2.

3. The Polling Layer

Modern Network Management Systems and platforms rely on an information model of the managed network. Basically, this model is composed of objects with attributes (or properties) that have to be instrumented out of collected managed data from the network. In order to fulfill the requirements mentioned in the previous section, we propose to provide a separate Polling Layer that allows to efficiently poll data from the network. The polling layer is introduced between the management applications and the SNMP-managed network components. Its role is to provide the object attributes at the application level with their respective values in order to have these attributes regularly updated with fresh values. The aim is to give the network administrator the transparent and integrated view of the managed resources regardless of where and how their related information are gathered.

The remainder of this section details the design of the polling layer as well as the different optimizations performed.

3.1 General View

The polling layer has two interfaces, as shown in Figure 1. The lower interface interacts with the SNMP agents deployed in the network elements. As such, the polling layer ensures the manager role vis-a-vis those agents. It is responsible for the polling of the required variables on these agents and communicating the received values to the object properties in the application-level layer at the upper interface. Each instrumented object property has an associated "*Image*" of the corresponding polled variable that fits exactly the usage to be made of that property in the management application. The main concern is that the management application uses the instrumented object in a transparent way as if it were the object itself that performs the update of its attributes.

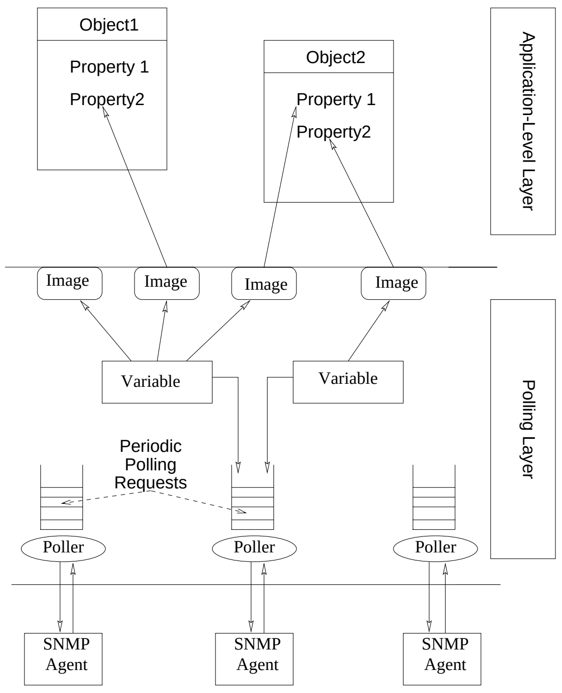

Figure 1: The Architecture of the Polling Layer

The concept of an "image" is introduced to reflect that the same variable on an SNMP agent can be used in different manners, possibly by different applications, and therefore, should be polled with different strategies or periods. The way an image reflects its associated variable is described using a certain number of parameters called the "*Usage Parameters*". These parameters are detailed in the following section.

3.2 The Usage Parameters

The way a variable is polled depends on the usage to be made of its value. For example, an instrumented object's attribute used to perform precise statistical computations on a network parameter does not need the same type of polling as another attribute that is solicited only occasionally by the management application. For this reason, one of the main functions of the polling layer is to provide usage-adapted *images* of the variables polled from the SNMP agents. Each object attribute that requires a variable with a certain usage type is associated to an *image* that is exactly suitable for that usage. The polling layer is responsible for the permanent compliance of this image to the requirements of the instrumented attribute.

We identified a set of parameters that describe how a variable is polled.

- **Freshness Interval Δt:** It is important for an object property to have a fresh value available each time this value is required for a management task. Depending on several parameters, the freshness interval for an attribute may vary from few seconds to several hours. This interval is to be decided according for example to the importance or the precision of the task involving this data.

- **Freshness Precision ε:** This is a precision over the freshness interval Δt. Since there is no guarantee on the time at which an SNMP query is processed by an agent, it is not possible to have the exact polling period between each consecutive pollings. The freshness precision allows to specify when a polling request can be accepted or not. In general, if a polling query is supposed to be achieved at time $t = n.\Delta t$, then the polled value is accepted only if the SNMP agent processes the query in the interval $[t - \varepsilon, t + \varepsilon]$.

- **Offset δ:** In order to maximize the chance of having many variables to be polled at the same time, it is necessary to refer to the same time origin for all the pollings. All the polling requests are synchronized to this origin, which we denote by t_0. However, there are cases in which it is necessary to add an offset δ to t_0.

  Consider for example a large polling period of, let us say, one day, but for which the polling has to be performed at a specific hour, e.g. 8 am. If t_0 refers to the midnight hour, then it is necessary to specify an offset of eight hours in order to have the polling occur at the desired time.

  The offset can also be used for a more sophisticated reason. Take a large network of 10,000 nodes that have to be polled for example each day. If all the pollings are synchronized to t_0, then each day the management application will have to send 10,000 polling requests at the same time. In addition to the processing overload due to the emission and the reception of polling packets and their replies, the management station will be overwhelmed with a burst of replies within a short period and the network may be congested causing packet losses. This "Feedback Implosion" problem can easily be overcome by dividing the one-day period to, let us say, 1440 time slots separated by a one-minute sub-period. At each slot, no more than five network nodes are polled. In this case, the network bandwidth usage as well as the processing time are spread out over the polling period.

- **Update Method:** This parameter tells whether and when the image should update the object attribute with the new value. Three update strategies are identified:

1. In some cases, the attribute prefers to have the freshest possible value. The image has to update the attribute's value whenever a new value is recorded, even if the attribute still has a fresh value regarding the required freshness interval. This kind of update is useful in many cases in order to take profit from the variable pollings achieved for other images. This kind of update strategy is called *eager update*.
2. In other cases, the attribute specifically requires its value to be updated only each Δt period. In this case, even if an intermediate value is obtained, the attribute is not updated. This kind of strategy is useful when it is necessary to perform arithmetic calculations on a set of attributes, and therefore, the values of the attributes should all be considered at the same time to have a coherent result. This strategy is called *periodic update*.
3. Finally, the attribute may not require to be updated unless it explicitly asks for it. This can be suitable for attributes that are only rarely solicited. For example, the operating system version running on a machine can be rarely required. When it is required, even if there is no available fresh value, it can be fetched by initiating a separate request. This kind of update strategy is called *lazy update*.

- **Availability:** This parameter allows to select one of two refreshment strategies. In the first strategy, the application's attribute is supplied with a fresh value at least as soon as the last polled value is out-of-date regarding Δt. In the second strategy, the attribute will not be refreshed unless explicitly accessed by the management application. In this case, the last polled value is provided unless it is out-of-date. If the last polled value is out-of-date, then a polling is initiated to retrieve a new fresh value.

  While the first strategy allows the application to have a prompt fresh value each time the attribute is accessed, the second strategy may introduce more delays in the application's response time while ensuring that no polling is generated unless explicitly needed.

3.3 Architecture of the Polling Layer

The structure of the polling layer is shown in Figure 1. It's components are the *images*, the *periodic polling requests*, the *variables* and the *pollers*.

Images

The image holds a representation of the polled variable that conforms to the usage parameters specified. The role of the image is to verify whether the polled values need to be themselves communicated to the object attribute or not. The image has also to constantly verify whether its usage parameters are satisfied or not. If they are violated, then the management application is notified with a warning telling the reason of the violation: network-level problem, SNMP-related error, etc.

Periodic Polling Requests

In order to provide the images with fresh values that meet their usage parameters, periodic pollings have to be performed. The characteristics of periodic pollings are encapsulated into *Periodic Polling Request* objects. A periodic polling request holds specific values for the polling period Δt, the freshness precision ε and the offset δ.

Variables

A *variable* instance in the polling layer is associated to a MIB variable polled from a determined SNMP agent. The variable performs a first step of polling optimization by merging the usage parameters of its different images and producing the minimum number of periodic polling requests to be achieved by the poller. If the variable has already enough images whose periodic polling operations can satisfy the newly submitted image, the variable does not ask for a new periodic polling operation. This optimization process is described in Section 3.4.

Pollers

The poller is viewed as a server that performs periodic polling operations submitted by the variables. It is responsible for the actual sending of SNMP polling packets. By executing an optimization algorithm which is presented in the next subsection, it can group as many polling requests as possible within the same sent packet. This optimization algorithm is described in the following section.

3.4 Optimizations

Poller-Level Optimization

A Poller accepts two kinds of polling requests: periodic polling requests, i.e. pollings that have to be achieved each period, and on-demand requests, i.e. single pollings to be achieved only once. A periodic polling can be identified by a tuple $\Pi = (\Delta t, \varepsilon, \delta, Oid)$, where Oid is the full MIB Object Identifier of the variable to be polled. The purpose of the poller is to include the maximum number of variables within a minimum number of SNMP PDUs. This is achieved by choosing the suited time to send the PDU. For example, in the case of Figure 2 in which three pollings are ready to be sent, the darkened time zone is the only time interval in which the three polling requests can be included in the same packet. The time interval $[\alpha, \beta]$ within which the poller must send the next polling request is calculated as follows.

Firstly, given the current time t, the next polling interval for a periodic polling $\Pi_i = (\Delta t_i, \varepsilon_i, \delta_i, Oid_i)$ is the interval $[\alpha_i, \beta_i]$ where $\alpha_i = \delta_i + \lceil \frac{t}{\Delta t_i} \rceil \Delta t_i - \varepsilon_i$ and $\beta_i = \delta_i + \lceil \frac{t}{\Delta t_i} \rceil \Delta t_i + \varepsilon_i$.

Secondly, the time at which the request should be sent cannot ne later than $\beta = Min_i\{\beta_i\}$. For example, in the case of Figure 2, $\beta = \beta_1$. Therefore, all the pollings having $\alpha_i < \beta_k$ could be included in the same polling request. Consequently, the

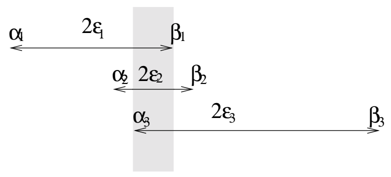

Figure 2: Time interval in which the polling query should be sent

polling query should be sent at $\alpha = Max_i\{\alpha_i, \alpha_i < \beta_k\}$, that is $\alpha = \alpha_3$ in the case of Figure 2. Preferably of course, the polling query is sent at the beginning of the time interval.

The poller automatically includes the `sysUptime` MIB-II variable, which is essential if cumulative variables are being polled.

Variable-Level Optimization

The Variable is responsible for generating and submitting the required periodic pollings to the poller in order to satisfy the usage parameters of its different images. Thanks to the explicit specification of the usage wanted from an image, the variable is capable of generating just the minimum number of periodic polling requests. As a matter of fact, the variable perceives a new image as a tuple $(\Delta t, \varepsilon, \delta, U, A)$ where U is the update method and A is the availability of the image value.

What can be noticed, is that except in the case that a periodic update is required, the image could benefit from pollings having shorter periods of freshness Δt. In fact, using a more frequent polling will only refresh the image more often. In the contrary, if the periodic update is required for the image, then a new periodic polling request should be submitted to the poller in order to have the value of the variable refreshed at the exact period. Consequently, a set of images requiring either no update, or update for any new variable value, can be satisfied with a unique periodic polling request with a period equal to the smallest freshness period required by these images. These optimizations hold if the availability parameter A requires a prompt fresh value each time the image is accessed.

If the availability A does not require a prompt value for the image, then no periodic polling is necessary. Each time the image value is required, the image checks itself whether its last recorded value is still fresh or not. If it is not fresh, the image asks to perform an on-demand polling.

SNMP-level Optimization

This is a simple optimization that allows to remove duplicated variables that are already included in the polling PDU. This optimization is necessary because the same variable may be required by two distinct images.

Table Handling

In the first version of SNMP (SNMPv1), it was not possible to perform bulk retrieval of variables on an agent [5]. The only possible way to retrieve tables and to perform a “walk” on a MIB is by sending successive `GetNextRequest` packets. This remains the basic way to poll tables in SNMP agents since the new SNMP versions that support the bulk retrieval are not yet deployed except in a limited number of newly-acquired devices.

Most of the management platforms existing today retrieve tables by performing successive `GetNextRequest`s starting from the table root. This is inefficient since it requires as many polling requests as elements in the table. In addition, the management applications rarely require all the data of a table. Instead, an application is in general interested only in a few number of columns. For example, an application monitoring the `interfaces` of a certain host may not need to poll the `ifDescr`, `ifSpeed` and `ifMtu` columns since they have constant values.

The way tables are handled in the Polling Layer is through columns. The application declares what columns are needed for the instrumentation of its objects so that only these columns are polled. The poller performs the polling of multiple columns in parallel by using the same `GetNextRequest` packets. If the columns belong to the same table, then each `GetNextRequest` allows to retrieve a row of this table. Therefore, the maximum number of requests sent to poll a table is equal to the number of table rows. Columns from different tables can be polled using the same `GetNextRequest`. As soon as the rows of a table are completely retrieved, its columns are removed from the `GetNextRequest` packets. The process continues until the columns of the table with the greatest number of rows is completely polled.

Therefore, the knowledge of the exact information required by the management applications allows to sensibly reduce the number of requests required to poll tables. For example, the usual interfaces table of a desktop computer having two interfaces is retrieved in only two packet exchanges regardless of the number of columns required by the management application.

4. Performance Results

In order to evaluate the performances of our polling layer, we randomly submitted images requests to a particular SNMP agent and measured the number of polling packets sent during five-minute periods. We measured the same number of packets that would have been sent if the polling layer had not been used, i.e. without optimization. Submitted images had random values for the different usage parameters and were cancelled after a random time interval. The freshness precision ε is chosen randomly between 5% and 15% of the freshness period Δt.

At a first stage, we show in Figure 3 the results when freshness periods are random multiples of ten seconds. We considered both cases, under a normal low monitoring demand (on the left side), and under a heavy monitoring demand resulting from an intensive use of the management application (on the right side). Under a low polling demand, we notice that the polling layer performs at most the same number of transmitted polling packets as without optimization. In fact, during the time interval between $t = 1000$ and $t = 1300$ (five hours), we had 74 polling packets without optimization, and 66 packets when using the polling layer, which implies a gain of 10% nevertheless.

The right side of Figure 3 shows how the polling layer behaves compared to a non-optimized polling. During the time interval between $150mn$ and $200mn$, an intensive monitoring activity is started. We can easily notice how the number of sent polling packets were cut nearly by half when the polling layer is used. Precisely, between $t = 150$ and $t = 200$ we had 620 packets without optimization, and 278 packets with optimization. This implies a gain of more than 55%.

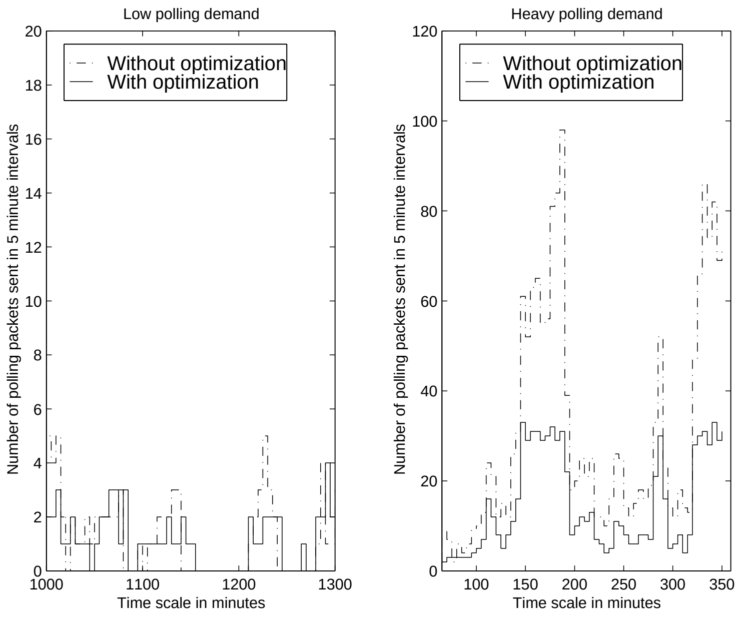

Figure 3: Polling optimization with freshness periods multiple of 10 seconds

However, choosing Δt as a multiple of 10 seconds is rather pessimistic. In practice, the polling freshness periods are expressed as multiples of higher time units. Unless a very detailed sampling is required, the polling periods take regular values such as 1mn, 2mn, 5mn, 15mn, 30mn, 1 hour, etc. In order to study the impact of the freshness granularity, we performed similar measurements as above, but with Δt chosen as multiple of $30sec$ instead of $10sec$. We obtained the results depicted in Figure 4.

The left side shows the difference of the numbers of transmitted polling packets under a medium monitoring demand. The figure shows a clear gain compared to the case of Figure 3. At time $t = 300$, we registered 933 packets for a non-optimized

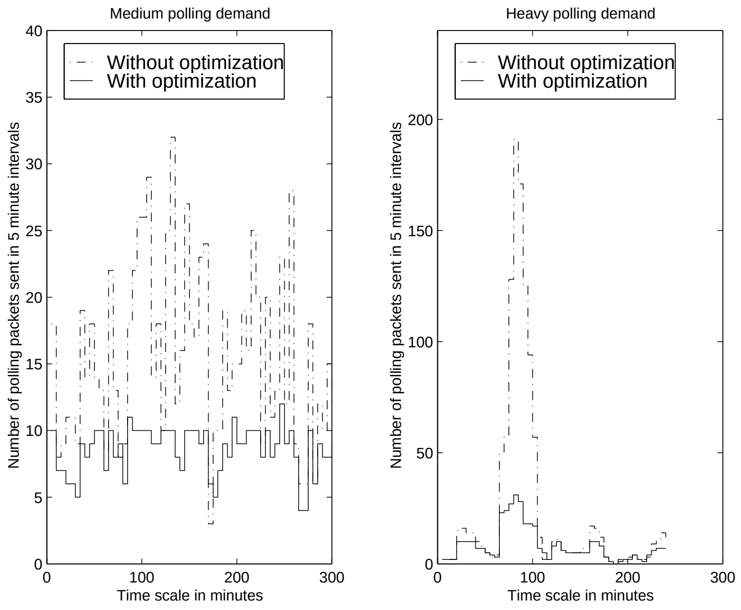

Figure 4: Polling optimization with freshness periods multiple of 30 seconds

polling versus 513 packets using the polling layer. This implies a gain of 45%.

The right side of Figure 4 shows the case of a burst of a heavy monitoring demand. In this case, the polling layer offers a very important gain. Between $t = 60$ and $t = 100$, we registered 821 packets without optimization and 172 packets with the use of the polling layer. This presents a gain of 79%.

These results show that the polling layer reduces the total number of required polling packets sent to the SNMP agent. At the worst cases, it would produce the same number of packets as without optimization. These worst cases occur when the monitoring demand is very low. As soon as the polling demand grows, the polling layer provides a considerable gain. What is important from a network administrator viewpoint is that in the case of an intensive monitoring activity such as a network failure diagnosis or a precise performance measurement, the network will not be overloaded with management traffic which could worsen the network failure or misguide the performances results.

5. Discussion

Some of the optimization mechanisms we used in the polling layer are not new for SNMP-based management. For example, parsing MIB tables in columns was mentioned in [6] and [7]. There are also many other approaches to reduce the SNMP traffic. Most of these approaches are revolutionary and intrusive. For example, the works described in [8] and [9] make use of the Mobile Agent technology enabled by the Java programming language. However, Mobile Agents require an additional distributed run-time environment to be installed on the hosts where the agents may migrate. Moreover, many network components cannot run the Java Virtual Machine yet. Other works suggest to improve SNMP with new capabilities or even to replace it in favor of other protocols such as HTTP. A leading work in this direction is described in [10] and [11]. However, such suggestions can be considered only for the long term since the SNMP agents which are already deployed in a large number of network devices cannot be replaced in the short term. Our approach is not intrusive and do not require any modification on the existing SNMP-instrumented devices and software.

A comparable approach was described in [12] and then further elaborated in [13]. Basically, the polling is governed by a caching model that defines different update policies. Major defined update policies include a *periodical update* with a given interval, a *lazy update* for which no polling is generated unless the variable value is explicitly requested, and a *lazy update with latency* which is a lazy update that generates a polling only if the last update time exceeds a certain period of time. In [13], more refined update policies are introduced based on the deviation of the variable value. In our opinion, the optimization of polling traffic according to the evolution of the polled variable, and more generally, by predicting the future changes of its value (e.g., see [14]) is costly in terms of resources. The polling layer must be able to keep information on the past behavior of the polled variable and to infer its future evolution. This will lead to huge memory and processing usages when the number of polled variables increase.

The polling layer we developed is also based on the concept of usage-based polling in which the polling activity is adapted to the management applications' evolving requirements. These requirements are expressed using the usage parameters introduced in subsection 3.2. The usage parameters allow for a finer control on the polling activity. For example, the polling precision ε and the offset δ have no equivalent in the above-mentioned work ([12] and [13]). In addition, our polling layer combines this usage-based polling with an optimization at the SNMP packet level which is performed in the poller component described in subsection 3.3. The combination of these optimizations lead to an important reduction of the polling packets sent over the network.

There are however some remaining issues for our polling layer. An important issue is that an SNMP PDU has a limited size. If the number of the polled variables is large, the reply from the SNMP agent may not fit into the PDU and the whole packet is therefore lost. It is difficult to estimate the number of variables that can be included within the same packet. This number depends on the size of the Object Identifier as well as on the variable type. However, during the performance measurements that were run over long periods, this problem never occurred. We successfully could include 20 string variables within the same packet without causing a response overflow. A more rigorous approach will be to make the poller component predict a pessimistic size of the response PDU according to the types of the polled variables as described in the MIBs.

Another factor that may cause the polling packet to be lost is that in the case of an SNMP-level error in any variable in the `GetResponse` or `GetNextResponse` packet, a null value is returned for all the other variables by the SNMP agent. This is particular for SNMPv1 and was fixed for the later versions.

6. Conclusion

We proposed to introduce a polling layer that ensures that the polling operations are performed in a way that minimizes the use of network resources. A key enabling factor is the introduction of the usage parameters which define how an application requires the image of a polled variable to be and how it is to be used. By declaring the usage parameters of each image, the polling layer is capable of generating only the minimum number of polling queries to the SNMP agents. The performed measurements show that at the worst cases, the polling layer provides at least a small gain compared to a non-optimized polling. Importantly however, in the critical periods, i.e. when the monitoring activity is very intensive, the polling layer provides a large gain. This is a great advantage because the SNMP agents still receive only an acceptable number of queries even when they are heavily polled.

The polling layer can be deployed in a very flexible way. Being completely written in Java, it can be easily integrated with existing management systems as an intermediate polling server. It could also be integrated in middle-level manager agents in a hierarchical management system.

In our team, the polling layer is to be used to ensure the instrumentation of high-level information models of several management applications. These include applications of Virtual Reality for Network Management ([15]) and applications of distributed intelligent agents.

References

[1] Raúl Oliveira, Dominique Sidou, and Jacques Labetoulle. Customized network management based on applications requirements. In *Proceedings of the First IEEE International Workshop on Enterprise Networking - ENW '96*, Dallas, Texas, USA, June 27 1996.

[2] Patricia G. S. Flofissi, Yechiam Yemini, and Danilo Florissi. QoSockets: a new extension to the sockets API for end-to-end application QoS management. In Sloman et al. [16].

[3] Peter Parnes, Kare Synnes, and Dick Schefström. Real-time control and management of distributed applications using IP-multicast. In Sloman et al. [16].

[4] CIM specification v2.0. http://www.dmtf.org/cim/index.html, March 1998.

[5] William Stallings. *SNMP, SNMPv2 and RMON, Practical Network Management*. Addison-Wesley, USA, 1996.

[6] Jean-Philippe Martin-Flatin and Ron Sprenkels. Bulk transfers of MIB data. *The Simple Times*, 7(1), March 1999.

[7] Nikos Anerousis. An information model for generating computed views of management information. In *Proceedings of the 9th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, Newark, DE, October 1998.

[8] Michael Zapf, Klaus Herrmann, Kurt Geihs, and Johann Wolfgang. Decentralized SNMP management with mobile agents. In Sloman et al. [16].

[9] B. Pagurek, Y. Wang, and T. White. Integration of mobile agents with SNMP: Why and how. Submitted to NOMS'2000, 2000.

[10] Jean-Philipe Martin-Flatin. Push vs. pull in web-based network management. In Sloman et al. [16].

[11] Jean-Philippe Martin-Flatin, L. Bovet, and Jean-Pierre Hubaux. JAMAP: a web-based management platform for IP networks. In *9th IFIP/IEEE International Workshop on Distributed Systems: Operations & Management (DSOM'99)*, Zurich, CH, October 1999. Available at http://ica2www.epfl.ch/˜jpmf/papers/dsom99.pdf.

[12] F. Stamatelopoulos, N. Roussopoulos, and B. Maglaris. Using a DBMS for hierarchical network management. Engineer Conference, NETWORLD+INTEROP'95, March 1995. http://www.netmode.ece.ntua.gr/papers/interop_e16.eps.

[13] Fotis Stamatelopoulos and Basil Maglaris. A caching model for efficient distributed network and systems management. In *3rd International Symposium on Computers and Communications - ISCC'98*. IEEE, July 1998. http://www.netmode.ece.ntua.gr/papers/iscc98.ps.

[14] Jia Jiao, Shamin Naqui, Danny Raz, and Binay Sugla. Minimizing the monitoring cost in network management. In Sloman et al. [16].

[15] P. Abel, P. Gros, D. Loisel, and J. P. Paris. Virtual reality and network management. In *7ème journées du GT Réalité Virutelle*, June 1999.

[16] Morris Sloman, Subrata Mazumdar, and Emil Lupu, editors. *Integrated Network Management VI: Distributed Management for the Networked Millennium*, IFIP/IEEE International Symposium on Integrated Network Management, Boston, MA, USA, May 1999. IEEE Publishing.