Efficient Web Services Event Reporting and Notifications by Task Delegation

Aimilios Chourmouziadis, George Pavlou

Centre for Communication Systems Research, School of Electronics
and Physical Sciences, University of Surrey,
GU27XH Guildford, United Kingdom,
{A.Chourmouziadis, G.Pavlou}@surrey.ac.uk
http://www.ee.surrey.ac.uk/CCSR/

Abstract. Web Services are an XML technology recently viewed as capable of being used for network management. A key aspect of WS in this domain is event reporting. WS-based research in this area has produced a collection of notification specifications, which consider even aspects such as filtering to reduce machine and network resource consumption. Still though, additional aspects need to be addressed if WS event reporting is to be used efficiently for network management. This paper borrows an idea in network management that of policy based task delegation and applies it in the context of WS-based management by using the WS-Notification standard messages, to increase event reporting efficiency. More specifically, we are adding functionality to the entity that produces events making it capable of performing a set of tasks apart from simple ones such as collecting and reporting notification data. This functionality allows an entity, such as a manager, capable of delegating tasks of various complexities to an event reporting entity where they can be performed dynamically. As a proof of concept that the approach is feasible and increases efficiency we analyze a complex event reporting scenario where task delegation is used. We compare this approach for performance to a plain WS-based event system and also to simple SNMP traps.

1 Introduction

The growing use of the eXtensible Markup Language (XML) for data representation, coupled with the development of many XML standards and technologies such as Web Services (WS), has spurred research in a variety of fields other than the ones these technologies were originally designed for. One such field is network management.

One significant aspect of network management is event reporting. To use WS for event reporting two problems have to be addressed (a) asynchronous communication (push) (b) efficiency. The former is required since the time of the production of an event is not known and thus synchronous (pull) style communication is not possible. Efficiency is also an important aspect since for example it does not make sense to produce events that nobody is interested in receiving, or to produce events that someone is not interested to receive as this will make unnecessary use of resources.

In order to provide asynchronous communication between WSs, a callback mechanism is required. A Uniform Resource Locator (URL) is such a mechanism but is inadequate since (a) it only allows a single protocol to be defined to reach a service, (b) it can not describe all transport mechanism types, and, (c) it doesn't necessarily convey interface information. The proprietary WS-Addressing [1] specification solved this problem by defining two mechanisms that can be used as an efficient

callback mechanism: (a) endpoint references, (b) message-information headers [2]. Despite its drawbacks [3] this specification has opened the way for three specification documents to be defined: WS-Events [4], WS-Eventing [5], and WS-notification [6].

In the HP WS-Events specification [4], the consumer of an event can (a) discover event-types an event producer supports, (b) subscribe to an event, (d) perform data filtering, (e) define an expiration date for receiving events, and, (f) provide a callback URL for an event. Filtering mechanisms are not specified in [4] but the means for unwanted events not to be produced or consumed are provided. In WS-Eventing [5] things become clearer; this specification supports the XML Path (XPath) for event filtering and WS-Addressing to provide a better callback mechanism. In WS-Notification [6] more features are added. [6] allows (a) consumers to receive content in an application-specific or raw format (b) define several types of expressions for filtering (XPath etc), (c) define event-types a consumer needs to receive with expressions called topics, (d) provide support for notification brokering.

All the above standards are on the right track for providing efficient and reliable event reporting communication. Still WS notifications can be used more efficiently for network management. Consider the management scenario where a manager has to be notified when an interface of a Quality of Service (QoS) enabled network fails. Upon receiving this event, the manager needs to determine the traffic contracts affected and requests for more data. In cases such as the previous, event reporting triggers actions at the event receiver which in turn requests for more system data or performs other changes i.e. configuration. Finding a way to perform a set of actions, normally performed by the entity receiving an event, in order for the tasks to be performed by the entity producing them, would make the notification process more efficient. The process where an entity is given the task to perform a set of actions for another entity is called task delegation. Task delegation can be used for WS-event reporting as long as the entity with the responsibility to perform a set of tasks is not a very resource-constrained system. This is more a reality today [7] (dumb agent myth).

Using WS-based event reporting with task delegation can be important for two reasons. The first one applies to data retrieval. In many event-reporting scenarios event data represents a small amount of the data carried over the network in comparison to the HTTP and the Simple Object Access Protocol (SOAP) header data. The use of WS notifications is not justified in these cases since WS perform badly when retrieving small amounts of data [8], [9]. As such, adding additional data, normally retrieved after the receipt of an event, in the initial report in order to reduce latency and traffic overhead can be beneficial. Secondly by task delegation a higher degree of autonomy can be achieved as the manager's supervision is limited.

A prominent way to perform task delegation for WS-based event reporting is through policies and the WS-Notification messages support their use. Delegating tasks though policies to improve the communication between entities in the event reporting process is not a new idea. Applying it to WSs to check if it is feasible and if potential benefits can be gained from it, is something that needs to be explored. As such a WS-based event service has been built supporting task delegation with the use of WS-Notification messages and policies. To prove the viability and the gains of the approach, an event reporting scenario is analyzed based on a QoS enabled network. We analyze the performance of event reporting for three systems: (a) A WS-based notification system where only event data are reported and then a set of actions

triggered by the event are performed to collect more data (b) A WS-based event system where event data and data collected from subsequent tasks are gathered and sent by the entity that produces events in the initial report (c) An SNMP trap system.

The remainder of this paper is structured as follows. In section 2, details of the event reporting scenario based on a QoS-enabled Traffic Engineered (TE) on which we will comparing the three systems are provided. Section 3 analyzes the WS-notification standard messages used for event reporting in our scenario and it is shown how to use these messages to configure our event service to perform a set of tasks of varying complexity. Section 4 presents the WS-notification compliant messages that need to be sent for configuring an event service for handling event tasks, and the interactions between the different entities of the event reporting process. Section 5 presents a perfornance evalution between the two WS-based systems and a system based on SNMP traps. Section 6 presents our conclusions.

2 QoS Event Reporting Scenario

2.1 QoS management system

Providing QoS in a single or across different domains is a widely researched topic. QoS is currently provided on the basis of Service Level Agreements (SLAs). An SLA is a set of terms that clients and providers of services have to abide by when they are accessing or providing a service respectively. The technical part of an SLA is a Service Level Specification (SLS) and it represents the means to define QoS-based IP services [10]. IP Differentiated Services [11] (DiffServ) is considered the most prominent framework for providing QoS-based services. All QoS-based services are quantified by means of performance parameters such as throughput, delay, loss and delay variation. One of the means to support the DiffServ architecture is over Multi-Protocol Label Switching (MPLS) traffic engineered networks.

Monitoring and event reporting of the network status and its resources is an essential process in order to ensure a QoS network's operation. To ensure the latter the use of Traffic Engineering (TE) is required. TE requires the collection of various data in order to ensure the network's smooth operation. This is achieved by a suitable monitoring and event reporting system which is scalable in terms of network size, customers' size etc. This constitutes a significant challenge in QoS-networks.

Previous examples of research in monitoring and event reporting has been performed in the TEQUILA [12], and the ENTHRONE frameworks [13]. These systems used in these frameworks are mostly based on the Manager-Agent paradigm. This is the paradigm we also adopted to collect data either with WS or SNMP for event reporting (Fig. 1). This system performs three kinds of operations: active, passive measurements and event reporting. Active measurements are performed by injecting synthetic network traffic. Passive measurements are conducted using Management Information Bases (MIBs) from SNMP and involve measuring throughput, load and packet loss at the traffic class (Per Hop Behavior-PHB), traffic contract (Service Level Specification-SLS) and the path (Label Switched Path - LSP) level. In Fig. 1, the manager is responsible for configuring software on the agents attached to the routers it needs to retrieve data from so as to perform active or passive

measurements or event reporting. The agent operates either on a dedicated PC attached to a router or, if future routers support such functionality, on the router itself.

To perform measurements for our scenario at the PHB, LSP or SLS level we selected two of the SNMP MPLS MIBs to represent management data. These are the Label Switching Router (LSR) MIB [14] and the Forwarding Equivalence Class to Next Hop Label Forwarding Entry (FEC-To-NHLFE) MIB [15]. The former is used to perform PHB and LSP measurements and the latter is used for SLS measurements. For WS, equivalent MIBs had to be built and be deployed as WS interfaces.

2.2 Management Information Retrieval for QoS WS-based event reporting

Retrieving management data from a managed system in our scenario requires facilities to be able to pick data in a bulk or selective way. This is achieved by the parser presented in [16]. Selective and bulk retrieval is achieved by dispatching appropriate queries. The reason behind building our own parser is to keep resource usage, latency and traffic overhead low. Thus selective retrieval at SOAP level is not an option since more data than required would have to be retrieved, encoded and selected. At the same time XPath can be a heavy-weight tool for management tasks especially if large documents need to be searched or loaded in memory. In addition processing raw data is less intesive than processing a verbose XML document.

We use our parser in our event reporting scenario to collect management data.

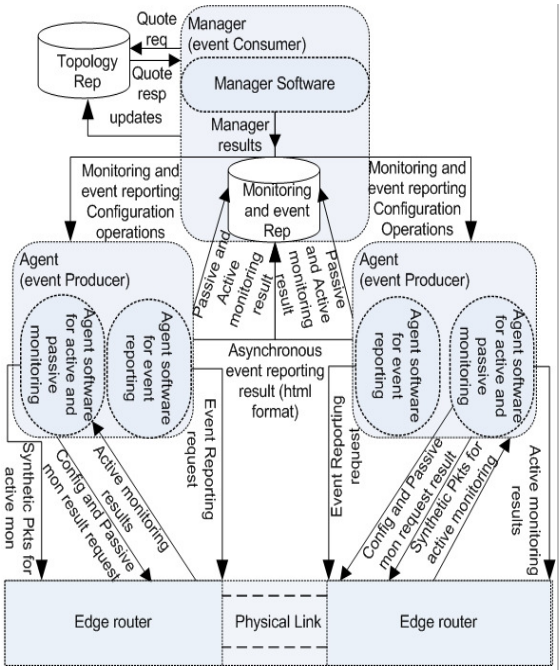

Fig. 1. Management system for monitoring and event reporting (manager-agent paradigm)

```
<wsnt:Subscribe>
 <wsnt:ConsumerReference>
   http://131.227.88.70:8080/
   notifications /notifications_
   Consumer
 </wsnt: ConsumerReference>
 <wsnt:TopicExpression dialect=
  "http://131.227.88.70/eventTopics">
   tns:notify-down
 </wsnt:TopicExpression>
 <wsnt:UseNotify> True/False
 </wsnt:UseNotify>?
 <wsnt:Precondition>
   wsrp:QueryExpression
 </wsnt:Precondition>?
 <wsnt:Selector>
   wsrp:QueryExpression
 </wsnt:Selector>?
 <wsnt:SubscriptionPolicy>
   Event-Condition-Action
   Policy-like XML document
 </wsnt:SubscriptionPolicy>?
 <wsnt:InitialTerminationTime>
  2007-03-11T13:00:00
 </wsnt:InitialTerminationTime>?
</wsnt: Subscribe>
```

Fig. 2. WS Notification Subscription message [6]

2.3 QoS Event reporting Scenario

We consider an event reporting scenario in which the manager is notified that an MPLS interface failed in the ingress router. Upon receiving this event the manager needs to collect more data so as to determine the LSPs and SLSs that are affected by the failing interface so as to take appropriate measures. To apply this event reporting scenario, we have built and deployed a WS event service (Fig. 1) at the agent side. The event service is configured to perform a number of management tasks normally performed after the receipt of a notification, dynamically before dispatching the event report to the manager. By having the manager delegate tasks to other management entities its burden is minimized and the event process becomes more efficient. To demonstrate the benefits of such an approach, two WS-based event reporting approaches are considered which are analyzed in section 5. For both approaches WS-Notification compliant messages are used to configure the event source for notifications and for event reporting. A comparison to SNMP traps is also provided.

3 WS-Notification messaging for event reporting

The WS-Notification family of specifications defines a system architecture to support WS-based event reporting. In this architecture a *publisher* is an entity sending notifications about a range of events called *topics* to other entities called *consumers*. *Brokers* are defined as intermediate entities between producers and consumers controlling the flow of events with filtering. For a consumer to receive events it must register with the broker or the producer by selecting the appropriate topics.

WS-Notification defines the features and messages exchanged between entities participating in the event reporting process. In this paper we are only interested in (a) the request message a consumer sends to a producer to register for an event topic, (b) the response to the request message, and, (c) the event messages the producer sends to the consumer. We do not tackle aspects such as brokering, topic filtering, etc, as these are out of the scope of our scenario. We demonstrate the use of WS-Notification messages to (a) configure the event service we have built for event reporting and task manipulation, (b) report events and, (c) investigate potential benefits of adding varying complexity tasks to the event producer. As such, in the next two sections we only address WS subscription (request and response) and notification messages.

3.1 The WS Notification Subscription message

The WS-Notification specification defines that in order for an event consumer to receive a notification from a producer, it has to send a subscription message. The format of such message is given in Fig. 2. Here, the *consumer reference* tag is a URL providing a call-back mechanism for event delivery. The *topic expression* tag defines the event topics a consumer can register to receive. Our event service implementation supports four general topics, (a-b) a threshold is exceeded going upwards-downwards (notify-high or notify-low), (c-d) the state of a unit has changed to active-inactive (notify-up or notify-down).The *UseNotify* tag is used by a consumer to select whether

events will be formatted in an application specific way or in a WS-Notification *Notify* message. In addition, the selector and precondition expressions are used for data filtering. To define the period for which an event consumer registers for events, the termination time of the subscription has to be specified (*InitialTerminationTime*).

In the subscription message, the *subscription policy* element is a component used to specify application-specific policy requirements/assertions. The semantics on how an event producer will react to these assertions depends on the application-specific grammar used. A non -normative way to define policies is the WS-Policy standard. The greater vision of IBM for using the policy element is to be able to define concrete policies that allow a service to describe its approaches for subscription management or to specify directives that the event source must follow.

The response to a subscription may contain lots of data. Primarily though it contains the address of a WS defining messages that can be exchanged to manipulate subscription resources and fault information for subscription failure.

The *Notify* message contains the following: (a) a *topic* header that describes the event topic an event consumer subscribed initially to receive (b) a *producer reference* element that describes the endpoint of the service that produced the event, and (c) *message* elements where the actual payload of a notification is inserted. Our event service supports both *Notify* and application specific messages.

3.2 Policy-like configuration of Events for network management

Apart from the IBM specifics on policies, the vision of policies for network and service management is described in [17]. According to [17] policies are an aspect of information influencing the behavior of objects in a system. All policies can be expressed as a hierarchy where a high level policy goal can be refined into multiple levels of lower level policies. Effectively policies are rules used as the means to successfully achieve a goal. Furthermore, policies can be broadly classified into (a) *authorization* policies that define what is permitted, or not, to be performed in a system, and, (b) *obligation* policies that define what must be performed, or not, in order to guide the decision making process of a system. Both types of policies can be defined using an event-condition-action model of definition. Thus it is evident that policies can be reduced to set of rules, actions, utility functions that can be used to (a) ensure compliance, (b) define behavior, and, (c) achieve adaptability of a system.

In the network management world events are viewed as a state that usually demands an action to be taken. An event can be comprised of information about (a) the event itself, (b) the condition that produces it, and, (c) the type of actions to be performed after event generation. All this information is consistent with the network management view of policies (event-condition-action). As such, it is possible to use WS-Policy or any domain specific grammar to configure an event process as a policy. Thus the *subscription policy* element and any domain specific grammar can be used so as to pass to an event service (event producer), data in order to configure the event information, the event production condition and any event tasks-actions as policies.

In our event service implementation we use the *subscription policy* element to send to the event producer an XML document that consists of three sections: (a) general event data, (b) the conditions that trigger event-production, and, (c) subsequent actions. This document configures events as policies and its grammar is validated

through an XML schema. Details on this are given in section 4. The grammar configuring events as policies constitutes by no means a formal policy. Still we can use it within the WS-Notification subscription policy element so as to be able to configure an event source to perform a set of varying complexity tasks. This allows us to delegate a set of tasks that the manager would otherwise perform to other entities (event service) so that WS event reporting is made more efficient.

4 Event Reporting Scenario Operations

4.1 Event reporting process description

To configure the event service developed for the QoS event reporting scenario that we presented, the event consumer has to send a subscription message to the event producer. In reference to Fig. 1, the consumer is the manager and the producer is the agent. In Fig. 3 these roles are assumed by XML SOAP messaging services and WSs.

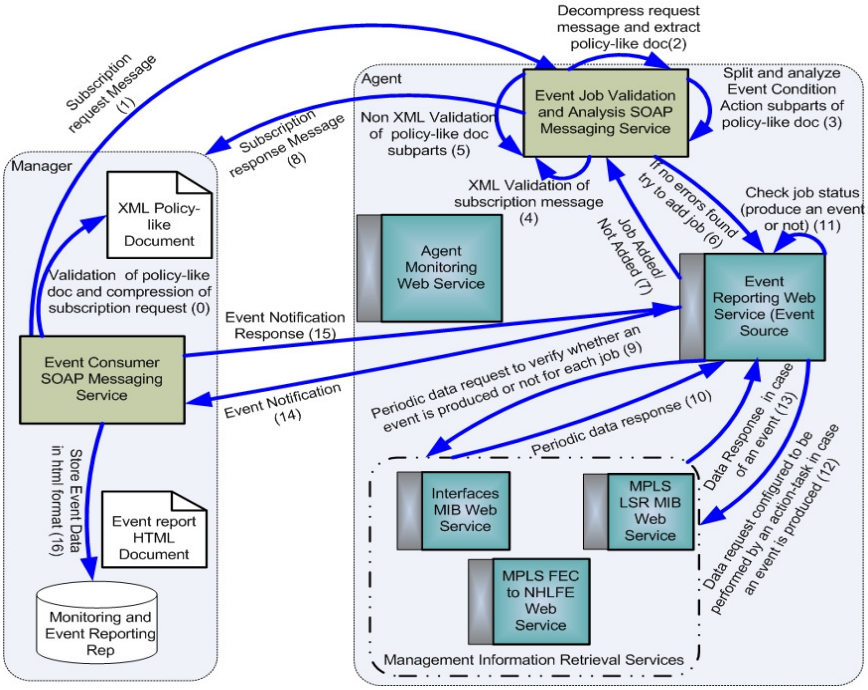

Fig. 3. Components interaction for event reporting

An overview of the operations that need to be performed for a receiver of events to actually start receiving notifications is given in Fig. 3. Here the subscription process starts by validating the event condition action policy-like document to avoid subscription request failure. Then the request is compressed and sent to the agent. At the agent the subscription request is decompressed, the policy-like document is extracted and split into its event-condition-action sub-parts. After SAX parser validation of each message part, the XML policy-like document is also searched for

any discrepancies not captured by XML validation. This is necessary since inter-dependencies between different elements of the policy-like document exist and cannot be expressed by an XML schema. If errors are found the manager's SOAP messaging service is notified. On the opposite case, the agent's messaging service adds an event job to the event service. An event job can still be rejected for various reasons (job exists etc). Successful or unsuccessful addition of a job is reported to the manager. Apart from adding a job, the event service supports features such as (a) resume, (b) suspend, (c) remove, and, (d) update.

Upon successful addition of a job, the event sub-part is processed, and event data are collected using the Java reflection API to dynamically invoke the appropriate WSs exposing management data. Selective data retrieval is performed using the parser developed in [16]. Because we use our parser to filter data when collecting it, the *selector* and *precondition* expressions offered by the WS-Notification standard for filtering are not used. Following the data collection phase, the condition part of the policy-like document is processed to determine whether an event has been produced. If no event is produced the process is repeated according to the granularity of operations. If an event is produced, the action sub-parts of the policy-like document are executed. The actions in our event reporting scenario involve tasks to gather extra data to determine the LSPs and SLSs affected by a failing interface. Calling the appropriate WSs to gather these data is performed dynamically and any queries to retrieve management data are formed on the fly using recursive methods since these queries can contain data not known in advance. When the event data and data from the configured tasks are collected, an event report is sent to the manager which confirms its receipt. The event report data is stored in HTML format.

4.2 Policy-like event configuration document

The policy-like document consists of an event, a condition and an action part. The event part (Fig. 4) consists of sections which define (a) which parameter(s) need(s) to be monitored (*OIDstoMonitor*), (b) how to retrieve the data to be monitored (*EvenTask and its sub-elements*), and, (c) how to handle and process the retrieved data (*Result and its sub-elements*). The condition part of the document (Fig. 5) contains information to determine whether an event has been produced or not, such as (a) the type of monitor used (mean monitor, variance monitor, etc), (b) the measurement granularity, (c) the smoothing window size, and, (d) the clearing value that re-enables event

```
<ns:EventSpec name="" jobid="" date="" time="">
 <ns:OIDsToMonitor>…</ns:OIDsToMonitor> {1}
 <ns:EventTask actionid="">
  <ns:ServiceEndpoint>...
 </ns:ServiceEndpoint> {1}
  <ns:Method  namespace="">...</ns:Method> {1}
  <ns:Use>...</ns:Use> {1)
  <ns:Style>...</ns:Style>{1}
  <ns:MethodParams>
   <ns:Param name="" pmid="" namespace=""
    type="">
    <ns:Param> +
  </ns:MethodParams> ?
  <ns:Result resid="" type="" namespace=""
  qname="" name="">
   <ns:ResultParam pmid="" type="">...
   </ns:ResultParam>*
    <ns:ResultFormat forid=""  dependsON="">
     <ns:FormatValue>...</ns:FormatValue>?
     <ns:FormatPattern>...</ns:FormatPattern> ?
    </ns:ResultFormat> ?
  </ns:Result> *
 </ns:EventTask> {1}
</ns:EventSpec> +
```

Fig. 4. Event part of the policy like document

reporting if it has been disabled. The action part(s) of the document contains data on how to call the appropriate WS to perform a task and also on how to process the result of any WS calls (Fig. 6).

```
<ns:EventCondition jobrefid="">
  <ns:MonitoringObjectType     monid="">
    <ns:granularity>...</ns:granularity> {1}
    <ns:window>...</ns:window>{1}
  </MonitoringObjectType> {1}
  <ns:Threshold>
    <ns:tType>...</ns:tType> {1}
    <ns:value>...</ns:value> {1}
    <ns:clearvalue> </ns:clearvalue> ?
  </ns:Threshold> {1}
</ns:EventCondition> +
```

Fig.5. Condition part of the policy like document

```
<ns:ActionOnEvent jobrefid=""actionid="">
    <ns:ServiceEndpoint>...
  </ns:ServiceEndpoint> {1}
    <ns:Method  namespace="">...
    </ns:Method> {1}
    <ns:Use>...</ns:Use> {1)
    <ns:Style>...</ns:Style>{1}
    <ns:MethodParams>
      <ns:Param name="" pmid=""namespace=""
      type="">
        <ns:Param> +
    </ns:MethodParams> ?
    <ns:Result resid="" type="" namespace=""
    qname=""    name="">
      <ns:ResultParam pmid=""type="">...
       </ns:ResultParam>*
       <ns:ResultFormat forid="" dependsON="">
         <ns:FormatValue>...</ns:FormatValue>?
         <ns:FormatPattern>...</ns:FormatPattern> ?
       </ns:ResultFormat> ?
    </ns:Result> *
</ns:ActionOnEvent>
```

Fig. 6. Action part of the policy like document

5 Scenario Measurements

5.1 Evaluation Setup

For the evaluation aspects of our scenario, a big number of LSPs need to be setup for some measurements. As this is difficult in a small test-bed, we resorted to other means for evaluating the SNMP performance overhead. For traffic overhead, the average size of each message is calculated by looking into it and analyzing the size of its subparts. For latency a similar number and type of objects as in the MPLS MIBs are instantiated and the Advent-Net SNMP v3.3 is used to access a Net-SNMP agent. For WS, the software used is Apache Axis 1.3, JAXP 1.3, SAAJ 1.3 and JAXM 1.1. Java's zip facilities are used to compress/decompress messages and Java's reflection API was used to make WS dynamic calls. All MIBs are deployed with literal encoding so that the verboseness of XML tags is reduced and traffic overhead is minimized. The manager and agent were deployed on a 1000MHz/256MB RAM and 466MHz/192MB RAM machine respectively running Red-hat Linux 7.3, thus simulating a lower end system for the agent.

5.2 Measurements

The measurements presented in this section demonstrate the potential benefits of data filtering and task delegation for event reporting. Two WS-based approaches are examined. In the first, the manager is notified about a failing interface, and then queries the agent to determine the affected LSPs and SLSs. In the second approach the agent is configured by the manager to perform dynamically the set of tasks the

latter would otherwise perform, and sends back all the collected data. The second approach is more complex since it requires the event service to call the appropriate WSs to determine the affected LSPs and SLSs at run time. This also requires forming parser queries to retrieve management data on the fly, since these queries may also contain data that are not known in advance.Through task delegation and filtering we show that the second approach is plausible and results in traffic and latency benefits. SNMP's, traffic and latency performance was also measured for comparison.

For SNMP traffic overhead measurements we rely on previous research performed in [8] and [18]. In these papers the traffic overhead for SNMP operations is given by:

$$L_{get,\ getNext} \approx n1 * (54 + 12 + 2L_1 + L_2) \quad (1)$$

$$L_{getBulk} \approx 54 + 1 * (6 + L_1) + n_1 (6 + L_1 + L_2) \quad (2)$$

$$L_{trapSNMPv\ 1} = 49 + n_1 * (3 + L_1 + L_2) \quad (3)$$

$$L_{trapSNMPv\ 2} = 75 + L_3 + n_1 * (3 + L_1 + L_2) \quad (4)$$

In equations (1), (2), (3) and (4) L_1 is the size of the Object Identifier (OID) of a variable, L_2 is the variable value size, n_1 is the number of OIDs to retrieve and L_3 is the trap OID. Taking into account the size (Table 1) of the data that needs to be reported the traffic overhead for SNMP can be computed.

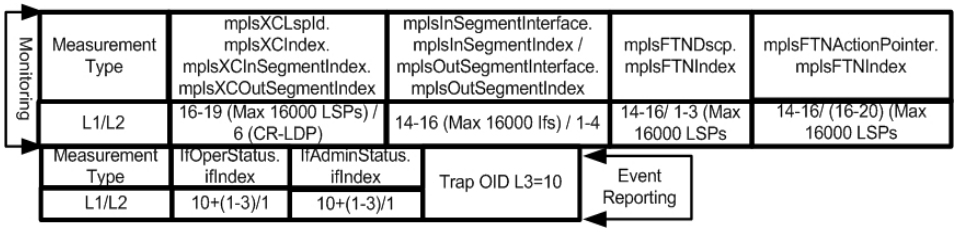

Table 1. Information size in ASN.1 format inside an SNMP message

For the measurements, the ingress router is configured to have 900 and 30 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 to the resulting event data. A further assumption is the number of LSPs and SLSs affected by the failing interface, which is considered to be six. Although it is not easy to determine a plausible number of LSPs assigned to each interface, six is a reasonable number for small networks. This number may not be realistic for 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 to SNMP even if a small volume of data is retrieved (not shown in [8] and [9]). Additionally, keeping the same number of affected SLSs and LSPs for both small and large networks we can keep the traffic latency comparison between them on the same terms.

For the manager or agent to determine, for each WS approach the affected LSPs and SLSs, three queries must be sent. These queries: (a) determine the interface indices of the LSPs associated with this interface, (b) use the previous step indices to

determine the affected LSPs, and, (c) determine the affected SLSs using the LSP IDs from the previous step. Parts of the three queries are the following:

$$\{\text{mplsInSegmentInterface}[\,], \text{mplsOutSegmentInterface}[\,]\} \quad (5)$$
$$\{\text{value} = \text{ifIndex}, \text{value} = \text{ifIndex}\}$$

$$\{\text{mplsXCLspId}[\,]\} \quad (6)$$
$$\{\text{mplsInSegmentXCIndex} = \text{mplsInSegmentIndex}_1 \text{ OR}$$
$$\text{mplsOutSegmentXCIndex} = \text{mplsOutSegmentIndex}_1 \text{ OR} \ldots\}$$

$$\{\text{mplsFTNDscp}[\,]\} \quad (7)$$
$$\{\text{mplsFTNActionPointer} = \text{mplsXCLspId.mplsXCIndex.}$$
$$\text{mplsXCInSegmentIndex.mplsXCOutSegmentIndex OR} \ldots\}$$

The measurements for our scenario are presented in Figures 7, 8, 9 and 10. In Fig. 7 configuration latency for the WS-based event services is quite significant since (de)compression of the subscription request, and XML validation takes place for both the first (S) and the second WS-based approach (C) (WS(C)/ WS(S) config). Configuring the event service though is not a time critical task and happens once for a specific event-job. Therefore, we do not consider configuration latency in the event reporting overall latency of the WS approaches since it is not a time critical task.

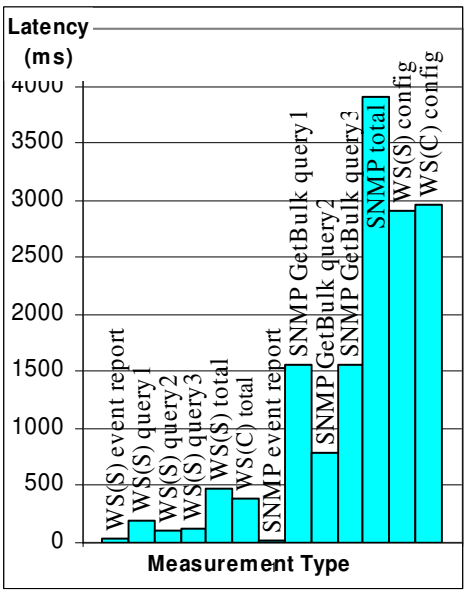

Fig.7. Latency measurements for SNMP and for the two WS based approaches (900 LSPs)

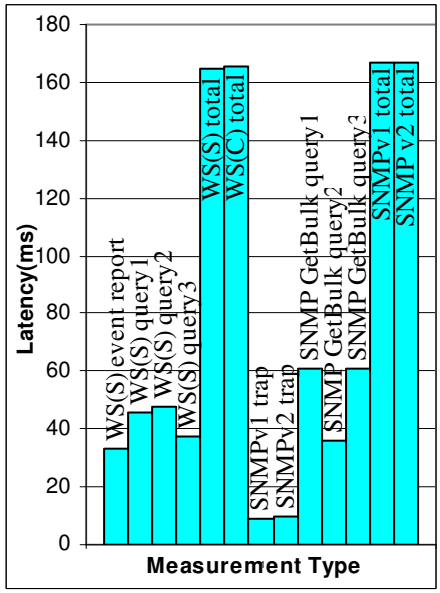

Fig. 8 Latency measurements for SNMP and for the two WS based approaches (30 LSPs)

Comparing the two WS-based approaches in terms of latency for small networks, it can be seen that the difference is very small (Fig 8). This occurs because the latency benefit from performing local WS calls for the WS approach with task delegation is counter-balanced from the latency incurred from performing dynamic WS calls and building data queries on the fly. For big networks though, latency decreases by around

75 ms if task delegation is used (Fig. 7). Comparing the WS-based approaches with SNMP, latency is about the same in the case of small networks (Fig. 8). For big networks SNMP suffers from a substantial increase in latency (Fig. 7). This occurs for two reasons, the first one being that SNMP does not offer facilities for task delegation so that data retrieval operations can be performed locally. The second reason is that SNMP does not offer filtering capabilities. Therefore determining the LSPs and SLSs affected from the failing interface requires retrieving more data than required from the relevant tables in the MPLS MIBs so as to be processed by the manager.

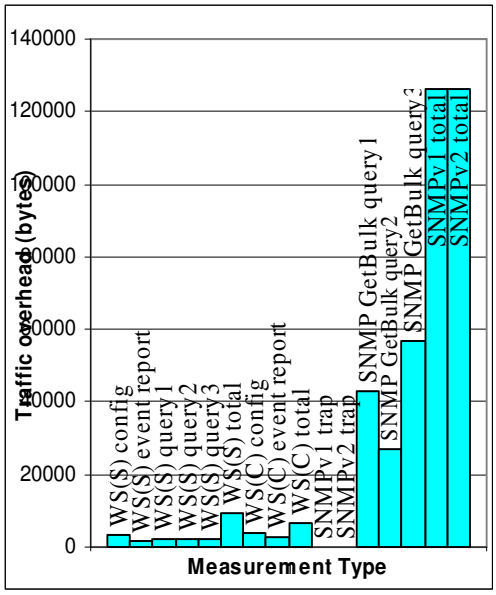

Fig. 9 Traffic measurements for SNMP and the two WS based approaches (900 LSPs)

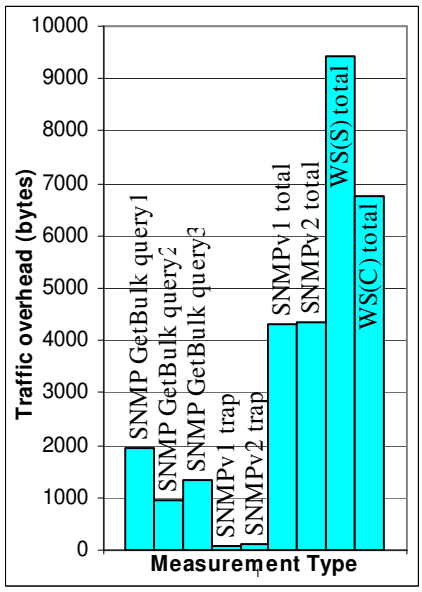

Fig.10. Traffic measurements for SNMP (30 LSPs) and total traffic for the WS schemes.

As far as traffic overhead is concerned 2700 bytes are saved by task delegation for both small and large networks (Fig. 9). This reduction occurs since SOAP and HTTP header data for the second WS approach are less. For every time an event is produced, our approach will save more traffic and latency. Therefore in the initial configuration of the event service we can select to monitor with a small expression all the interfaces of the ingress (relevant) router. The latter is not possible with SNMP without increasing traffic overhead since all MIB variables that need to be monitored must be defined. SNMP's traffic overhead for big networks is 120 kilobytes more due to lack of filtering and task delegation facilities (Fig. 10). For smaller networks SNMP's traffic overhead is less by 2300 bytes when compared to the WS approach based on task delegation (Fig 10). If more events are produced though, configuration traffic overhead included in the total traffic overhead of any WS based approach (3767 bytes for WS(C)) will not be included again since this happens only once for each event job. As such SNMP's traffic overhead becomes worse than the WS event reporting by task delegation approach by 1467 (3767-2300) bytes for each new event (Fig 9).

6 Conclusions

In this paper we have shown that facilities such as task delegation for WS event reporting can lead to significant gains in latency and traffic overhead since many of the tasks that must be performed upon receipt of an event report can be performed locally at the agent. We have also shown that such facilities have major performance gains for WS against SNMP. Offering such facilities is plausible today since the technical characteristics of devices used for management have increased.

Our work though on event reporting can also be improved by refining our policy-like grammar to meet closely the requirements of policy management. Currently our event reporting system is manually configured to perform a set of tasks dynamically at run-time. The essence of policy-based management for event reporting though would be to design an event reporting system that will autonomously deduce the actions to perform. This is in our future goals. Finally it is in our goals to apply our event reporting system to other fields and more resource constrained environments.

Nevertheless our event reporting system has great application potential. Through a realistic scenario we have demonstrated that sophisticated facilities for WS event reporting can lead to significant gains. Distributing task load for event reporting is extremely important, resulting in more distributed scalable, self adaptive systems.

References

[1] D. Box et al "Web Services addressing" http://www.w3.org/Submission/ws-addressing/

[2] S. Vinoski, IONA Technologies "Web Services Notifications" IEEE computer society, IEEE Internet Computing, Volume 8, Issue 2, March-April 2004 pp. 86 - 90.

[3] S. Vinoski IONA Technologies "More Web Services Notifications" IEEE computer society, IEEE Internet Computing, Volume 8, Issue 3, May-Jun 2004 pp. 90 - 93

[4] N. Catania et al, "WS-EVENTS2.0" HP, http://devresource.hp.com/drc/specifications/wsmf

[5] D. Box et al, "WS-Eventing," http://www.w3.org/Submission/WS-Eventing/

[6] S. Graham et al , "Web Services Base Notification" http://www-128.ibm.com/

[7] J.P Flattin "Web-Based Management of IP networks and Systems" ©Wiley series 2003.

[8] A. Pras et al "Comparing the Performance of SNMP and WS-Based Management," *IEEE* eTNSM, Vol 1 Number 2 December 2004

9] G. Pavlou, P. Flegkas, and S. Gouveris, "On Management Technologies and the Potential of Web Services," *IEEE Communications Magazine,* Vol. 42, no. 7, pp. 58-66 July 2004.

[10] S. Blake *et al.*, "An Architecture for Differentiated Services," RFC 2475, Dec. 1998

[11]D. Goderis *et al.*, "SLS Semantics and Parameters," draft-tequila-sls-02.txt, Aug. 2002.

[12]H. Asgari, R. Egan, P. Trimintzios, G. Pavlou "*Scalable monitoring support for resource management and service assurance",* IEEE Network, Vol 18, Issue 6. 2004 pp.6 – 18.

[13]ENTHRONE, http://www.enthrone.org/. 2nd phase 1/9/2006

[14]C. Srinivasan, et al, "MPLS Label Switching Router MIB" RFC 3813, June 2004.

[15]T. Nadeau, et al , "MPLS Forwarding Equivalence Class To Next Hop Label Forwarding Entry MIB*"* RFC 3814, June 2004.

[16]A. Chourmouziadis, G. Pavlou, "Efficient Information Retrieval in Network Management Using Web Services" DSOM 2006 Proceedings, October 23-25, 2006, Dublin, Ireland.

[17]M. Sloman, "Policy Driven management for Distributed Systems," JNSM Vol.2 No4 1994.

[18]W. Lima et al "Evaluating the performance of Web Services and SNMP notifications" NOMS 2006. 10th IEEE/IFIP pp. 546- 556