J Netw Syst Manage (2009) 17:105–136
DOI 10.1007/s10922-009-9125-5

On the Performance and Scalability of Web Services for Monitoring MPLS-based Networks

Aimilios Chourmouziadis · Marinos Charalambides · George Pavlou

Published online: 15 May 2009

Abstract Web Services is an XML-based technology that has attracted significant attention for building distributed Internet services. Considerable research efforts have been trying to extend it to become a unifying management technology; initial evaluation efforts have demonstrated promising application potential in the domain of network and service management. This paper analyzes the monitoring requirements of a management system used in architectures for providing quality of service, and shows how Web Services can be used effectively to fulfill these requirements. In order to achieve this, a WS-based framework was designed and a query tool was developed to support distributed, optimized retrieval of management information exposed as Web Services. The query tool supports selective and bulk retrieval from MPLS-capable routers. Management information retrieval is demonstrated through three monitoring scenarios, and the measured performance in terms of latency and bandwidth is compared against a conventional approach based on SNMP.

Keywords XML · MPLS · Web services · SNMP · Monitoring · Management

A. Chourmouziadis (✉)
CCSR, University of Surrey, GU27XH Guildford, UK
e-mail: A.Chourmouziadis@surrey.ac.uk
URL: http://www.ee.surrey.ac.uk/CCSR/

M. Charalambides · G. Pavlou
University College London, WC1E 7JE London, UK

M. Charalambides
e-mail: M.Charalambides@ee.ucl.ac.uk
URL: http://www.ee.ucl.ac.uk/

G. Pavlou
e-mail: G.Pavlou@ee.ucl.ac.uk
URL: http://www.ee.ucl.ac.uk/

1 Introduction

Previous management technologies such as the Simple Network Management Protocol (SNMP), the Common Object Request Broker Architecture (CORBA), and the Common Management Information Protocol (CMIP) are well established and have already found their niche markets. While all these technologies have gone a long way trying to address management objectives efficiently [1], a new candidate has also made its appearance. Web Services (WS) is an emerging XML technology whose promise of faster development, interoperability, application integration and industry acceptance has led researchers to also consider it as a potential management technology.

While investigating the potential of WS for network and service management, researchers in [2, 3] evaluated the performance of WS against traditional management technologies such as SNMP. Although their results suggest that WS bear a considerable improvement in terms of latency and bandwidth consumption when retrieving large amounts of data, performance evaluation depends heavily on the tools used to deploy and manage WS. Such tools are for parsing, serialization and compression [4–6]. It was only recently, for example, that suggestions in [5] to parse XML schemas only once in order to minimize encoding latency have been adopted by the JAVA API for XML Processing (JAXP). Additionally, *pull parsing* techniques suggested in [6] to optimize WS performance in cases where the XML elements of a document need to be accessed in succession or when some elements have been parsed before, have been adopted by the Apache AXIS 2.0 and Codehaus XFire toolkits. These SOAP toolkits use the Streaming API for XML processing pull parser (StAX) to build a partial XML Infoset tree in memory, allowing applications to start processing XML content even before the entire document has been parsed, thus increasing performance. Furthermore, Gzip, one of the standard but not very fast tools for compression used with SOAP, continues to be popular compared to differential encoding schemes that have been recently gaining ground [7]. In addition to parsing and compression, WS performance suffered from conceptual limitations. The use of MIME encodings to serialize binary data increases processing cost, code size and decreases performance. The use of DIME binary encodings as suggested in [5] to overcome these problems, are now supported by the Java API for XML Web Services (JAX-WS), the latter being supported by AXIS 2.0 and XFire.

By adopting all the above optimizations the performance of SOAP toolkits has increased. Currently, some of the fastest toolkits available are Apache AXIS 2.0 and XFire. In a comparison between AXIS 2.0 (v.1.4) and AXIS 1.3, the authors of [8] showed that using StAX, and the ADB (Axis Data Binding) or the XML Beans binding frameworks for serializing data, has improved the latency performance of AXIS 2.0 by 4–5 times compared to AXIS 1.3. Additionally, the same authors in [9] have shown that the ADB framework is better compared to the JAXB framework and this is the main reason behind the superiority of AXIS 2.0 over XFire.

The performance of WS though is not just influenced by the parsing and serialization techniques used in SOAP/WS toolkits, but is also affected by the processing required to handle the XML infoset. Recent work in [10] has shown that

the performance of the XML Path language (XPath) used to search and process management data stored in XML is worse compared to a query toolkit that processes raw data instead. In [11] we have shown that the overhead introduced by processing management data is equally and in some cases even more important than the parsing and serialization overhead (i.e. XPath implementations introduce 10–16 times more latency than the raw data query tool in cases where the volume of information to be processed is large). It is thus evident that the performance of tools to process management data can affect greatly the performance of monitoring and event reporting operations.

Motivated by the fact that the performance of WS has not been appropriately investigated and compared against other technologies in scenarios where information processing is required, a study was performed in order to: (a) look into what has been done so far in the field of WS-based monitoring and event reporting; (b) investigate what facilities, such as bulk and selective retrieval or facilities for distributing monitoring load and tasks, can offer in these two fields; (c) examine if it is plausible to offer such facilities using WS or other technologies and whether they can help in addressing monitoring requirements; and (d) test the performance of these facilities based on a real monitoring and event reporting scenario.

As it would be difficult to investigate the performance of polling based monitoring for all available management technologies in the context of this paper, we chose to limit the scope of our work to SNMP and WS. Event reporting, which is also an important aspect of monitoring has been examined in [12] where the IBM WS-Notification standard [13] has been used to compare WS notifications to SNMP traps. In contrast to previous research, this paper aims to accomplish the following:

- Study the scalability of WS and SNMP for polling based monitoring when load/task distribution and information processing is performed using tools for bulk and selective data retrieval.
- Study if current WS tools are suitable for load/task distribution and information processing. As described in Sect. "4.2", new lightweight tools may have to be designed since currently available facilities are not suitable for this purpose. We have designed such a tool and developed a framework for distributed monitoring to support it.
- Investigate the scalability of WS and SNMP using a real case study where distributing tasks and operations for bulk information retrieval, filtering and data processing may be required. The scenarios chosen are based on a network providing QoS over MPLS enabled devices. As such, we examine: (a) what measurements are required for QoS; (b) how these measurements are performed with specific SNMP Management Information Bases (MIBs) and WS; and (c) identify measurement scenarios requiring bulk retrieval, filtering and data processing.

The remainder of this paper is structured as follows. In Sect. "2", we provide a review of related work in the field of monitoring with WS and SNMP. This section also includes a discussion on tools for bulk/selective information retrieval and information processing. Section "3" investigates QoS monitoring over MPLS enabled devices. This section analyzes three scenarios where relatively complex

monitoring tasks and information processing are required. Based on these scenarios we investigate the performance of the tools described in Sect. "2" for WS and SNMP. Section "4" discusses how to perform the measurements with WS and SNMP for the three QoS scenarios. In Sect. "5" we provide details about the experimental setup and the scenario measurements that involve latency and traffic overhead. Finally, Sect. "6" presents our conclusions.

2 Related Work

2.1 Distributed Monitoring with Web Services

Over the past few years, XML has proven capable of providing the means to build interoperable applications. Two such examples are the Distributed Management Task Force's (DMTF) Web Based Enterprise Management (WBEM) collection of specifications and the OASIS (IBM, HP) WS Distributed Management (WSDM) group of documents. From the first group of specifications the WS for Management (WS-Management) specification [14] is of most notable importance. This document specifies how to identify a manageable resource and how to initiate communication with it. WS-Management depends on other specifications such as XML Schema to describe the content and structure of XML documents [15, 16] and the WS Description Language (WSDL) to describe network services. Additionally WS-Addressing is used to identify the endpoints where services are offered, WS-Eventing [17] for subscribing or receiving events and WS-Enumeration [18] for reading event logs and other data collections. Moreover WS-Transfer [19] is used as a general SOAP-based protocol using various communication models for accessing XML representations of Web service-based resources, and WS-Security offers mechanisms for message integrity, message confidentiality and single message authentication [20]. The WSDM group on the other hand has issued two specification documents; (a) the Management Using WS (MUWS) specification [21], and (b) the Management of WS (MOWS) [22] specification. The former describes the use of WS for resource management and the latter describes the management of WS endpoints through WS protocols. These specifications roughly depend on the same standards as WS-Management, with the exception of WS-Eventing which is replaced by the WS-Base Notification standard [13]. Both the MUWS and WS-Management standards define operations and mechanisms to perform WS management with the latter being more lightweight. Efforts are currently underway to merge these two standards.

The above two standards (MUWS, WS-Management) have originally been designed for service management but can be tailored to address the needs of network management. Such an example can be found in [23] where the authors use these standards to simulate SNMP operations (get, getBulk) and test their performance against the latter when performing polling based monitoring. SNMP operations can be supported by both standards but with a different degree of complexity. Through native operations MUWS provides a simplified way to simulate SNMP operations. This is not the case for WS-Management [23]. Another

way to simulate SNMP with both approaches is through XPath. The WS-Management mechanism though that supports these expressions, called *fragment level transfer*, is optional. This allows applications implementing the specification to omit this feature leaving only non practical ways to simulate SNMP. MUWS supports XPath expressions by default through the *QueryResourceProperties* operation and thus is more flexible to simulate SNMP. XPath expressions in both standards can also be used to support bulk or selective retrieval facilities. In this investigation the authors found these frameworks more heavyweight than SNMP. Nevertheless, the authors do not investigate the potential of query tools with either standard. Instead they focus on emulating SNMP, based on the assumption that such mechanisms will result in performance penalties on WS monitoring operations, thus inhibiting the performance of WS even more in comparison to SNMP.

In addition to [23] substantial research effort has been directed in the past to assess the performance of WS-based network management. In [24–26], authors investigate different approaches to support the management of SNMP legacy devices by WS managers. WS to SNMP gateways are examined and their performance is evaluated. In [2, 3] the performance of a pure WS framework is compared against that of SNMP and CORBA for polling based monitoring. Both authors support that SNMP's performance for polling based monitoring is better than that of WS especially for a small volume of data. This may not be true as this work, and also [23] do not investigate the potential of distributing load and tasks to end devices for polling based monitoring. In [2, 23] authors assume their WS framework implementation will become more resource intensive compared to SNMP if extra facilities are added. We do not share this opinion and, as it will be demonstrated later, facilities for bulk and selective retrieval to distribute monitoring load and tasks can minimize the footprint of WS management applications. Apart from the performance of WS and SNMP for polling based monitoring the scalability of WS notifications has been investigated in [12, 27]. In [27] and also [12] event reporting is examined in a management by task delegation environment. In [12] we use the WS-Notification standard to define policy rules on the actions that need to be performed when certain conditions occur. The performance of our task delegation approach is compared to SNMP traps and is found to be better, even when retrieving small amounts of data, contradicting the conclusions of [2, 3].

To the best of our knowledge, the research community has not yet examined the impact on scalability when the monitoring load is distributed in the context of polling based monitoring and where management information processing is required. We believe that processing overhead is equally and in some cases more important than the serialization and encoding overhead for monitoring and event reporting operations. This has been the main motivation behind our work which investigates the feasibility and performance of tools used to process management data to offer bulk and selective data retrieval capabilities using WS. The most obvious selection of such tools to realize the required functionality would be software implementations of the MUWS specification along with XPath 1.0 [28] or 2.0 [29]. Concerns in the NetConf community [30] regarding memory and latency overheads of XPath, led us to design and develop an information retrieval query tool and parser [31] for monitoring and event reporting. It should be noted that the above

concerns have been verified by [32, 33]. In a monitoring and event reporting case study of the query tool in [10, 11] we demonstrated that its performance is superior to both XPath 1&2. Since this query tool is not supported by MUWS, the choice of framework should not compromise interoperability and performance. For this reason we have developed a new framework for distributed monitoring with a well accepted WS toolkit (AXIS 1.4). The encoding style used in this framework is RPC/Literal, which is a style supported by the WS-Interoperability profile (WS-I) for building interoperable services. MUWS and the WS-Management framework though have moved in developing management applications using a Document/Literal style. This is accounted to two main reasons: (a) fewer toolkits support the RPC/literal style [34]; and (b) the RPC/Literal style is difficult to validate. Nevertheless, the Document/Literal style has two shortcomings: (a) it drops the operation name in the SOAP body making it difficult or impossible to dispatch a message to the implementation of the operation; and (b) it is WS-Interoperability (WS-I) compliant but with restrictions [34]. For the above reasons all literal styles are useful and currently supported by WS-I. Additionally, frameworks such as MUWS, as shown in [23], can be less lightweight than SNMP. The authors of [35] suggest the use of custom lightweight solutions within a network domain for performance and standards like WS-Notification at the edges for interoperability. This can also be applied in polling based monitoring using for example MUWS at the edges, and a custom framework in the core that minimizes processing, parsing, and serialization overheads.

2.2 Distributed Monitoring with SNMP

To investigate similar facilities for bulk and selective retrieval and information processing with SNMP requires examination of how distributed monitoring principles can be applied. Back in 1999 the IETF Distributed Management Working Group (DISMAN) was chartered to define a set of managed objects for specific distributed network management applications. Part of this group's work was the Definitions of Managed Objects for the Delegation of Management Scripts (Script MIB) [36] and the Distributed Management Expression MIB (Expression MIB) [37].

The Script MIB was conceived by the DISMAN working group and employs the advantages of distributed management over the centralized concept to tackle the increasing demands of network management. The Script MIB promised delegation of tasks, CPU and network load to mid-level managers and agents, thus increasing the robustness and reliability of a network. Despite the benefits, a number of important shortcomings limited this MIB's deployment. One problem was the "management of management" [38] increasing maintenance cost incurred by the need to distribute, run, update and control a script, and to gather and correlate the intermediate and final execution results—significant performance issues are identified in [38, 39]. Additionally, development costs are high since many aspects were left undefined such as: (a) how to get the Script MIB tables populated; (b) how to get scripts re-fetched and restarted after updates; and (c) procedural details of event forwarding. Most importantly there are security issues. Running software or

scripts is inherently dangerous (difficult to check what the nature of a script is) and makes it clear why approaches such as mobile agents or the script MIB never found significant deployment [1]. Furthermore, SNMPv3 that properly addressed security problems came too late. Eventually the IETF abandoned any further development of SNMP and consequently of its extensions [1].

The Expression MIB [37] was introduced as a way to create new, customized MIB objects for monitoring. Customized objects are required, for example, in event reporting so that monitoring is not limited to objects in predefined MIBs. The Expression MIB works by defining the objects to be monitored and also an expression evaluating these objects to create new ones. In theory, the expression MIB can also be used to provide bulk and selective retrieval capabilities for polling based monitoring, but the requirement is to use the Expression MIB in an architecture similar to the one in [40]. Here, a parser analyzes an expression to tokens of pre-defined grammar which invoke action routines assigned to retrieve management objects and evaluate the result. Although feasible, there are a number of problems associated with this approach. First and foremost, integrating the Expression MIB in existing agents that do not support it is not possible unless the agent uses an extensibility feature like AgentX [41]. Even so, "sub-agent access from the master agent for MIB variables" [41] is a non goal for AgentX. In addition, the expression MIB allows access only to local data for expressions since remote addresses are not supported. This eliminates the use of remote objects in the evaluated expressions, and thus limits the applicability of distributed monitoring principles with the Expression MIB only to local agent resources. The inherent distributed nature of WS allows us to overcome such problems since HTTP calls are usually accepted over firewalls [42]. Moreover, the standard itself [37] states that using the Expression MIB is often not a good trade-off for objects that are simply to be recorded or displayed, which is often the objective of monitoring and event reporting. Furthermore, as stated in the DISMAN charter [43] "implementing the Expression MIB is non-trivial work and takes lots of months to complete". Last but not least, the use of the Expression MIB to support bulk and selective retrieval is not practical, even with wildcarding, as the objects to be accessed and evaluated by an expression have to be re-set for different monitoring tasks. Due to the above deficiencies the Expression MIB has never seen any significant uptake in the real world.

2.3 Discussion

Despite the promise of the Script and Expression MIBs for distributed management, security problems, the lack of industry support, and to some extent their impractical nature, led the Internet management community to practically abandon their use. The decision in 2002 [44] to stop any further development of SNMP and the focus of the management community to XML-based technologies prevented further evolution of these technologies. Though the Expression MIB could be used to perform a set of more complex monitoring tasks such as a the query tool introduced later to minimize the processing overhead of WS monitoring operations, its use as shown later is impractical and introduces greater overhead. Additionally the Script

MIB could be used to perform a set of monitoring tasks by introducing a script for each task. Apart from the security concerns this raises compared to having a standard query tool for monitoring, it will be shown later that the need to update and control a script does not offer the flexibility of a query tool. As such, in the QoS scenarios of the next section, performance evaluation will be based on the classic centralized use of SNMP using bulk but not filtering mechanisms with no distributed management extensions.

The distributed nature of WS and the warm support received by the industry has led to the development of many standards supporting distributed management, security, event reporting and notifications, as well as bulk and selective retrieval for monitoring and event reporting. As in the case of SNMP, there are also problems associated with the use of certain XML-based tools and standards for WS management. As previously mentioned, XPath has scalability issues [10, 32, 33] which make its use prohibitive in standards such as MUWS, WS-Management, and WS-Notification for WS monitoring and event reporting [23, 27]. Therefore, these technologies are not used in our monitoring scenarios. We have instead developed a new framework for distributed monitoring that supports a number of custom based operations for monitoring, and uses the custom query tool and its parser in [31] for improved scalability [10]. Although our query tool is not currently supported by standard frameworks, it presents good integration potential with MUWS. The latter supports the definition of relationships between state data and operations for their retrieval, which are used by our query tool for effective monitoring. Furthermore, MUWS is designed to be independent of the implementation specifics representing the resources of managed devices. As such, it permits object based representations of the underlying resources which is required by our query tool. Future plans for MUWS also include the support of resource specific query tools as defined in [45].

3 QoS Monitoring Requirements and Scenarios

3.1 Introduction

Quality of Service provisioning has been the subject of extensive research over the years. Examples of such work are the frameworks proposed by the TEQUILA [46, 47], CADENUS [48], and ENTHRONE [49] projects. QoS is currently provided on the basis of a set of terms that clients and service providers have to abide by, called Service Level Agreements (SLAs). The technical part of a SLA is known as a Service Level Specification (SLS) [50], which defines the network characteristics corresponding to specific services. IP Differentiated Services [51] (DiffServ) is currently considered as the framework for providing QoS-based services. A common approach to support the DiffServ architecture is over Multi-Protocol Label Switching (MPLS) traffic-engineered networks. Traffic Engineering (TE) requires a scalable monitoring system capable of acquiring long and short term data. In MPLS networks, for scalability reasons, measurements are performed only at the MPLS Label Switched Path (LSP) level, or at the QoS traffic class level (Per Hop Behavior—PHB). Two types of measurements can be identified: active and passive

ones. Active measurements are performed by injecting synthetic traffic to the network in order to monitor delay, jitter and packet loss, whereas passive measurements can be conducted using Management Information Bases from SNMP and involve measuring throughput, load and packet discards at the PHB, SLS and LSP levels [46, 47, 49]. The type and the specific points where passive measurements can be performed are the following:

- LSP load at the ingress router (LSP L-I).
- LSP throughput at the egress router (LSP T-E).
- PHB throughput at every router (PHB T).
- PHB packet discards at every router (PHB D).
- Offered load at ingress per SLS or flow (SLS L).
- Offered throughput at egress per SLS or flow (SLS T).

As this paper focuses on testing and enhancing the scalability and performance of WS and SNMP for collecting management data, active monitoring will not be considered. Active measurements can be performed in the same manner for both SNMP and WS, and are not of any benefit in the context of this work.

3.2 QoS Monitoring Scenarios

Three scenarios are considered based on which the performance and scalability of SNMP and WS are examined. The first scenario compares the performance of WS and SNMP for retrieving data related to the passive measurements described in the previous section. A bulk of data is retrieved simultaneously and monitoring with WS is performed in a similar manner to SNMP's GetBulk operation. The performance of consecutive SNMP GetNext operations to acquire the same data is also examined. The second scenario investigates how bulk and selective retrieval facilities used for data processing at the agents can relieve the manager from this burden. Again, a bulk of data is retrieved, but this time data is processed, using the selective retrieval facilities of the WS query tool and parser developed, before delivered to the manager. Such facilities are not supported by SNMP for the reasons explained in Sect. "2.1". This scenario shows how to delegate and distribute the data processing load when monitoring a large number of agents. This approach will be demonstrated as a more viable option for management using WS. The third scenario assumes a failing MPLS interface where the manager needs to determine the affected LSPs and service contracts (SLSs). In this scenario our query tool is used to perform data filtering to retrieve only the relevant information from the MIB hosted at an agent.

4 WS and SNMP Monitoring

4.1 SNMP Monitoring

To perform the measurements required for the three scenarios, some of the SNMP special Management Information Bases (MIBs) must be used. The first five passive

measurements of Sect. "3.1" can be performed using two of the SNMP's MPLS MIBs; the MPLS Label Switching Router (LSR) MIB (RFC 3813) [52], and the MPLS Forwarding Equivalence Class to Next Hop Label Forwarding Entry (FEC) MIB (RFC 3814) [53]. The LSR MIB can be used to perform PHB and LSP measurements, whereas the FEC MIB can be used to perform the SLS load measurements. The sixth measurement cannot be performed with the FEC MIB as this is only deployed at an ingress router. As such, this measurement is not analyzed in the scenarios. For experiments involving SNMP, we used a Net-SNMP agent and the AdventNet SNMP toolkit. The manager-agent paradigm is employed, with the agents hosting the appropriate MIBs (Interfaces LSR, FEC). For all scenarios, bulk retrieval is allowed but selective retrieval mechanisms are not possible as plain SNMP does not support them. Distributed management extensions are also not applicable for the reasons mentioned previously.

The process of retrieving SLS, PHB and SLS related data for the scenarios is explained in the next sub-section.

4.1.1 Retrieving Passive Measurement Data with SNMP

For the first five passive measurements using SNMP, a number of MIBs, tables and entries must be accessed. For LSP measurements, the manager must query the agent for the following data:

- LSP-IDs in order to determine how LSPs are organized in a specific table. The row identifiers (row IDs) of the relevant table are also returned so as to perform the next step.
- LSP load at the ingress (LSP L-I) or throughput at the egress (LSP T-E) from tables holding statistical information using the row-IDs returned in the previous step.

For PHB measurements the process is the following:

- Retrieve the PHB IDs in order to determine the PHB each LSP belongs to.
- Query for the throughput (PHB T) or packet discards (PHB D) for each PHB, based on the row IDs retrieved from the previous step.

To determine SLS load, the manager must do the following:

- Query which LSPs each SLS is associated with (SLS IDs, Differentiated Service Code Point field DSCP and LSP IDs).
- Based on the row IDs obtained, the manager can access the statistics table for each LSP in order to compute the load for each traffic contract (SLS).

Determining the IDs of LSPs, SLSs and PHBs as a first step is an essential process. For all measurements it is assumed that the manager is aware of the network topology through a topology repository. This implies that the manager, apart from topology information, also has knowledge of the ID of every contract, the ID of each LSP at the ingress and egress routers, as well as the various traffic classes supported. Data queries though are based on the exact location of tabular information. As such, the manager needs to initially determine the order in which

LSP, PHB, and SLS data is organized in the tables, and then retrieve the information relevant to the passive measurement scenarios.

4.1.2 Granularity of SNMP Monitoring Operations on Passive Measurement Data

Another aspect that requires special consideration is the sampling granularity for each type of data. Since each class of service has different requirements, it would be expected that different sampling frequencies should be applied to the traffic of each class. A premium class of service, for example, requires more frequent measurements than a best-effort service. Using different sampling frequencies for SLSs would make the monitoring architecture more complex. For this reason, we use a selection of three different sampling frequencies based only on the type of data that needs to be retrieved. Determining, for example, the IDs for each LSP requires a relatively long sampling period (long granularity, in the order of days or weeks) since the same LSP configuration is typically retained over a relatively long period of time (a provisioning period). The value of tabular objects referring to PHB and SLS specific data, on the other hand, may change more frequently. This occurs due to failures on MPLS-capable interfaces, load balancing to meet the requirements of each traffic contract, etc. During the load balancing process the traffic on heavily utilized LSPs is assigned to less utilized ones. Solving such problems might be possible without reconfiguring the entire network, by routing the traffic mapped on the failing interface or on a heavily utilized link to another. As such, PHB and SLS related information change more frequently and the polling period is of medium granularity, in the order of 5–10 min. Lastly, sampling periods for more dynamic data such as load and throughput are in the order of 5–20 s (short granularity).

4.2 WS Monitoring

The measurements performed with WS also make use of the LSR and FEC MIBs. These MIBs have been re-implemented from the beginning as native WS, following the tree structure described in their RFCs—SNMP agents are not re-used. All data is stored in single values, tables and linked lists (raw format). To represent the MIB hierarchy, each data structure is connected to another with special pointers allowing the navigation of the data. Each MIB is deployed as a WS and exposes three functions that allow access to different portions of the MIB data (single instance, multiple instance, and all data view).

The steps and the granularity that is required to retrieve passive measurement data for the three scenarios using WS are the same as the ones in Sects. “4.1.1” and “4.1.2”.

4.2.1 WS Monitoring Process Description

To support bulk and selective retrieval in the case study scenarios of Sect. “3.2”, we have developed the query tool in [31]. The tool uses four types of queries, each one with a special functionality. Each type of query is a parameter in the three methods exposing a MIB’s data as a WS. The types of queries supported are: (a) *service*

selection; (b) *single instance* data; (c) *multiple instance* data; and (d) *filtering* data queries. *Service selection* queries are practically WS endpoints with URI syntax [54] and allow the selection of local or remote WS representing a MIB from which data are retrieved. SS queries also allow the navigation of relationships that state data share for more efficient monitoring. As explained in [31] this allows a manager to acquire a more complete view on the state of a device exposed through a WS interface and is based on an idea similar to the scoping mechanism used in CMIP++ [55] to select managed objects. *Single* and *multiple instance* data queries allow the retrieval of single and multiple instance data (i.e. a table row is represented by multiple instance objects in a linked list) from a WS exposing management data, respectively. *Filtering data* queries can be applied to *multiple instance* queries to filter the collected data.

The process of retrieving management data from WS implementing the aforementioned MIBs is depicted in Fig. 1. In order to support the necessary monitoring operations we have defined a new framework that supports three methods, each of which takes a callback address for the manager and the tool queries as parameters. These methods are supported by all agents and all the WS exposing management data. Initially the manager, having knowledge of the ID of every contract, the IDs of each LSP at ingress and egress routers, as well as the various traffic classes from the topology repository, selects one of three methods exposed by a WS MIB depending on the data it requires to retrieve. The manager then sends a request for data to the agent containing service and data selection queries, as well as a callback address. The agent analyzes each service selection query to determine whether a local or remote WS exposing management data needs to be accessed. In case a remote WS MIB needs to be accessed, the current agent tries to route the service and data selection queries it received to the remote agent. This is achieved by extracting the remote agent's address from the service selection query and sending a request to the next hop agent through which this address can be reached. This continues from agent to agent until the remote agent is reached. For this scheme to work efficiently it is better not to dispatch queries for remote resources to agents far from the agent hosting the remote resources. Each request made to a next hop agent contains the data queries associated with the service selection query and also the callback address of the manager. At the remote agent, the manager's callback address is used to send back the required data. This is a load and task distribution process, where a manager delegates tasks to several other agents. If local data need to be accessed the queries are analyzed using the query tool instance of the relevant WS and the appropriate data are retrieved from the router an agent is associated with. Currently, routers and agents are hosted on Linux machines supporting MPLS for Linux. After all the required data is collected the agent sends a response to the manager which can be then stored in the monitoring repository.

The process of how each WS MIB is deployed at the agent and how the agent is handling the tool queries for a case where local data need to be accessed, is depicted in Fig. 2. The agent first deploys all the WS required (step 1—the LSR, FEC, and Interfaces MIBs have been implemented for our scenarios). Each WS creates its own query tool instance during steps 2–4. Once all services are deployed the agent

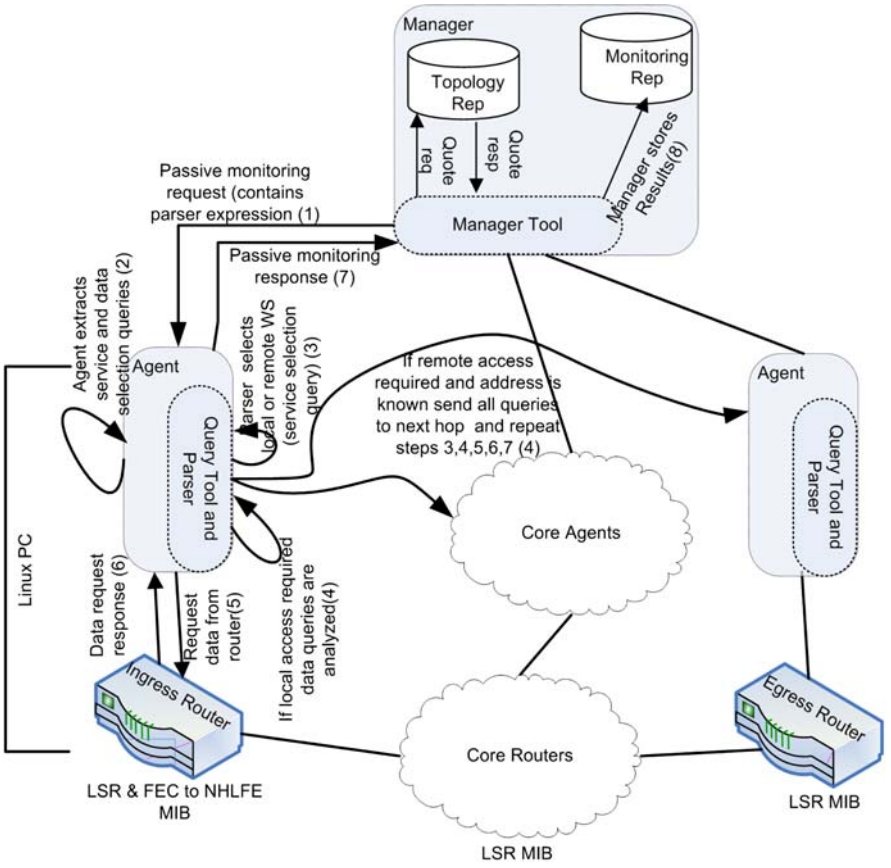

Fig. 1 Distributed monitoring framework functionality (remote and local resource monitoring)

also deploys itself as a service component (step 5). Although data independent, each tool instance is aware of the type of data exposed by the relevant WS and is able to resolve incoming queries for inexistent data faster (steps 2–4). As demonstrated in Fig. 2, the manager's service selection queries, as in Fig. 1, are resolved through the agent's query tool parser instance (steps 6–8). Each service selection query is analyzed to determine whether remote or local data need to be acquired in step 7. For local data the queries are provided as parameters in each method the manager invokes, and are evaluated by each WS MIB's tool instance (steps 9–11). When all data are collected from each WS (step 12) the agent concatenates it (step 13) and sends a response back to the manager (step 14).

Based on the above it is evident that the benefits of our query tool are threefold. First our query tool supports as part of the architecture described in Fig. 1 the distribution of the monitoring load. This is possible by designing WS-agents that support the functionality of XPath or our query tool or any other query tool. Distribution of the monitoring load can also happen through means of a signaling protocol as suggested in projects like Enthrone. The latter approach may be a more

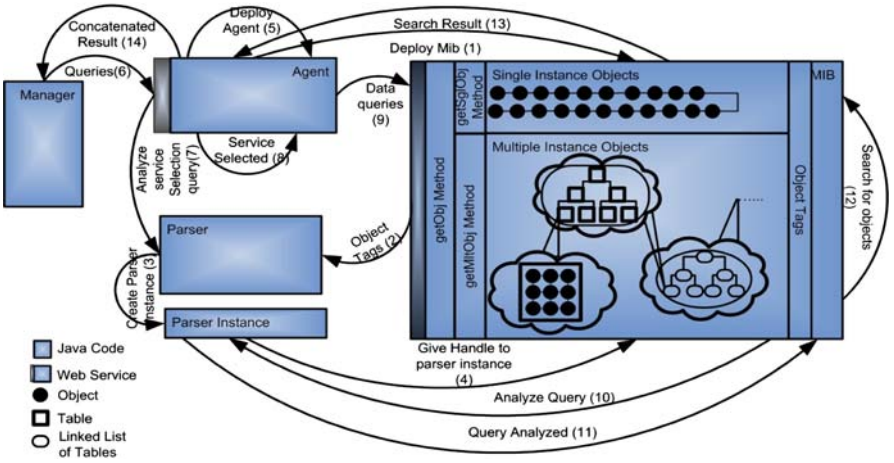

Fig. 2 Interactions between components of the WS polling based monitoring scheme to retrieve management information (only for local data)

obtrusive solution. At the same time our query tool supports the navigation of relationships between state data relationships for effective monitoring. Finally our query tool supports information processing for bulk and selective retrieval.

4.2.2 WS Monitoring Example

An example of using our tool for bulk and selective retrieval will make Sect. "4.2.1" more comprehensive. The example involves retrieving the incoming TCP connections of type HTTP or FTP from an agent hosting the RFC 1213 MIB groups as WSs. The queries would be as follows:

Qry_1 = {http://AgentAddress:AgentPort/TcpGroup} (service selection query).
Qry_2 = {tcpConnEntry[]} (multiple instance query).
Qry_3 = {tcpConnRemPort = 22 AND tcpConnRemPort = 80} (filtering query).

Selecting one of the three standard monitoring operations that the agent exposes, the manager sends the above queries as the operation's parameters. A callback address parameter is also included. Using query 1 the agent determines if this involves a local or remote resource. If the former then the equivalent method of the TcpGroup WS is called with the above multiple instance and filtering queries as parameters. Within the service methods, the parser of the query tool of each service is used to analyze the data queries, and the data hierarchy (raw data) is searched to determine the relevant data to be returned to the manager in XML format. Finally, the agent concatenates all data received from all the selected services and returns the result in an XML document (part of the SOAP body) to the manager. If remote data need to be accessed this is done by the remote agent.

5 Measurements

5.1 Evaluation Setup

For the evaluation aspects of our scenario, a big number of LSPs need to be setup for some of the measurements. As this is difficult to be achieved in a small testbed, we resorted to other means for evaluating the MPLS MIBs performance overhead for SNMP. For traffic overhead, the average size of each message is calculated by looking into the message and analyzing the size of its subparts as presented in the next section. For computing latency a similar number and type of objects as in the MPLS MIBs are instantiated and Advent-Net SNMP v3.3 is used to access a Net-SNMP v5.0.2 agent. For WS the Apache Axis 1.4 SOAP toolkit was used to deploy the three MIBs as WS, with the same information as in SNMP. All MIBs were deployed using an RPC/literal encoding style so that the verboseness of XML tags is reduced and traffic overhead as well as coding latency is minimized. For the parser of the query tool that all WS MIBs use to support selective or bulk information retrieval, Java 1.4.2.10 and its regex engine was used. The traffic overhead was calculated as analyzed in the next section for SNMP using an average for each type of measurement. For WS the Linux *tcpdump* utility was used to measure traffic, and latency measurements for WS and SNMP were performed using Java's currentTimeMillis() function by averaging 20 measurements for each sampled result. The manager and agent were deployed on a 1,000 MHz/256 MB RAM and 466 MHz/192 MB RAM machines, respectively, thus simulating a lower end system for the agent. Both machines run Red-hat Linux.

5.2 SNMP Traffic Consumption Calculation

SNMP messages use ASN.1 syntax [56]. When retrieving data, a SNMP manager can use three types of messages or Protocol Data Units (PDUs): *Get*, *GetNext* and *GetBulk*. All three types consist of several fields including the version, the community, the PDU type, the request ID field, and the variable-bindings field. Get or GetNext also include an error status field and an error-index field, whereas GetBulk includes a Non-repeaters and a Max-repetitions field. In the experiments performed:

- The version field is v1/v2.
- The community field is "public".
- The PDU type is GetNext or GetBulk.
- Object requests will not exceed 2,000 and so the request-ID field will not exceed this value.
- The error-status field can take only five values.
- Since the error-index shows the variable in the variable binding list that caused an error and the number of variables we retrieve in a packet will not exceed 127, this value cannot exceed this number.
- For the experiments the Non-repeaters = 0 and the Max-repetitions field will not exceed 2,000.

Measurement Type	L1	L2
LSP IDs	16-19 (Max 16000 LSPs)	6 (CR-LDP)
LSP Load Ingress (L-I) Th/put (T-E) Egress	14-15 (Max 16000 LSPs)	1-4 or 1-8
PHB Th/put (T) and packet Discards (D) and SLS Load (L)	14-15 for each (Max 16000 LSPs)	1-4 or 1-8 (L) and 1-4 (D)
PHB IDs	14-15 (Max 16000 LSPs)	14
SLS IDs	14-16 (Max 16000 LSPs)	1-3 (Max 16000 LSPs)
SLS LSP IDs	14-16 (Max 16000 LSPs)	16-20 (Max 16000 LSPs)
Interface IDs	14-16 (Max 16000 Ifs)	1-4

Fig. 3 L_1 and L_2 length analysis for all scenarios

By calculating the size of each field using the Basic Encoding Rules in [57] and taking into account how the measurement requirements affect the size of each field as well as a previous study on the traffic of SNMP [2], the size of an SNMP request or response message can be computed as follows:

$$L\text{get, getNext, getBulk} \approx 27 + n \times (6 + L_1 + L_2) \tag{1}$$

In Eq. 1, L_1 is the size of the Object Identifier (OID) of a variable, L_2 is the size of the variable value itself, n is the number of OIDs to retrieve, and the numbers 27 and 6 relate to the encoded size of the message subparts. The request PDU in all scenarios contains one or two OIDs. In the case of GetBulk the number of OIDs is different than the number of objects returned. Therefore, the size of SNMP operations for information retrieval, if n_1 objects are retrieved and a single OID exists in the request packet, is given in Eqs. 2 and 3.

$$L\text{get, getNext} \approx n_1 \times (54 + 12 + 2L_1 + L_2) \tag{2}$$

$$L\text{getBulk} \approx 54 + 1 \times (6 + L_1) + n_1(6 + L_1 + L_2) \tag{3}$$

Figure 3 presents the size of L_1 and L_2 which have been calculated for each measurement type. These values can be substituted in Eqs. 2, 3 to determine the maximum, minimum and average traffic of SNMP information retrieval operations for all scenario measurements. In the results in the next section we list average traffic.

5.3 Scenario Measurements

5.3.1 Scenario One: No Filtering or Processing

In this scenario data are retrieved with SNMP either with consecutive GetNext operations or with a single GetBulk operation. For WS, data are retrieved with one request that is appropriately interpreted by the query tool similar to GetBulk. For this scenario traffic overhead for LSP, PHB and SLS measurements are presented in Figs. 5, 6, 7, 8, 9 and 10, whereas latency results are provided in Figure 4. For measurements of scenario 1 and 2 the following things need to be clarified:

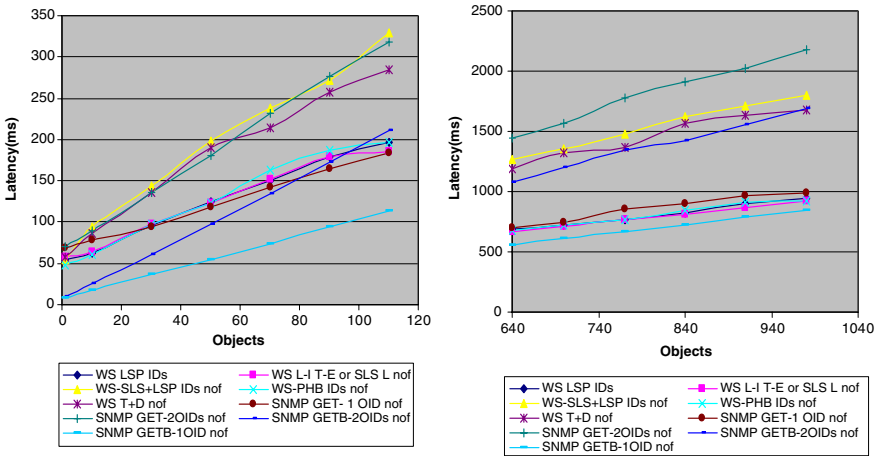

Fig. 4 Latency for all measurements for scenario 1

- For both WS and SNMP, PHB packet discards and throughput (T + D) are retrieved by asking for both variables in the request packet.
- For SNMP, SLS IDs and SLS LSP IDs (SLS + SLS LSP IDs) are retrieved by asking for both variables in the request packet. For WS only SLS IDs are retrieved since processing occurs at the agent side.
- We differentiate between the WS PHB and SLS measurements between the scenario 1 and 2 by incorporating a filtering flag (denoted *noffl*, respectively).
- LSP measurements are performed in the same way for both scenarios as no processing is required due to the way data are organized in tables.
- For SNMP, latency is about the same for all types of objects retrieved for scenario one and two and depends roughly on the number of objects retrieved and their location in the tree. We can find the average latency for an SNMP type of request packet (GetNext, GetBulk) by averaging the results we had for each measurement type in Fig. 3. In order to make the latency Figs. (4, 11) more readable, we present only average latency plots for all measurement types in Fig. 3 depending on the type of request packet (GetNext/GetBulk) and the number of OIDs in the request packet (1 or 2 OIDs).

From the graphical representations (Fig. 5-LSP-IDs Fig. 6-LSP load or throughput) it is evident that WS start producing less traffic than SNMP's consecutive GetNext operations if more than 30 objects are retrieved. This occurs because with the RPC/literal encoding XML tags are less verbose and thus the initial overhead of HTTP/SOAP is overcome. Still, the size of data and tags cannot be less than the size of the OIDs plus the variables of a GetBulk operation. Considering latency measurements that involve a single OID retrieval, the performance of WS is better than SNMP's consecutive GetNext operations (Fig. 4). Although SNMP encoding is faster, most SNMP agents do not support caching [2] which inhibits performance. When two variables are requested, SNMP produces even more latency as more data must be retrieved. This emphasizes the advantage of caching capabilities when

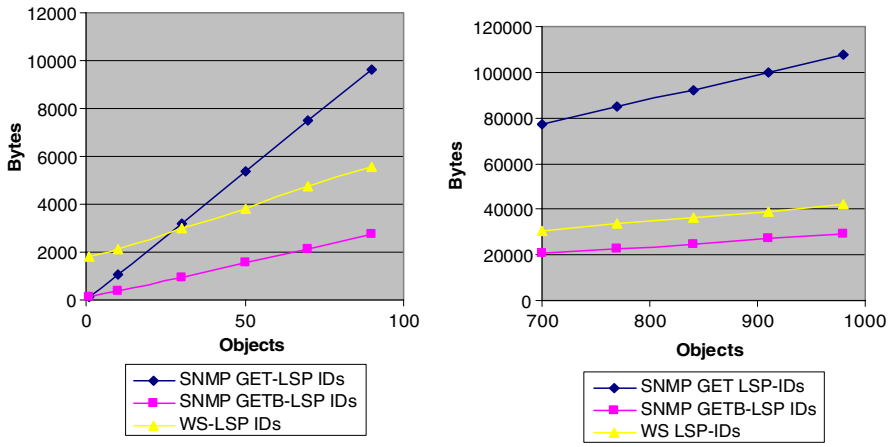

Fig. 5 Traffic load for retrieving LSP IDs, scenario 1&2, small and big nets

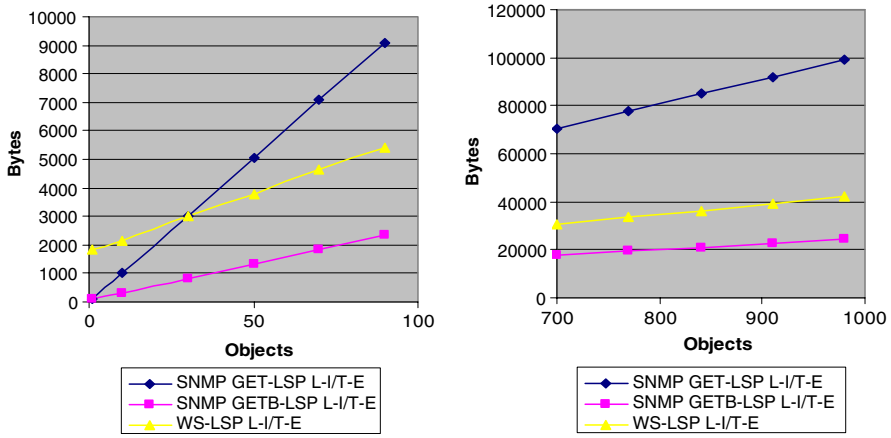

Fig. 6 Traffic load for retrieving LSP load/throughput, scenario 1&2, small and big nets

retrieving large amounts of data. Regarding GetBulk, the latency is slightly better than that of WS. In the case of requesting two variables though, the gradient of the GetBulk result is higher than that of WS. Thus, when the number of objects retrieved exceeds 1,000, the absence of caching makes even GetBulk's latency worse than WS.

5.3.2 Scenario Two: Data Processing

In this scenario the same information as in the previous one is retrieved, but this time PHB and SLS related data is processed at the agent using the query tool. This way, the manager receives state data on a per PHB or SLS basis. As such, load, throughput or packet discards can be computed by the manager without any

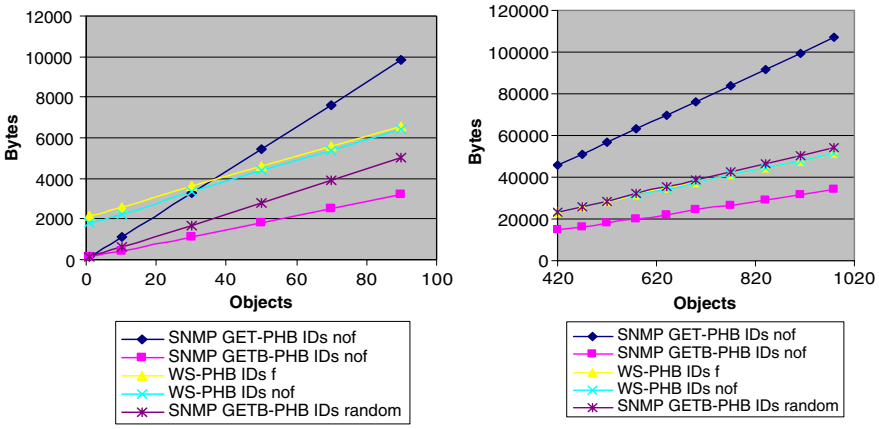

Fig. 7 Traffic load for retrieving PHB-IDs, scenario 1&2, small and big nets

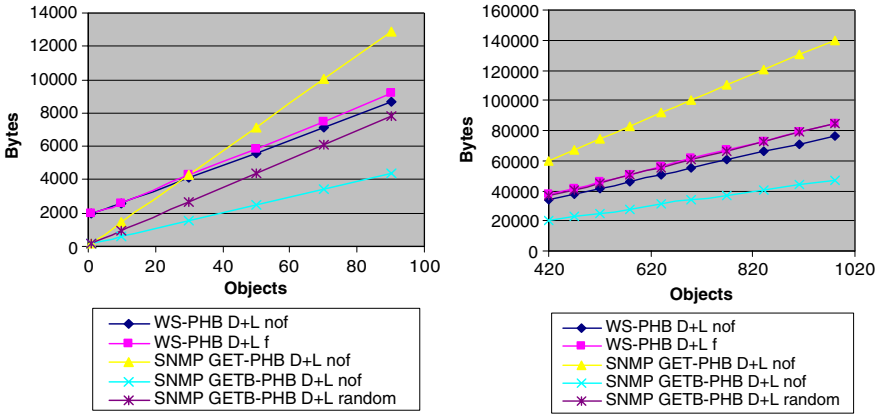

Fig. 8 Traffic load for retrieving PHB discards and load, scenario 1&2, small and big nets

processing at the manager side. This approach is better because although the manager is usually hosted in a high-end system, it will have to access many agents to retrieve PHB or SLS related data. Thus the manager runs the risk of being easily overwhelmed by the amount of processing required. Our custom query tool allows us to distribute the load to several agents. This is feasible today especially since the myth of the dumb agent is no longer valid [42].

Before analyzing the measurements of this scenario some aspects need to be highlighted. As such it must be noted that for PHB measurements the LSPs in each router are assigned to six different traffic classes. Also for SLS measurements we decided to investigate two cases. The first case concerns individual SLSs, each bound to a different LSP (1LSP/C, C = contract), for which we investigated traffic and latency for 1–300 Contracts (C). In the second case we investigated a Virtual

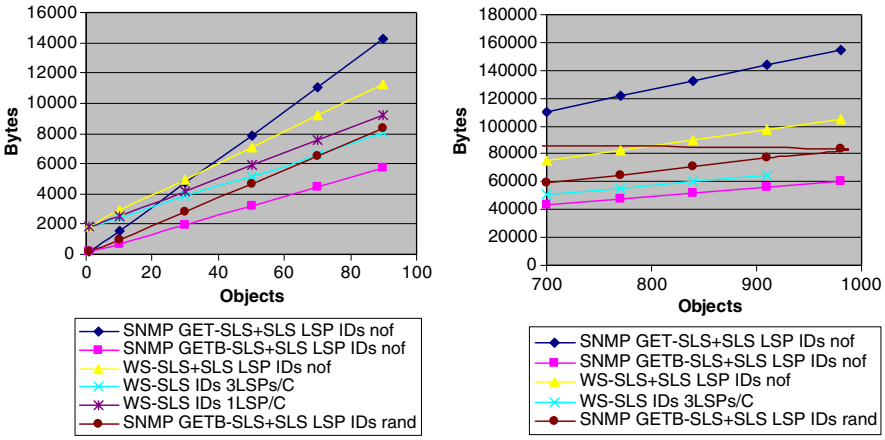

Fig. 9 Traffic load for retrieving SLS-IDs and SLS LSP-IDs, scenario 1&2, small and big nets

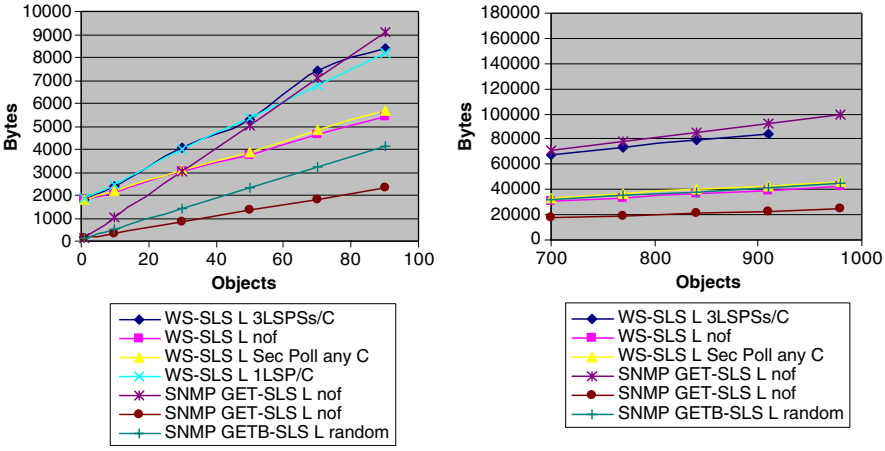

Fig. 10 Traffic load for retrieving SLS load, scenario 1&2, small and big nets

Private Network (VPN) scenario for the same number of contracts where each SLS in the ingress router is bound to three different LSPs (3LSPs/C).

In all the measurements of these scenario WS perform better than successive GetNext operations in terms of traffic overhead, but worse than GetBulk. It is worth analyzing though certain aspects, in order to show that if SNMP was used to retrieve data on a per PHB or SLS basis its performance could become worse.

To acquire data on a per PHB basis the following data queries must be sent to the agent as the parameters of the framework's methods:

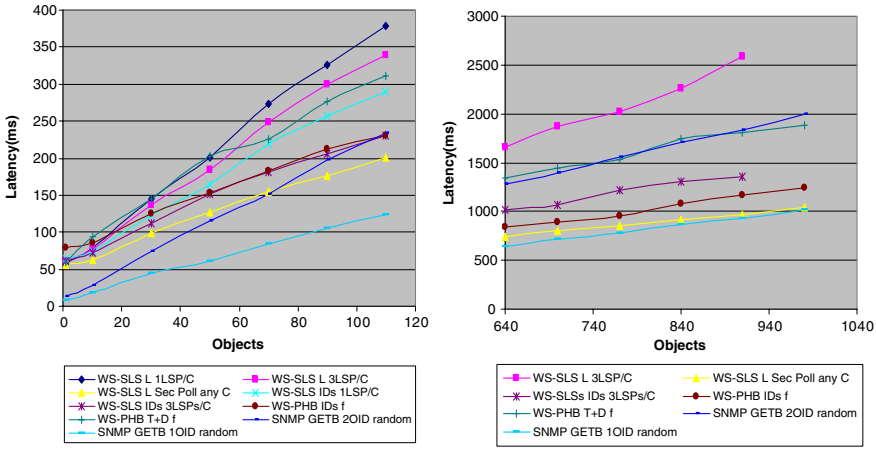

Fig. 11 Latency for measurements in scenario 2

{mplsInSegmentTrafficParamPtr[],mplsInSegment TrafficParamPtr[],…}	(multiple instance query)
{value = OID_1, value = OID_2,…}	(filtering query)
{mplsOutSegmentPerfHcOctets[id_1, id_2,…], mplsOutSegmentPerfDiScards[id_1, id_2,…]}	(multiple instance query)

Figure 7 (PHB IDs f/nof) suggests that the traffic overhead for discovering PHB IDs is roughly the same as in the previous scenario. Latency increases by a maximum of 300 ms when 980 LSP IDs must be retrieved (Fig. 4-PHB IDs nof and Fig. 11-PHB IDs f) as the relevant tables need to be searched six times. This happens because the query tool does not allow the use of a single filtering query for all traffic classes and still receive data on a per PHB basis. Consequently, latency can improve if the query tool is optimized to support more compact queries. For throughput/load and packet discards, the traffic overhead also increases by a maximum of 8,000 octets for 980 LSPs (Fig. 8 PHB D + L f and nof). The increase in traffic overhead occurs because the manager needs to access throughput on a PHB basis. The latter implies that the row identifiers in the statistics table for each PHB have to be included in the request packet. Despite the overhead, throughput/load (PHB T/L) and packet discards (PHB D) for each PHB are fairly easy to compute. Accessing throughput/load (PHB T/L) and discards (PHB D) on a per PHB basis also increases latency by a maximum of 200 ms for 980 LSPs (Fig. 4- T + D nof & Fig. 11-PHB T + D f). Still, for all PHB measurements, the granularity requirements for polling data from the agent are satisfied (5–10 min for PHB IDs and 5–20 s for T + D).

This might not be the case if SNMP was used without distributed management extensions. This is especially true for throughput and discards, as the manager would have to process data from many agents to discover the PHB each LSP row entry in the statistics table belongs to in order to query for throughput and packet discards and also to calculate these on a per PHB basis. This would introduce more latency even if the manager was in a high end system (for the manager used in the experiments, this could exceed half a minute for a single monitoring task). Also, since data would have to be retrieved on a per PHB basis to calculate throughput or discards, all OIDs of the required objects would have to be defined in the request packets. This forces GetBulk's latency to become even worse (not considering processing latency) than WS due to lack of caching as objects need to be accessed randomly (Fig. 11 GETB 1or 2 OID random). Getbulk's traffic overhead is also increasing since more OIDs must be defined in the request packet. More specifically, traffic overhead is slightly bigger than WS for the same measurements when more than 420 objects are requested (Fig. 7 GETB-PHB IDs random, Fig. 8-GETB-PHB D + L for load and discards).

Even with distributed management extensions such as the Expression MIB that could allow SNMP to calculate throughput or packet discards on a per PHB basis, usage of the Expression MIB would not be a good trade-off. This is because the manager would again need to determine the PHBs the row entries in the statistics table of each LSP segment belong to. As mentioned previously, processing data from many agents could be prohibitive when the manager has to process data from a number of agents. Even after determining the PHB each LSP belongs to, the manager has to set a number of parameters in the tables of the Expression MIB by following a two step procedure. In the first step the manager has to set in the expObjectTable of the expression MIB a series of four objects (4 OIDs expObjectID, expObjectWildcard, expObjectSampleType, expObjectRowStatus) for each LSP throughput or packet discards parameter. Although the expression MIB permits wildcarding to set all the parameters in the expObjectTable in one go, this is not possible in this case because the order according to which each LSP is assigned to a PHB in the mplsXCTable of the LSR MIB is random. As such, the LSP throughput or packet discards parameters have to be set in the expObjectTable explicitly because they need to be used explicitly in an expression to calculate PHB discards or throughput. In the second step, the manager will have to set in the expValueTable of the expression MIB a separate expression to calculate the PHB throughput or packet discards for each traffic class by referring to the expObjectID parameters of the expObjectTable.

After completing the two steps described above, the manager can retrieve all the values in the expValueTable to acquire PHB throughput or packet discards using a getNext operation. For the first step alone, usage of the expression MIB would roughly quadruple the monitoring overhead (four OIDs and four values for each LSP throughput or packet discard parameter have to be set) compared to SNMP without distributed extensions (one OID and one value for each LSP throughput or packet discard parameter). This does not take into account the latency overhead for setting such a big number of objects in the expression MIB. Additionally, for each object in the expObjectTable, the expression MIB would have to query the agent

hosting the data referenced in the expObjectTable with a series of consecutive getNext operations (one OID and one value for each LSP throughput or packet discard parameter). This increases latency and traffic overhead even further. The fact that PHB related data can change every few minutes, means that it might be necessary to change the parameters used in each expression quite frequently. It is evident from the above that using the Expression MIB for this scenario is not plausible.

Using the Script MIB to calculate throughput or packet discards on a per PHB basis, apart from security issues, it may also pose significant performance issues. In devices running at high speeds, a simple script would need 15 MBs of memory just to run, update and control the script [38, 39], ignoring the data that would have to be processed by it. Our query tool on the other hand (including the java libraries of apache AXIS and tomcat, and the query tool itself and processing overhead), requires not more than 6.5 MBs and 16 MBs for small and big networks, respectively.

Service Level Specification measurements are a bit more complicated. In this scenario the manager queries for the LSP IDs based on the SLS ID of each contract (i.e. the DiffServ Code Point—DSCP). The agent returns the LSP IDs per SLS but not the SLS IDs, since the latter are not required for the next query or for processing at the manager side. The data queries for this scenario sent as parameters of the framework's methods from the manager to the agent are the following:

{mplsFTNActionPointed[]}	(multiple instance query)
{mplsFTNDscp = $value_1$, mplsFTNDscp = $value_2$,…}	(filtering query)
{mplsInSegmentPerfHcOctets[], mplsInSegmentPerfHcOctets[],…}	(multiple instance query)
{mplsInSegmentIndex = $value_1$ OR mplsInSegmentIndex = $value_2$…, mplsInSegmentIndex = $value_3$ OR mplsInSegmentIndex = $value_4$…,…}	(filtering query)

Although the expressions for retrieving the LSP IDs for each SLS suggest that the request packet contains as many queries as the number of contracts, the overall WS traffic is much less in comparison to the previous scenario for the same measurement because less data is returned (very close to GetBulk traffic with no filtering involved—see SLS IDs 3LSPs/C or 1LSP/C on Fig. 9). Latency is also much less (maximum 200 ms less than GetBulk—see Fig. 4 SLS+LSP IDs nof, Fig. 11 SLS IDs 3LSPs/C or 1LSP/C). Despite the amount of processing required, even when a 900 entry traffic table has to be searched 300 times for the VPN scenario, WS performance is substantially better as less data needs to be encoded in the SOAP body. For SLS WS load measurements on the other hand, traffic and latency (Fig. 10 SLS L 1LSP/C or 3LSP/C and SLS-L nof & Fig. 4 & 11 SLS L *nof* and SLS L 1LSP/C or 3LSP/C) have increased (maximum three times more latency

and maximum 20 K more traffic). This happens only when load (L) is polled for the first time every granularity period (mins) since there is a need to determine the entries in the statistics table of the incoming segment referring to each LSP. After that, the SLS load is measured using the row identifiers obtained in the first measurement. Traffic overhead and latency for load after the first polling period, is similar to the WS measurement for SLS load in the previous scenario (Fig. 10 SLS L Sec Poll any C and SLS L nof, Fig. 4 SLS L nof and Fig. 11 SLS L Sec Poll any C). Identifying the entries in the incoming segment statistics table referring to each LSP needs to happen with a medium term granularity since the SLSs referring to each LSP load might change every few minutes. Therefore, the granularity requirements for polling data from the agent are satisfied, 5–10 min for SLS IDs, 10–20 s for SLS (L). For the same reasons as for the PHBs, this might not be the case if SNMP was used, especially for short-term measurements (Fig. 10-GETB-SLS L random, Fig. 11, GETB-1OID random, GETB-2OIDs random and Fig. 9, GETB-SLS + SLS LSP IDs rand). Nevertheless, performance of the WS SLS measurements could be improved if the query tool was enhanced to support more compact queries.

5.3.3 Scenario Three: Filtering

In this scenario we demonstrate how a WS-based monitoring system can benefit from data filtering. Here, we assume that an ingress router MPLS interface fails in which case a notification is dispatched to the manager. Upon receiving this event, the manager needs to determine the affected LSPs and SLSs so as to take appropriate actions. For the measurements, the ingress router is configured to have 910 and 50 LSPs to simulate big and small networks, respectively, each of which is assigned to a different customer. The reason behind assigning a different customer to each LSP is to keep things simple with respect to validity checks when data queries are performed. A further assumption in this experiment is the number of LSPs and SLSs affected by the failing interface, which we assume to be six. Although it is not easy to determine a plausible number of LSPs assigned to a single interface, six would be a reasonable number for small networks. It may not be realistic to use the same number in the case of large networks, but the aim is to keep the volume of data to be retrieved relatively low. This way we can show that WS can benefit from sophisticated retrieval mechanisms and exhibit superior performance against SNMP even though a small volume of data needs to be retrieved (not shown in [2, 3] or other WS performance related work). Additionally, by keeping the same number of affected SLSs and LSPs for both small and large networks we can maintain the traffic overhead and latency comparison between them on the same terms.

Three steps are required to perform this scenario's measurements: (a) determine the identifiers of the LSP incoming/outgoing segments associated with this interface; (b) use these identifiers to determine the affected LSPs IDs; and (c) determine the affected SLSs using these LSP IDs. The data queries without the

service queries sent as parameters of our framework's operations sent to the agent are the following:

{mplsInSegmentInterface[], mplsOutSegmentInterface[]}	(multiple instance query)
{value = ifIndex, value = ifIndex}	(filtering query)
{mplsXCLspld[]}	(multiple instance query)
{mplsInSegmentXCIndex = mplsInSegmentIndex$_1$ OR mplsOutSegmentXCIndex = mplsOutSegmentIndex$_1$ OR…}	(fiterling query)
{mplsFTNDscp[]}	(multiple instance query)
{mplsFTNActionPointer = mplsXCLspld.mplsXCIndex.mplsXCInSegmentIndex. mplsXCOutSegmentIndex OR…}	(filtering query)

The measurements for this scenario are presented in Figs. 12, 13, 14 and 15. The total amount of traffic produced as a result of the three queries for both big and small networks using WS is 6,653 octets (the same amount of information is retrieved since filtering is supported and the number of SLSs and LSPs affected by the failing interface is the same). The total latency for WS in small and big networks is 134 and 425 ms, respectively. If SNMP was used for the measurements, all the interface entries, the LSP IDs and the SLS IDs along with their LSP IDs would have

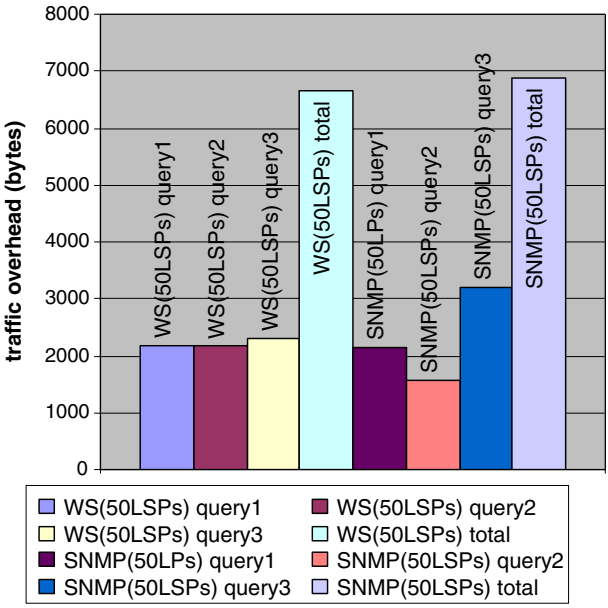

Fig. 12 Traffic overhead measurements for WS & SNMP for a small network

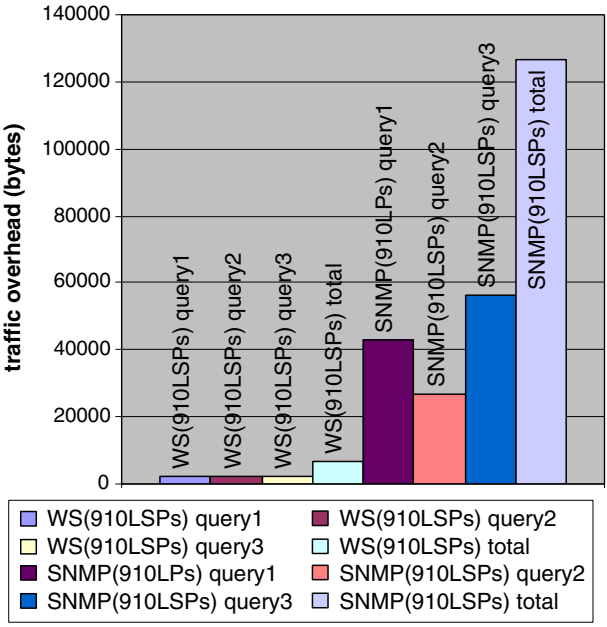

Fig. 13 Traffic overhead measurements for WS & SNMP for a big network

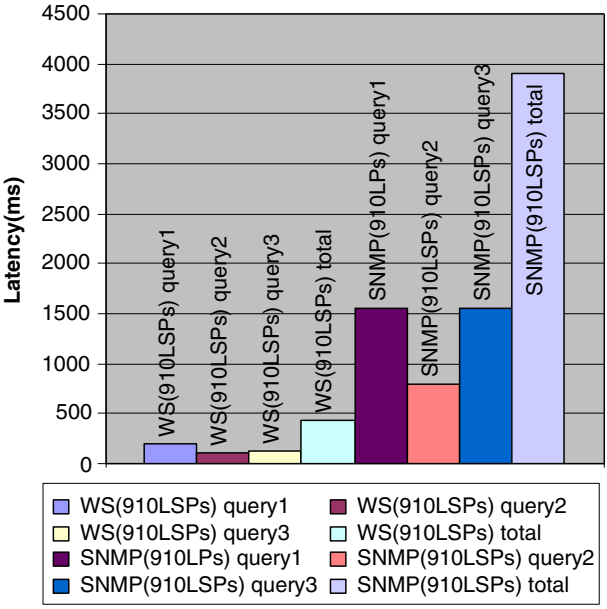

Fig. 14 Latency measurements for WS & SNMP for a big network

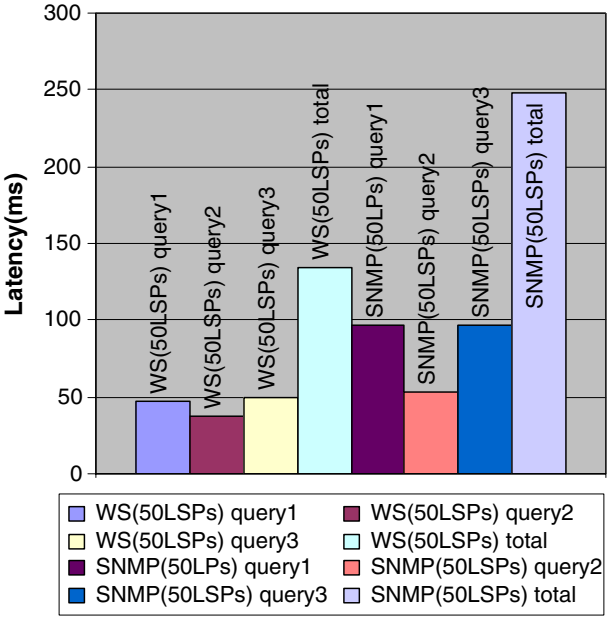

Fig. 15 Latency measurements for WS & SNMP for a small network

to be acquired, since filtering is not supported and processing has to be done at the manager. The total traffic for SNMP, using GetBulk, is 6,890 and 126,360 octets, and the total latency is 247 and 3,902 ms for small and big networks, respectively. Even if more LSPs and SLSs (more data to be retrieved for WS) were affected by the failing interface, it is clear that WS exhibit better performance to SNMP in this case and at the same time meet the objectives for scalable management. The only case where SNMP might have shown better performance than WS in this scenario is if very small networks were investigated so that the amount of information to be retrieved or processed would be very small; but such QoS-enabled networks do not exist in practice.

6 Conclusions

This paper demonstrates the importance of minimizing the processing overhead when performing network monitoring tasks. In contrast to what other researchers have done in this field, we chose to investigate monitoring in cases where task and load distribution are required to support tasks such as bulk and selective retrieval by performing data processing operations. We more specifically investigated the use of a query tool for bulk and selective retrieval and for distributing monitoring load in the context of two technologies, WS and SNMP. Our study has shown than while both technologies can be used to perform distributed monitoring, security problems, limited distributed management capabilities, luck of industry support,

and increasing development costs led the management community to practically abandon the distributed management extensions of SNMP. On the other hand, the industrial support of WS has led to the development of standards supporting distributed management, security, event reporting, transaction support, bulk and selective retrieval for monitoring and event reporting. Still, our research has shown that use of certain tools with these standards is not always the best option. The use for example of the heavyweight, in terms of resources, XPath language along with standards such as MUWS and WS-Management impairs the use of WS for performing monitoring tasks. XPath can introduce 10–16 times more latency as shown in [11] compared to a custom query tool we have designed and deployed. MUWS and WS-Management [23] are less lightweight than SNMP. For these reasons, we have designed and deployed our own framework to test the scalability and performance of WS for polling based monitoring. With such an effort we want to demonstrate that more lightweight solutions can be used inside a network domain for monitoring, whereas more heavyweight standards can be used at the edges for interoperability. As we have shown, this is more than possible since our query tool has good integration potential with standards such as MUWS.

Based on the above we investigate and compare the performance of WS against that of SNMP using a realistic case study. The latter is that of a monitoring system of a network that provides QoS guarantees over MPLS enabled devices. As part of this investigation we have: (a) analyzed the measurements required for QoS; (b) shown how to perform these measurements with the MPLS MIBs both for SNMP and WS; and (c) chosen three scenarios to demonstrate how the monitoring process can benefit from bulk and selective information retrieval.

In the first scenario data are retrieved with both WS and SNMP in a bulk manner. This is similar to what previous researchers have done, with the difference that we used literal encoding for the data to minimize the verboseness of XML tags in order to minimize XML coding latency and traffic overhead. We demonstrated that WS can perform a lot better in terms of traffic overhead and latency in comparison to retrieving data with consecutive SNMP getNext operations. We also showed that when the required management information increases, the latency of WS becomes less compared to SNMP's GetBulk operation, mainly because SNMP agents do not support caching. We have also shown that a newer SOAP toolkit (Axis 2.0 ver 1.4), as opposed to Axis v1.3 used in this work, can further improve the performance of our WS monitoring framework. Therefore, as long as care is taken to meet management objectives, and improved versions of XML tools are used to deploy a WS framework, WS can potentially become a scalable technology for monitoring and event reporting.

The second scenario uses the WS framework we have developed for distributing the monitoring load for processing management data at the agent side in order to receive the information in a specific manner (e.g., per PHB, SLS). This is a way to show how important it is to minimize the processing overhead of WS monitoring and event reporting operations in addition to minimizing the serialization and parsing overhead of WS toolkits. With this scenario we also demonstrate that by distributing the monitoring load to several agents the manager is relieved from this task. The latter is a more scalable option for WS compared to

technologies such as SNMP. Although SNMP could support this scenario through its distributed management extensions, the use of the expression MIB is not a scalable option as it increases the latency and traffic overhead. Without distributed extensions, an SNMP manager would have to process a lot of information as several agents must be monitored, for example in the order of hundreds for a large ISP network, which would introduce considerably more latency. Retrieving data on a per PHB or SLS basis directly from the agent in this scenario also shows that such an operation would produce more traffic for GetBulk, since all the OIDs of the required objects would have to be defined in an appropriate order in the request packet. This also makes GetBulk latency worse than that of WS on some occasions because of the lack of caching. Our lightweight parser and query tool allow processing of management data at the agent, thus meeting any granularity requirements for monitoring, which may not be true for SNMP short term measurements.

In the third scenario, the superiority of using our query tool for deploying WS is demonstrated for information retrieval operations that require filtering. The traffic overhead and latency in this scenario are in some cases 25 and 15 times less than the equivalent SNMP ones. This might not be the case if XPath was used as shown in [10, 11].

Despite the three listed benefits of our query tool in Sect. "4.2.1", our query tool also has some shortcomings. The measurements in the second scenario suggest that the query tool has room for improvement as more compact queries for example would improve its latency and traffic overhead consumption.

The approach and measurements presented in this paper suggest that WS can be used for distributed management meeting the monitoring requirements of a complex environment. Distributing the processing load to the agents is extremely important for WS, resulting in a more distributed and scalable system that can support sophisticated monitoring and event reporting operations. Of course, our agent, query tool and any WS framework that supports it would need to be installed in routers, interfacing with the services providing elemental management information. For this reason, the query tool was designed so that it can be used as an add-on service functioning over existing services. This way it could be part of any standard framework, such as MUWS, since it is in the plans of this framework to also allow the use of resource specific query tools. Furthermore, MUWS allows defining and exploiting the relationships between state data which our query tool supports for more effective monitoring. Security features need also to be incorporated in our minimal framework. It is in our future incentives to use implementations of WS security standards such as WSS4 J with our framework.

In summary, we hope that sophisticated WS management interfaces and lightweight tools will be supported in future managed devices, supporting sophisticated distributed management. Realistic scenarios in this article demonstrate that WS-based approaches performing load and task distribution can exhibit good performance, in addition to expressiveness in addressing management monitoring needs.

Acknowledgments The research work in this paper was partly supported by the EU EMANICS Network of Excellence on the Management of Next Generation Networks (IST-026854) and the IST ENTHRONE phase 2 projects.

References

1. Pavlou, G.: On the evolution of management approaches, frameworks and protocols: a historical perspective. JNSM **15**(4), 425–445 (2007)
2. Pras, A., Drevers, T., et al.: Comparing the performance of SNMP and web services-based management. IEEE. Trans. Netw. Serv. Manag. **1**(2) (2004)
3. Pavlou, G., Flegkas, P., Gouveris, S.: On management technologies and the potential of web services. IEEE Commun. Mag. **42**(7), 58–66 (2004)
4. Davis, D., Parashar, M.: Latency performance of SOAP implementations. 2nd IEEE ACM International Symposium on Cluster Computing and the Grid (2002)
5. van Engelen, R.: Code generation techniques for developing light-weight XML WS for embedded devices. ACM symposium on applied computing, pp. 854–861 (2004)
6. Govindaraju, M., Slominski, A., et al.: Towards characterizing the performance of SOAP implementations. 4th ACM international symposium on grid computing (2004)
7. Werner, C., Buschmann, C., et al.: Compressing SOAP messages by using differential encoding. IEEE international conference on web services, pp. 540–547 (2004)
8. Srinivas, D., Fremantle, P., Suriarachchi, A., et al.: Web services are not slow—Axis2/Java—performance testing round #1. WS02 oxygentank the developer portal for soa (2006)
9. Srinivas, D., Fremantle, P., Suriarachchi, A., et al.: Web services are not slow—Axis2/java performance testing round #2—Apache Axis2 vs. Codehaus XFire in a Contract First Scenario. WS02 oxygentank the developer portal for SOA (2007)
10. Chourmouziadis, A., Pavlou, G.: Web services monitoring: an initial case study on the tools perspective. NOMS (2008)
11. Chourmouziadis, A., Pavlou, G.: An evaluation of web services tools to address efficient information retrieval for network management. 8th International symposium on computer networks (ISCN 2008), pp. 108–114 (2008)
12. Chourmouziadis, A., Pavlou, G.: Efficient WS event reporting and notifications by task delegation. DSOM2007, vol. 4785/2007, pp. 242–255 (2007)
13. Graham, S., Murray, B.: Web services base notification 1.2 (WSBaseNotification). Working Draft 03 (2004)
14. Arora, A., et al.: Web services for management (WS-management). Desktop Management Task Force (DMTF), DSP0226 (2006)
15. Thompson, H.S., Beech, D., Maloney, M., Mendelsohn, N.: XML schema part 1: structures (2nd edn). W3C recommendation (2004)
16. Biron, P.V., Malhotra, A.: XML schema part 2: datatypes (2nd edn). W3C recommendation (2004)
17. Box, D., et al.: Web services eventing (WS-eventing). W3C member submission (2006)
18. Alexander, J., et al.: Web services enumeration (WS-enumeration). W3C member submission (2006)
19. Alexander, J., et al.: Web services transfer (WS-transfer). W3C member submission (2006)
20. Nadalin, A., Kaler, C., Hallam-Baker, P., Monzillo, R.: Web services security: SOAP message security 1.0 (WS-Security 2004). OASIS Standard 200401 (2004)
21. Vambenepe, W.: Web services distributed management: management using web services (MUWS 1.0) part 1. Organization for the Advancement of Structured Information Standards (OASIS) (2006)
22. Wilson, K.: Web services distributed management: management of web services (MOWS 1.0). Organization for the Advancement of Structured Information Standards (OASIS) (2006)
23. Moura, G.C.M., Silverstrine, G., et al.: On the performance of web services management standards—an evaluation of MUWS and WS-management for network management. IM, pp. 459–468 (2007)
24. Choi, M., Hong, J., Ju, H.: XML-based network management for IP networks. ETRI J. 445–463 (2003)
25. Neisse, R., et al.: Implementation and bandwidth consumption evaluation of SNMP to web sevices gateways. NOMS pp. 715–728 (2004)
26. Fioreze, T., et al.: Comparing web services with SNMP in a management by delegation environment. IM pp. 601–614 (2005)

27. Lima, W.Q., et al.: Evaluating the performance of SNMP and web services notifications. NOMS. pp. 546–556 (2006)
28. Clark, J., et al.: XPath version 1.0. W3C recommendation (1999)
29. Berglund, A., et al.: XPath version 2.0. W3C recommendation (2007)
30. Enns, R.: NETCONF, draft-ietf-netconf-prot-11, http://www.ietf.org/internet-drafts/draft ietf-netconf-prot-11.txt (2006)
31. Chourmouziadis, A., Pavlou, G.: Efficient information retrieval in network management using web services. DSOM 2006 Proceedings, October 23–25 (2006)
32. Yoo, S-M., et al.: Performance improvement methods for NETCONF-based configuration management. APNOMS pp. 242–252 (2006)
33. Cridlig, V., et al.: A NetConf network management suite: ENSUITE. IPOM 152–161 (2005)
34. Ye, W.: Web services programming tips and tricks: improve interoperability between J2EE technology and.NET, part 1-WSDL, RPC/encoded style, and WS-I conformance. IBM article, http://www.ibm.com/developerworks/webservices/library/ws-tip-j2eenet1/ (2004)
35. Saiedian, H., Mulkey, S.: Performance evaluation of eventing web services in real-time applications. IEEE Commun. Mag. **46**(3), 106–111 (2008)
36. Levi, D., Schoenwaelder, J.: Definitions of managed objects for the delegation of management scripts. RFC 3165 (2001)
37. Kavasseri, R., Stewart, B. Distributed management expression MIB. RFC 2982 (2000)
38. Straub, F.: Advantages and disadvantages of the script MIB infrastructure. Script MIB project report (2000)
39. Schoenwaelder, J.: Traditional approaches to distributed management. NMRG meeting 19, Stockholm (2006)
40. Lopes, R., Oliveira, J.: Delegation of expressions for distributed SNMP information processing. IM pp. 395–408 (2003)
41. Daniele, M., et al.: Agent extensibility protocol version I. RFC2257 (1998)
42. Martin-Flattin, J.P.: Web-based management of IP networks and Systems. Wiley Series, http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0471487023.html (2003)
43. IETF DISMAN.: Proceedings, December 97, http://www3.ietf.org/proceedings/97dec/97dec-final-67 (1997)
44. Schoenwaelder, J.: Overview of the 2002 IAB network management workshop. IETF Informational RFC 3535 (2003)
45. Cline, K., Kohen, J., Davis, D., et al.: Towards converging WS standards for resources events and management. White paper by IBM, HP, Intel and Microsoft (2006)
46. Asgari, A., Egan, R., Trimintzios, P., Pavlou, G.: Scalable monitoring support for resource management and service assurance. IEEE Netw. **18**(6), 6–18 (2004)
47. Asgari, A., Trimintzios, P.: Building quality-of-service monitoring systems for traffic engineering and service management. JNSM **11**(4), 399–426 (2003)
48. Creation and Deployment of End-User Services in premium IP networks (CADENUS): http://wwwcadenus.fokus.fraunhofer.de/(2000)
49. End to End QoS through Intergrated Management of Content, Network and Terminals (ENTHRONE): http://www.enthrone.org/ (2006)
50. Goderis, D., et al.: Service level specification semantics and parameters. Internet draft (draft-tequila-sls-02.txt) (2002)
51. Blake, S., et al.: An architecture for differentiated services. RFC 2475 (1998)
52. Srinivasan, C., Bloomberg, L.P., Viswanathan, A.: Multiprotocol label switching (MPLS) label switching router (LSR) MIB. RFC 3813 (2004)
53. Nadeau, T., Srinivasan, C., Bloomberg, L.P., Viswanathan, A.: Multiprotocol label switching (MPLS) forwarding equivalence class to next hop label forwarding entry (FEC-To-NHLFE) MIB. RFC 3814 (2004)
54. Berners-Lee, T., Fielding, R., Masinter, L.: The uniform resource identifier (URI): generic syntax. RFC 3986 (2005)
55. Pavlou, G., Liotta, A., Abbi, P.: CMIS/P++: extensions to CMIS/P for increased expressiveness and efficiency in the manipulation of management information. 7th Annual joint conference of the IEEE Comput. Commun. Soc. INFOCOM. 98, vol. **2**, pp. 430–438 (1998)
56. Case, J., Fedor, M., et al.: A simple network management protocol (SNMP). RFC 1157 (1990)
57. ITU-T X.690.: Information technology—ASN.1 encoding rules: specification of basic encoding rules (BER), canonical encoding rules (CER) and distinguished encoding rules (DER) (2002)

Author Biographies

A. Chourmouziadis is a Researcher at the Centre of Communication Systems Research, in the University of Surrey, UK. He holds a diploma in Electrical and Computer Engineering from the Aristotle University of Thessaloniki, Greece, an MSc in Multimedia Signal Processing and Communications and a PhD both from the University of Surrey, UK. His research interests are in web services based network management, quality of service management, management technologies, communications middleware, traffic engineering, signal and video processing, video compression and coding and multimedia technologies.

M. Charalambides is a Research Fellow at the Networks and Services Research Laboratory, Department of Electronic Engineering, University College London, UK. He holds a B.Eng. in Electronic and Electrical Engineering and an MSc in Communications Networks and Software, both from the University of Surrey where he is also a PhD candidate. His research interests focus on policy-based management, policy analysis, and IP quality of service.

G. Pavlou is a Professor of Communication Networks at the Department of Electronic and Electrical Engineering, University College London, UK where he coordinates the activities of the Networks and Services Research Lab. He holds a MEng in Engineering from the National Technical University of Athens, Greece, and MSc and PhD degrees in Computer Science from University College London, UK. He has been responsible for a number of European and UK research projects and industrial collaborations. His research interests focus on networking, network management and service engineering, including aspects such as network dimensioning, traffic engineering, quality of service management, policy-based systems, infrastructure-less wireless networks, autonomic networks and communications middleware. He has contributed to standardization activities in ISO, ITU-T, TMF, OMG and IETF.