Comparing the Performance of SNMP and Web Services-Based Management

Aiko Pras, Thomas Drevers, Remco van de Meent, Dick Quartel

Abstract— **This paper compares the performance of Web services based network monitoring to traditional, SNMP based, monitoring. The study focuses on the `ifTable`, and investigates performance as function of the number of retrieved objects. The following aspects are examined: bandwidth usage, CPU time, memory consumption and round trip delay. For our study several prototypes of Web services based agents were implemented; these prototypes can retrieve single `ifTable` elements, `ifTable` rows, `ifTable` columns or the entire `ifTable`. This paper presents a generic formula to calculate SNMP's bandwidth requirements; the bandwidth consumption of our prototypes was compared to that formula. The CPU time, memory consumption and round trip delay of our prototypes was compared to Net-SNMP, as well as several other SNMP agents. Our measurements show that SNMP is more efficient in cases where only a single object is retrieved; for larger number of objects Web services may be more efficient. Our study also shows that, if performance is the issue, the choice between BER (SNMP) or XML (Web services) encoding is generally not the determining factor; other choices can have stronger impact on performance.**

Index Terms— **SNMP, Web services, performance, bandwidth usage, CPU time, memory consumption, round trip delay, BER, XML, compression, `ifTable`.**

I. Introduction

FIFTEEN years ago SNMP was designed as the protocol to manage the Internet. Through the years, new functions were added and nowadays SNMPv3, which includes a rich array of security functions, has reached the status of full Internet standard. Despite this status, there are still concerns about SNMP's deployment. The IAB, for example, discussed these concerns at a special Network Management Workshop, which was organized in summer 2002. One of the conclusions at that workshop was that it becomes time to investigate alternative network management technologies, in particular those that take advantage of XML technology [1], [2].

Web services is a specific form of XML technology. The interesting fact about Web services, which build upon W3C standards like SOAP [3] and WSDL [4], is that it is a generic technology, supported by many vendors and available on many platforms. There are many tools that ease the implementation of Web services based applications, integration with existing software is relatively simple, and many researchers are already familiar with this technology. Organizations like, for example, the DMTF and OASIS have already several years of experience in applying XML and Web services technologies in the area of applications and systems management. Thus, wouldn't Web services be an interesting technology for network management?

At the 11th meeting of the IRTF Network Management Research Group (NMRG), which was organized three months after the IAB workshop, a possible paradigm shift towards XML based network management was discussed [5]. The attendees agreed that such a shift would have a dramatic impact on the possibilities to model management information: the Structure of Management Information (SMI) [6], as used with SNMP, only supports simple variables and tables; whereas Web services and XML support the creation of far more sophisticated constructs. A move towards Web services based management would also have consequences for security and the possibility to create transactions. The attendees did not agree, however, on the consequences for performance; several attendees expressed their concern that the anticipated high demands of Web services on network and agent resources would hinder, or even prohibit, the application of this technology in the field of network management. Unfortunately, at that meeting, the issue could not be settled since no figures were known in which the performance of Web services was compared to that of SNMP.

The discussions at that NMRG meeting inspired researchers from the University of Twente to further investigate the possible performance differences between SNMP and Web services technology. To conduct tests, several Web services based prototypes were built, and the performance of these prototypes was compared to various SNMP agents. Amongst the investigated SNMP agents are open source packages, as well as commercial versions. This paper presents the outcome of our comparison.

The results of this paper show that there is a significant difference in the bandwidth requirements of SNMP and Web services. This difference may be particularly interesting whenever large amounts of management data must be exchanged. A good example is the case where software of cable modems must be updated; at this moment this is often done via SNMP [1]. Another example is the retrieval of interface specific data from access switches and DSLAMs. Such devices connect hundreds of users, and managers may need to retrieve for all these users counters from the interface table (`ifTable`) [7], for instance to determine if performance is still acceptable or to perform accounting. Especially if management is performed over out-of-band 64 Kbit links, the retrieval of such counters may put a heavy burden on the management link.

Since one of the goals of our study was to analyze real management scenarios (instead of theoretical scenarios that

The authors are with the University of Twente, P.O. Box 217, 7500 AE Enschede, the Netherlands. E-mail: {pras, meent, quartel}@cs.utwente.nl. Part of this research has been sponsored by the Telematica Instituut (TI) in the Netherlands.

may only occur in lab environments), our study focuses on the retrieval of `ifTable` data. In particular we investigate performance as function of the number of retrieved objects. Our interest is not only in finding the performance differences between message processing in SNMP and Web services, but also in discovering how much SNMP or Web services processing contributes to the total performance of the agent. In other words, is performance primarily determined by protocol processing, or have other factors stronger impact? For this reason our agents not only decode and encode messages, but also fetch actual data from within the agent system.

The following performance metrics were investigated: bandwidth usage, system resource usage (CPU and memory consumption), and round trip delay. The first one, bandwidth usage, primarily depends upon the protocol definition and should be the same for every implementation. Note that, in practice, this may not always be true, since implementations can choose slightly different options for encoding. Still the variance in bandwidth usage between different implementations should be small. The conclusions derived from our bandwidth usage measurements should therefore be generally applicable.

This is not necessarily true for our CPU, memory and round trip delay measurements. The results of such measurements depend very much on the hardware platform and the quality of the implementation; results may thus be completely different for alternative cases. Therefore, the conclusions derived in this paper for CPU and memory usage, as well as round trip delay, should be seen as indications of possible performance, and not as indisputable facts.

Since XML encoding is known to be verbose, we decided to also investigate the effect of compression. To keep the comparison fair, we not only investigated XML compression, but also SNMP compression.

The remainder of this paper is structured as follows. Section II gives an overview of related work. Section III gives details of our measurement set-up and the prototypes we've built. Section IV presents the bandwidth requirements for SNMP and Web services based network monitoring. Section V discusses CPU time and memory consumption. Section VI discusses round trip delay. Conclusions are given in Section VII.

II. Related work

In literature several papers can be found that investigate the performance of SNMP [8], compare the performance of SOAP to other middleware technologies [9], analyze the performance aspects of SOAP processing [10], and compare the performance of different SOAP and Web services toolkits [11]. This section discusses a number of publications that focus specifically on the performance of Web services for network management.

Mi-Jung Choi and James Hong have published several papers in which they discussed the design of XML-SNMP gateways. As part of their research, also the performance differences between XML and SNMP based management have been investigated [12], [13]. To determine bandwidth, they've measured the XML traffic at one side of the gateway, as well as the corresponding SNMP traffic at the other side. In their test set-up separate SNMP messages were generated for every object that had to be retrieved; the possibility to request multiple objects via a single SNMP message was not part of their test scenario. The authors also investigated delay and resource (CPU and memory) usage of the gateway. They concluded that, for their specific test set-up, XML performed better than SNMP.

Whereas the previous authors compared the performance of SNMP to XML, Ricardo Neisse and Lisandro Granville focused on SNMP and Web services [14]. Just like the previous authors, they implemented a gateway and measured traffic at both sides of that gateway. Two gateway translation schemes were distinguished: protocol level and object level. In the first scheme a direct mapping exists between every single SNMP and Web services message. In the second scheme a single, high level, Web services operation (like Get-IfTable) maps on multiple SNMP messages. The authors also investigated the performance impact of using Secure HTTP and zlib compression [15]. NuSOAP was used as web services toolkit. For protocol level translation they concluded that Web services always require substantial more bandwidth than SNMP. For object level translation they found that Web services perform better than SNMP if larger numbers of objects are retrieved.

Another interesting study on the performance of Web services and SNMP is performed by George Pavlou *et al.* [16], [17]. In fact their study is much broader than performance, and also includes CORBA based approaches. They take as example the retrieval of TCP MIB variables, and measure bandwidth and round trip delay. Their approach is comparable to the one described in this paper, in the sense that no gateways are used. Instead of a gateway, they implemented a Web services agent, using the WASP toolkit. The performance of this agent was compared to a Net-SNMP agent. As opposed to this paper, they did not investigate other SNMP agents nor the effect of fetching the actual management data from within the system. They concluded that Web services is a promising technology but has more overhead than SNMP.

III. Measurement set-up & prototypes

Within this study many of the measurements were performed on a Pentium 800 Mhz PC, running Debian GNU/Linux (kernel v2.4.22). The PC was connected via a 100 Mbit Ethernet card. In cases where it was necessary to have additional `ifTable` rows, tunnels were created to other systems. To get fair compression figures, we made sure that tunnel related objects would not all contain zeros.

A. Web services implementation

For our study four different Web services prototypes were built: one for retrieving single `ifTable` objects, one for retrieving `ifTable` rows, one for retrieving `ifTable` columns and one for retrieving the entire `ifTable` [18]. With these prototypes we were able to investigate the impact that different levels of granularity have on performance. The results presented in this paper have been obtained using the best performing prototype, in general this was the first (individual objects) or the last (complete table) prototype.

For the retrieval of interface specific data from within the system we wanted to use the same code in our SNMP and

Web services prototypes. In this way, differences between the measurements would be caused by differences in SNMP and Web services handling, and not by other, for this study irrelevant differences. For this reason we decided to base our Web services prototype on an existing open source SNMP software package, and to replace all SNMP code with Web services specific code. Since Net-SNMP is by far the most popular SNMP open source package, we used that package as starting point.

Since Net-SNMP is written in C, our Web services software had to support C too. After comparing several open source packages, our decision was to use gSOAP (v2.3.8) [19], [20] The code generated by gSOAP is quite efficient, since it generates a dedicated skeleton and does not use generic XML parsers, such as DOM and SAX. For compression zlib was used [15].

B. SNMP implementations

To measure bandwidth and delay, the following SNMP agents were used: 3Com (SuperStack II), Cisco (AGS+, 3750, 6500, 6502, 7200), HP (2626, 4000), IBM (8371), Nortel (Baystack 450, Passport 8610), UCD/Net-SNMP (4.2.3, 5.0.1, 5.0.9, 5.1 / Debian, Windows XP), Microsoft Windows XP agent, NuDesign agent, SNMP Research CIA agent, Cabletron Systems (ssr2000), and Xircom (GemTek).

Since the SNMP standards do not define compression and therefore none of these agents supports compression, an existing agent had to be modified to measure the effect of SNMP compression. In the past several SNMP compression mechanisms have been proposed. The oldest comes from the IRTF-NMRG, and is called ObjectID Delta Compression (ODC) [5]. An alternative comes from the IETF Evolution Of SNMP (EOS) working group [21], and is called ObjectID Prefix Compression (OPC). Code exists for the OPC proposal, and Net-SNMP v5.0.1 can be patched to include this code. Our SNMP compression measurements therefore used this software.

IV. Bandwidth usage

This section discusses and compares the bandwidth requirements of SNMP and Web services based network monitoring. Since SNMP is standardized, it is possible to analytically determine upper and lower bounds for the required bandwidth as function of the number of retrieved MIB objects. The section starts with deriving formulas that describe this function; since these formulas are generally applicable, they should also be useful in other SNMP related studies. Subsequently this section discusses how the formulas have been validated via a large number of measurements on the various SNMP devices at our disposal. The bandwidth requirements of Web services based network monitoring is discussed next. Since no standards for Web services based monitoring exist, the bandwidth usage of our prototypes had to be measured. The section concludes with a discussion of the effects of compression, and the bandwidth requirements at IP level.

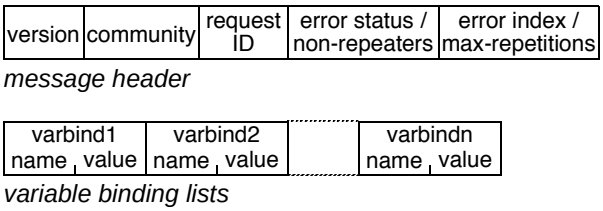

Fig. 1. SNMP message structure (v1/v2c)

A. SNMP messages and encoding

The structure of an SNMP message (v1/v2c) is shown in Figure 1 [22]–[24]. Each message consists of two parts: a header and a variable binding (varbind) list. The header contains five fields. The varbind list contains n varbinds, each carrying two fields: *object name* (OID) and *object value*.

The message and its various elements are defined in terms of ASN.1 constructs [25]; to transmit a message over the wire, Basic Encoding Rules (BER) are used [26]. Each BER encoded element consists of three parts: an ASN.1 *type* part, a *length* part and a *value* part. For SNMP, common ASN.1 *types* are `INTEGER`, `OCTET STRING`, `OBJECT IDENTIFIER` and `SEQUENCE (OF)`. In most cases, the BER encoding of an ASN.1 *type* takes, just as the *length* part, a single octet. The number of octets needed for the *value* part varies:

- `INTEGER` values require between one and five octets. For example, the value 127 can be coded into a single octet, while the value 232 requires five octets.
- an `OCTET STRING` requires the same number of octets as the length of the string.
- an `OBJECT IDENTIFIER`, in general, requires the same number of octets as its length, minus one (since the first two nodes are encoded into a single octet). For example, the length of OID 1.3.6.1.2.1.1.1.0 (= `sysDescr`) is 9; the encoded ASN.1 value part therefore needs 8 octets. The length of OID 1.3.6.1.2.1.2.2.1.10.1 (= `ifInOctets` of the first interface) is 11, and can be encoded in 10 octets. Note that, in cases where a node number within the OID is higher than 127, additional encoding octets will be needed.
- `SEQUENCE (OF)` is a constructor for other types (just like struct in the language C) and needs, besides its ASN.1 *type* and *length* parts, no additional octets.

The BER encoding of an SNMP message header is shown in Figure 2. The figure shows that four of the five header fields are of ASN.1 *type* `INTEGER` (version, request ID, error status / non-repeaters and error index / max-repetitions). The community field is of ASN.1 *type* `OCTET STRING`. Since the entire SNMP message is also defined as an ASN.1 `SEQUENCE`, two additional octets are needed to code the start of the message. The PDU type, which is implicitly coded, requires another two octets.

B. SNMP message size

From the previous explanation we can conclude that the length of a BER encoded SNMP header can be expressed as:

$$L_{\text{header}} = 4 + 4 \cdot L_{\texttt{INTEGER}} + L_{\texttt{OCTET STRING}} \qquad (1)$$

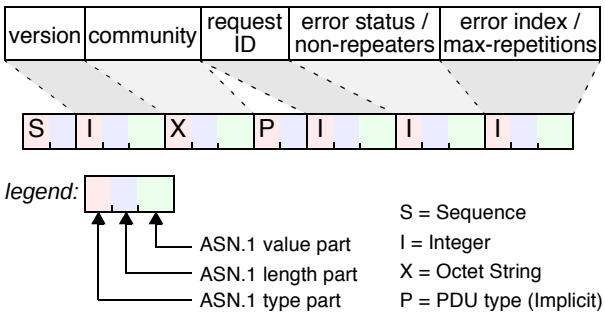

Fig. 2. SNMP header - BER encoding

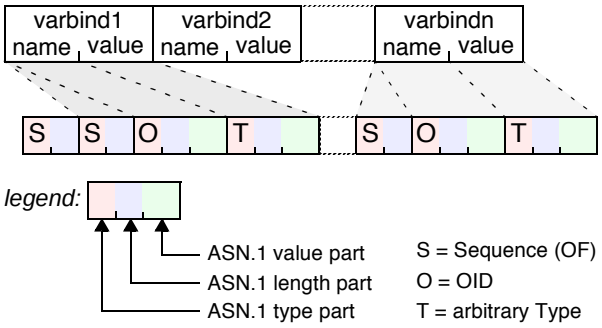

Fig. 3. SNMP variable binding list - BER encoding

If we assume that the community string is "public", the BER encoded SNMP header can take anything between 24 and 40 octets. From various measurements we've performed, we found as reasonable estimate:

$$L_{\text{header}} \approx 25 \text{ octets} \tag{2}$$

The BER encoding of the SNMP varbind list is shown in Figure 3. The encoding starts with two octets, to indicate that the varbind list is of ASN.1 type `SEQUENCE OF`. Each varbind also starts with two octets, to indicate that each varbind is of ASN.1 type `SEQUENCE`.

The length of a BER encoded SNMP varbind list can now be expressed as:

$$L_{\text{varbindlist}} = 2 + n \cdot (2 + L_{\text{name}} + L_{\text{value}}) \tag{3}$$

As discussed above, the number of octets needed for a BER encoded object identifier (OID) is:

$$L_{\text{name}} \geq 2 + OID_{\text{length}} - 1 \tag{4}$$

To get an idea of realistic values for OID_{length}, we've performed MIB walks over all the SNMP agents at our disposal (see Section III). Figure 4 shows, for each agent, the resulting OID_{length} distribution.

To BER encode an object value, two octets are needed for the ASN.1 *type* and *length* fields, plus $S_{\text{ObjectValue}}$ octets for the ASN.1 *value* part. Therefore:

$$L_{\text{value}} = 2 + S_{\text{ObjectValue}} \tag{5}$$

To get an idea of possible values for $S_{\text{ObjectValue}}$, we again performed measurements on all available agents. Also in this case a MIB walk was performed; every `Response` PDU was analyzed to determine the distribution of $S_{\text{ObjectValue}}$ (see Figure 5). We found that more than 90% of the object values

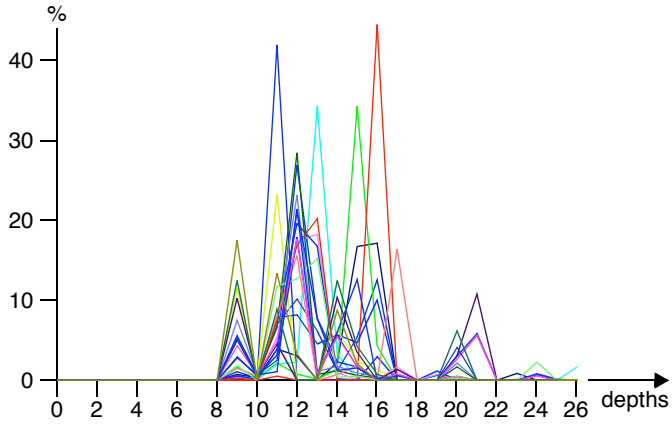

Fig. 4. SNMP OID length distribution (OID_{length})

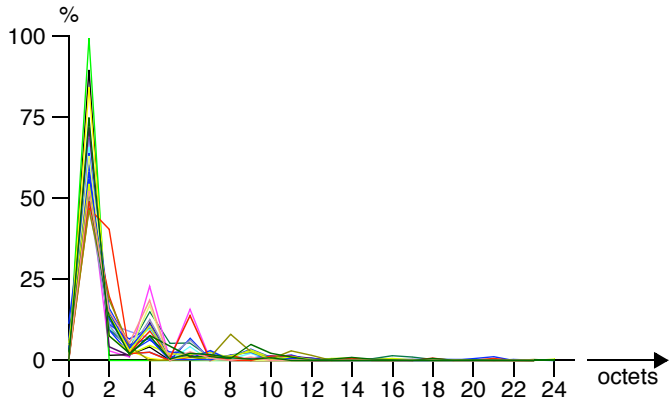

Fig. 5. SNMP object value size distribution ($S_{\text{ObjectValue}}$)

within our MIBs required for the BER encoding of the ASN.1 *value* part between 1 and 6 octets.

The number of octets needed to BER encode a complete SNMP message is:

$$L_{\text{SNMP message}} = L_{\text{header}} + L_{\text{varbindlist}} \tag{6}$$

Using (2) and (3):

$$L_{\text{SNMP message}} \approx 27 + n \cdot (2 + L_{\text{name}} + L_{\text{value}}) \tag{7}$$

Using (4) and (5), we find that generally:

$$L_{\text{SNMP message}} \approx 27 + n \cdot (5 + OID_{\text{length}} + S_{\text{ObjectValue}}) \tag{8}$$

C. Data retrieval with SNMP

For each data retrieval operation, two SNMP messages are needed: a request and a response. The number of octets required for the complete operation is therefore:

$$L_{\text{operation}} = L_{\text{request}} + L_{\text{response}} \tag{9}$$

In every requests, the BER encoding of the object value requires only two octets (ASN.1 *type* = NULL, *length* = 0):

$$S_{\text{ObjectValue, request}} = 0 \tag{10}$$

Note that $S_{\text{ObjectValue}}$ for `Response` PDUs was already given in Figure 5. Three operations exist to retrieve data: `Get`, `GetNext` and `GetBulk`. For each `Get` or `GetNext`

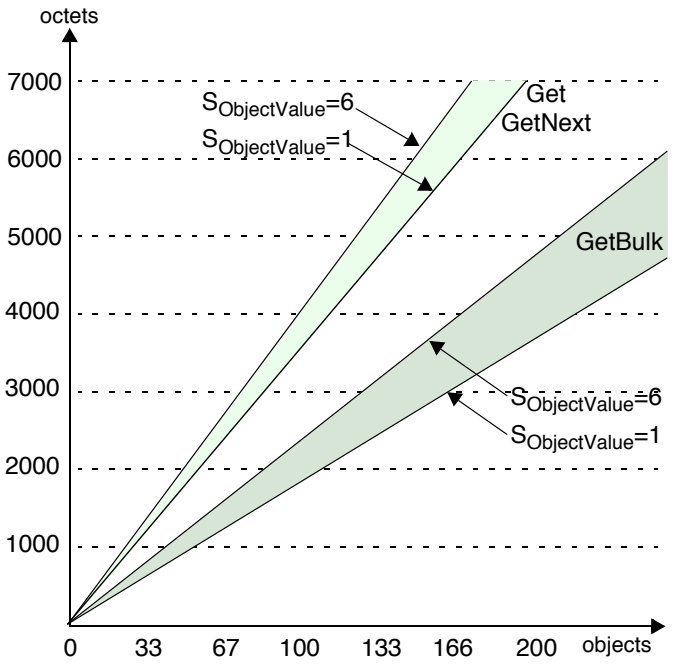

Fig. 6. Theoretical SNMP bandwidth consumption

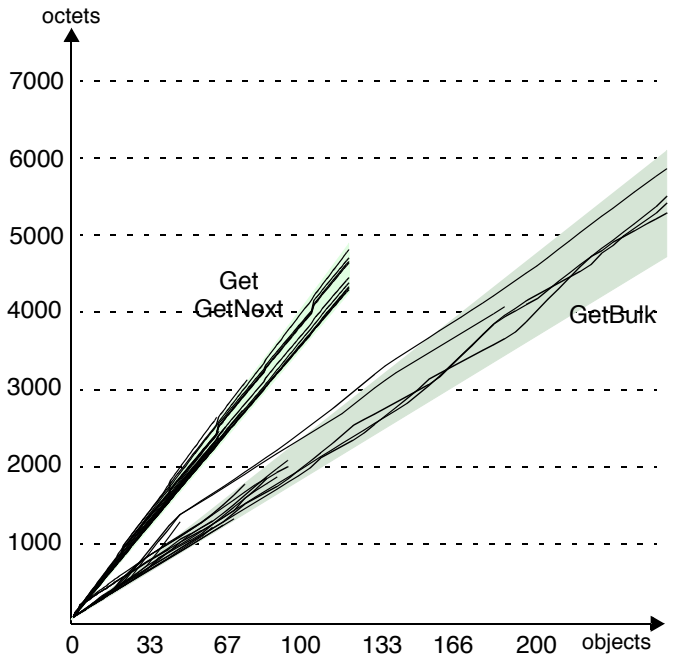

Fig. 7. Measured SNMP bandwidth consumption

operation, the number of varbinds in the request must be identical to the number of varbinds in the response. Therefore:

$$\begin{aligned} L_{\text{request}} &\approx 27 + n \cdot (5 + \mathit{OID}_{\text{length}}) \\ L_{\text{response}} &\approx 27 + n \cdot (5 + \mathit{OID}_{\text{length}} + S_{\text{ObjectValue}}) \\ L_{\texttt{GET}} &\approx 54 + n \cdot (10 + 2 \cdot \mathit{OID}_{\text{length}} + S_{\text{ObjectValue}}) \end{aligned} \tag{11}$$

For the `GetBulk` operation the number of varbinds in the request may be as low as one. Therefore:

$$\begin{aligned} L_{\text{request}} &\approx 27 + 1 \cdot (5 + \mathit{OID}_{\text{length}}) \\ L_{\text{response}} &\approx 27 + n \cdot (5 + \mathit{OID}_{\text{length}} + S_{\text{ObjectValue}}) \\ L_{\texttt{BULK}} &\approx 54 + 5n + n \cdot S_{\text{ObjectValue}} + (n+1) \cdot \mathit{OID}_{\text{length}} \end{aligned} \tag{12}$$

The formulas (11) and (12) allow us to determine upper and lower bounds for the number of octets needed to retrieve SNMP objects. If we assume that only objects from the `ifTable` will be downloaded, $\mathit{OID}_{\text{length}}$ will be equal to 11 and we can rewrite (11) and (12) as:

$$L_{\texttt{GET}} \approx 54 + n \cdot (32 + S_{\text{ObjectValue}}) \tag{13}$$

$$L_{\texttt{BULK}} \approx 70 + 16n + n \cdot S_{\text{ObjectValue}} \tag{14}$$

Using these formulas, it is now possible to graphically show SNMP's bandwidth consumption as function of the number of retrieved objects (Figure 6).

D. Verification of SNMP message size

To verify these formulas, we've retrieved from every available agent (see Section III) hundreds of MIB objects. Retrieval started from the first object of the `ifTable`, and often included more than 10 `ifTable` rows. To capture traffic, `tcpdump` was used [27]. The results are shown in Figure 7. It is interesting to note that a small number of measurement points fall outside the expected areas; the reason is that some agents place relatively long strings ($S_{\text{ObjectValue}} > 20$ octets) within their `ifDescription` objects, which are amongst the first objects to be retrieved. After more objects are retrieved, all lines fit within the expected area, however.

E. SNMPv3

The analysis thus far concentrated on the versions 1 and 2c of SNMP. For version 3, which is full standard, a similar analysis is possible. For sake of brevity, however, this paper only presents the conclusion of that analysis. With respect to bandwidth, the main difference is that the SNMPv3 header includes a number of additional fields, which increase its size to something between 42 and 143 octets (instead of 25, as needed for versions 1 and 2c). In addition, SNMPv3 may occasionally need to exchange extra messages to synchronize time (`EngineBoots` and `EngineTime`). Although the increase may be substantial in cases where only a single object is retrieved, it is almost negligible if a large number of objects are retrieved.

F. Web services

Since there are no standards for Web services based network monitoring, it is not possible to analytically derive formulas that give lower and upper bounds for the bandwidth needed to retrieve, for example, `ifTable` data. In fact, the required bandwidth depends on the specific WSDL definition, which may be different from case to case. This paper therefore only discusses the bandwidth requirements of our prototypes. The discussion focuses on the prototype that fetches the entire `ifTable` within a single interaction; similar measurements were also performed with the other prototypes.

The interaction starts with a SOAP request, which is shown in Figure 8. The first part of the request identifies several name spaces; the body includes the `GetIfTable` call, as well as

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/
                    soap/envelope/"
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/
                    soap/encoding/"
    xmlns:xsi="http://www.w3.org/2001/
               XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:utMon="urn:UTMON">
<SOAP-ENV:Body SOAP-ENV:encodingStyle=
    "http://schemas.xmlsoap.org/soap/encoding/"
     id="_0">
    <utMon:GetIfTable>
        <community xsi:type="xsd:string">
           public
        </commuity>
    </utMon:GetIfTable>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Fig. 8. SOAP request message

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/
                    soap/envelope/"
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/
                    soap/encoding/"
    xmlns:xsi="http://www.w3.org/2001/
               XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:utMon="urn:UTMON">
<SOAP-ENV:Body SOAP-ENV:encodingStyle=
    "http://schemas.xmlsoap.org/soap/encoding/"
     id="_0">
       ifEntry
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Fig. 9. SOAP response message - generic part

```
<ifEntry>
  <ifIndex xsi:type="xsd:unsignedInt">
     2</ifIndex>
  <ifDescr xsi:type="xsd:string">
     eth0</ifDescr>
  <ifType xsi:type="xsd:unsignedInt">
     6</ifType>
  <ifMtu xsi:type="xsd:unsignedInt">
     1500</ifMtu>
  <ifSpeed xsi:type="xsd:unsignedInt">
     10000000</ifSpeed>
  <ifPhysAddress xsi:type="xsd:string">
     0|40|63|C9|71|18</ifPhysAddress>
  <ifAdminStatus xsi:type="xsd:unsignedInt">
     1</ifAdminStatus>
  <ifOperStatus xsi:type="xsd:unsignedInt">
     1</ifOperStatus>
  <ifInOctets xsi:type="xsd:unsignedInt">
     354210076</ifInOctets>
  <ifInUcastPkts xsi:type="xsd:unsignedInt">
     1399059</ifInUcastPkts>
  <ifInDiscards xsi:type="xsd:unsignedInt">
     0</ifInDiscards>
  <ifInErrors xsi:type="xsd:unsignedInt">
     0</ifInErrors>
  <ifOutOctets xsi:type="xsd:unsignedInt">
     434349174</ifOutOctets>
  <ifOutUcastPkts xsi:type="xsd:unsignedInt">
     1508987</ifOutUcastPkts>
  <ifOutDiscards xsi:type="xsd:unsignedInt">
     0</ifOutDiscards>
  <ifOutErrors xsi:type="xsd:unsignedInt">
     1</ifOutErrors>
  <ifOutQLen xsi:type="xsd:unsignedInt">
     0</ifOutQLen>
  <ifSpecific xsi:type="xsd:string">
     0:0</ifSpecific>
</ifEntry>
```

Fig. 10. SOAP response message - `ifEntry`

the community string as parameter. The length of the request is approximately 500 octets.

The generic part of the SOAP response message is shown in Figure 9. Again the message starts with identifying several name spaces, after that data is followed for the first interface table entry (`ifEntry`). The length of the generic part is approximately 450 octets.

The interface specific part of the SOAP response is shown in Figure 10. In this case 18 objects have been retrieved; most objects are of type `unsignedInt`, except `ifDescr`, `ifPhysAddress` and `ifSpecific`, which are of type `string`. The size of this part is slightly over 1000 octets; many of these octets are used for identifying attributes (35%), like `ifIndex`, and types (15%). It is easy to see that, in principle, bandwidth usage can be lowered by choosing shorter attribute names. If the `ifTable` contains multiple rows (interfaces), the response messages contains multiple `ifEntry` parts; each part requiring approximately 1000 octets.

To compare the bandwidth requirements of our Web services prototypes to SNMP, several measurements were performed. The results are shown in Figure 11. It becomes clear that SNMP is far better than Web services, particularly in cases where only a few objects are retrieved. But, even in cases where a large number of objects are retrieved, SNMP remains a factor 2 (`Get`) to 4 (`GetBulk`) better than Web services.

G. Compression

Given the verbose nature of XML, we decided to additionally investigate the impact of (zlib) compression on Web services' bandwidth consumption. To make comparison fair, we also (OPC) compressed SNMP messages and measured their demands as well (see Section III for details). Our measurements show that, for Web services, compression reduces bandwidth consumption with a factor 2 in cases where only a single object is retrieved; if 50 objects are retrieved the gain is a factor 4, for 250 objects it is even a factor 8. Compared to normal SNMP, the bandwidth consumption of compressed SNMP reduces to approximately 75%. Because of the specifics of the compression algorithm, this percentage will not improve in cases where large numbers of objects are retrieved. Figure 12 shows the results: in cases where more than 70 objects are retrieved, compressed Web services performs better than SNMP.

H. Bandwidth usage at network level

The previous discussion focused on bandwidth usage at the application (SNMP / SOAP) layer. To determine SNMP's bandwidth usage at the IP layer, the IP header (20 octets) as well as the UDP header (8 octets) should be added for both

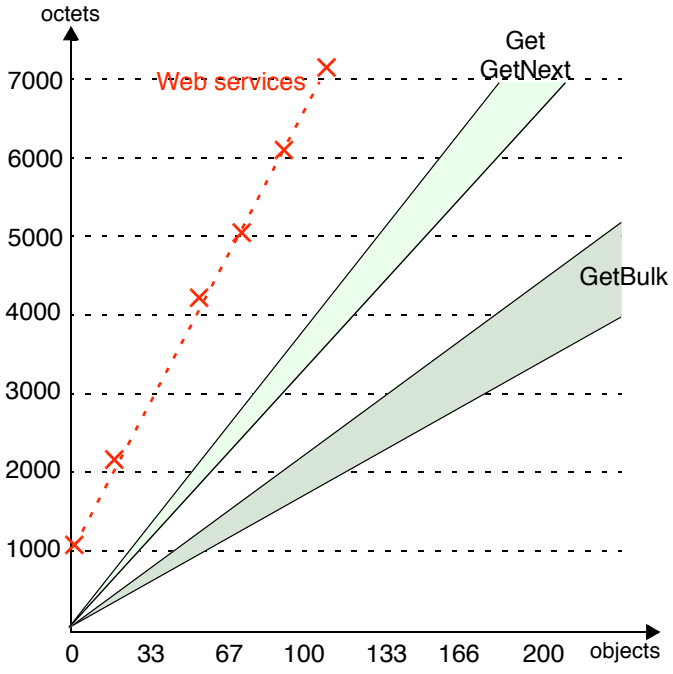

Fig. 11. Web services versus SNMP

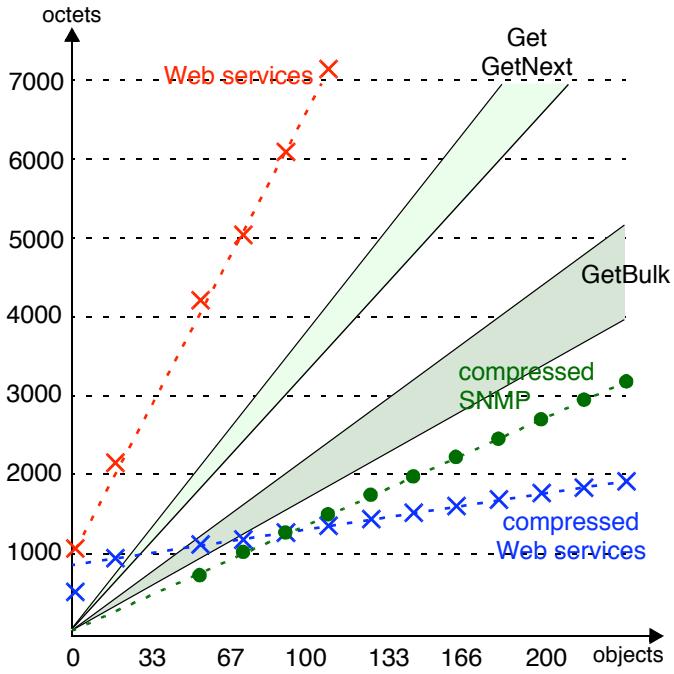

Fig. 12. Effect of compression

request and response message. A complete request-response interaction at the IP level therefore requires 56 more octets than at the SNMP level.

For Web services the calculation is more complex. To establish and release the TCP connection, 380 octets are needed. The SOAP data is preceded by HTTP, TCP and IP headers, and triggers the exchange of TCP acknowledgements. From the measurements performed at our prototypes we found that the SOAP data exchange at IP level requires between 273 (1 object) and 485 (250 objects) additional octets. In case of compression, these numbers vary between 613 and 715 octets. These numbers imply that the previous conclusions still hold, although the point where compressed Web services outperform SNMP, moves to somewhere between 125 and 150 objects.

V. System resource usage

To compare the CPU and memory requirements of our Web services prototype to that of SNMP, a Net-SNMP agent (v5.0.9) had to be modified to include measurement code. To determine delay, the `gettimeofday` function was used, which gives back the current time in microseconds. By subtracting the start-time of an operation (encoding, data retrieval etc.) from its end-time, the time needed to perform that operation can be determined. To measure memory usage, the `ps` utility and the `dmalloc` library were used.

A. CPU time

The first test was to determine the amount of CPU time needed for coding (decoding plus the subsequent encoding) BER and XML encoded messages. The results, as function of the number of objects within the message, is shown in Figure 13. It turned out that, for an SNMP message carrying a single object, the coding time is roughly 0.06 ms. A semantically equivalent, XML encoded message, requires 0.44 ms, which is 7 times more. Coding time increases if the number of objects within the message increases. Coding an SNMP message containing 216 objects requires 0.8 ms; coding a similar Web services messages requires 2.5 ms. We may therefore conclude that XML encoding requires 3 to 7 times more CPU time than BER encoding.

Since our bandwidth measurements showed that compressed Web services required considerable less bandwidth than uncompressed Web services, we were also interested in the effects of compression on CPU time. The results are also shown in Figure 13; it turns out that, compared to XML coding, compression is quite expensive (a factor 3 to 5). In cases where bandwidth is cheap and CPU time expensive, it might therefore be better to not compress Web services messages.

To determine how much coding contributes to the complete message handling time, also the remaining time to retrieve the actual data from within the system was measured. In case of the `ifTable`, but also in case of many other tables, such retrieval requires, amongst others, a system call. It turned out that retrieving data is relatively expensive; fetching the value of a single object usually takes between 1.2 and 2.0 ms. This time is considerably larger than the time needed for coding.

Figure 14 shows data retrieval time as function of the number of retrieved objects; to facilitate comparison, the BER and XML coding times are shown as well. The figure shows that data retrieval times for Net-SNMP increase quite fast; five objects already require more than 6 ms, for 54 objects 65 ms were needed and for 270 objects even 500 ms. These results can only be explained from the fact that Net-SNMP performs a new system call every time it finds a new `ifTable` object within the `Get(Bulk)` request. This could have been avoided, since the previous system call already delivered all information needed to fill the subsequent `ifTable` objects. Since

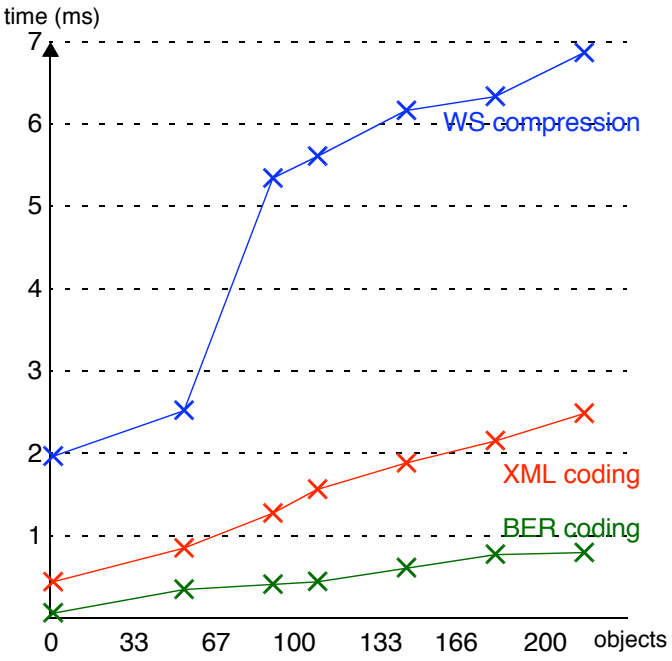

Fig. 13. CPU time for coding and compression

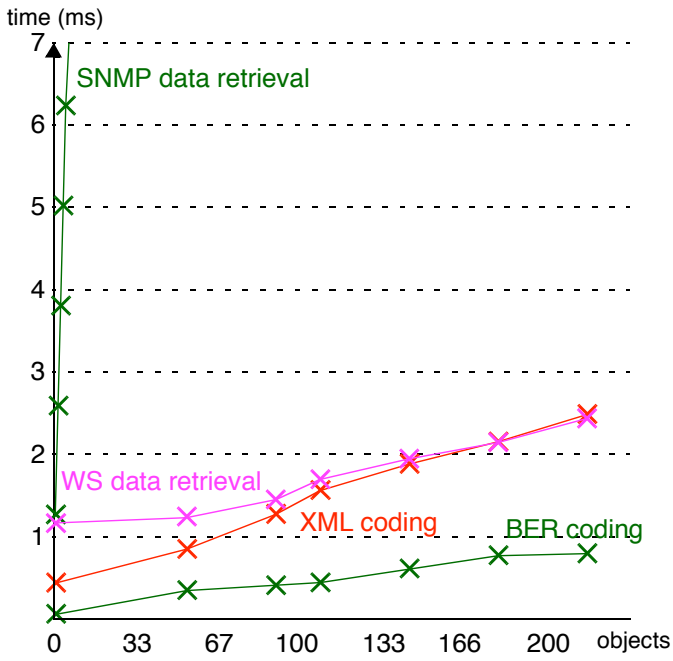

Fig. 14. CPU time for coding and data retrieval

data retrieval is far more expensive than BER encoding, the performance of Net-SNMP could dramatically be improved if caching of system calls gets implemented. It should be noted however, that on the same hardware, older versions of UCD/Net-SNMP (v4.3.2) provide far better data retrieval times; these times are roughly one third of those needed in newer versions, like v5.0.9.

Since caching was implemented in our Web services prototype, data retrieval will be far more efficient than Net-SNMP in case a single request includes multiple `ifTable` objects. For a few objects data retrieval takes more time than XML coding, but if more than 180 objects are requested, XML coding gets more expensive.

B. Memory usage

Memory is needed for holding program instructions as well as data. In case of data memory, a distinction can be made between static and dynamic memory. Static memory is allocated at the program start, and remains constant during the programs lifetime. Dynamic memory is allocated after a request is received, and released after the response is transmitted. If multiple requests arrive simultaneously, dynamic memory is allocated multiple times.

Table I shows the memory requirements of Net-SNMP and our Web services prototype. It turns out that Net-SNMP requires roughly 3 times more instruction memory than our Web services prototype. Also Net-SNMP requires 20 to 40 times more data memory, depending on the number of objects contained in the request.

The importance of these numbers should not be overestimated, however. Although we removed from Net-SNMP all MIB code not related to the `ifTable`, the functionality of Net-SNMP is still much richer than that of our Web services prototype. For example, Net-SNMP supports three different protocol versions, includes encryption, authentication

TABLE I
MEMORY REQUIREMENTS

		data	
	instructions	static	dynamic
SNMP	1972 kB	129 kB	70-160 kB
Web services	580 kB	470 B	4 kB

and access control, and is written in a platform independent way. Our prototype has limited functionality, and is based on gSOAP, which is generally known for its efficiency. Others, for example, implemented comparable prototypes, but used the WASP Web Services platform instead of gSOAP. It turns out that such alternative implementation can easily require 6 times more memory than our prototype, and 2 times more memory than Net-SNMP [16]. Also if we add compression to our Web services prototype, memory requirements for dynamic data increases with more than 300 kB for one operation.

VI. ROUND TRIP DELAY

One of the interesting results thus far, is that the time needed to retrieve management data from within the system, is higher than the time needed to (de-)encode messages. In addition, Net-SNMP does not cache previously fetched data, so data retrieval time increases linearly with the number of retrieved objects. The question is now whether this result is specific for Net-SNMP, or is also valid for other SNMP implementations. To determine this, measurements were performed on all available SNMP agents (see Section III). Since it is impossible to add to these agents code that measures BER encoding and data retrieval times, we decided to measure round trip delay instead. This delay can be measured external to the agent; in our case we used tcpdump [27] at the manager side to capture all traffic plus timing data.

TABLE II
ROUND TRIP DELAY (IN MS) AS FUNCTION OF THE NUMBER OF RETRIEVED OBJECTS

	1	22	66	270
WS	1.7	2.6	10.3	36.5
WS-Comp	3.3	4.3	5.6	11.8
SNMP-1	0.4	1.6	3.9	21.1
SNMP-2	0.4	1.9	5.0	
SNMP-3	0.5	1.6	4.2	
SNMP-4	0.5	1.7	4.4	
SNMP-5	0.5	1.8	4.8	
SNMP-6	0.7	2.2	5.7	
SNMP-7	0.8	1.8	2.9	
SNMP-8	0.9	1.6	3.9	
SNMP-9	0.9	6.6	18.5	
SNMP-10	1.1	1.8	3.4	58.5
SNMP-11	1.2	2.9	6.7	
SNMP-12	1.3	2.7	5.4	
SNMP-13	1.5	14.0	40.1	
SNMP-14	1.6	5.0	15.1	
SNMP-15	1.7	4.2	9.6	
SNMP-16	2.7	44.5	127.6	178.7
SNMP-17	2.7	47	140.4	251.7
SNMP-18	3.5	17.2		
SNMP-19	3.7	24.3	77.9	
SNMP-20	4.1	76.7	100.8	
SNMP-21	11.1	83.7	243.0	
SNMP-22	11.3	238.7	727.6	
SNMP-23	87.7	1822.2		

We focused our measurements on single request-response interactions. In the case of our Web services prototype, we ignored connection establishment and release times, since one might use an existing TCP connection for multiple interactions. Basically we measured the time between sending the first TCP PSH segment, and receiving the last TCP PSH segment. This time was measured for retrieving 1, 22, 66 and 270 objects, for normal as well as compressed Web services (see Table II). It is interesting to note that round trip delay for normal Web services increases faster than that of compressed Web services. The reason is that the amount of uncompressed data is such, that multiple TCP PSH segments must be used; because of flow control (TCP's delayed acknowledgements), the time between transmitting these different segments is, in our test environment, considerable.

Table II also shows the round trip delay for all SNMP agents that we could use (see Section III). Since our goal is to compare SNMP against Web services, and not to find the best performing SNMP agent, we decided to shield the identity of the individual agents and simply number them from 1 to 23. This should be sufficient for our purpose, and avoid possible discussions with legal departments of companies. Each measurement was repeated 10 times; the figure shows the lowest delay that was obtained. For most measurements we used a single `GetBulk` operation, with a max-repetition value of 1, 22, 66 or 270; we also checked that `GetBulk` would be the fastest operation. For agents that do not support `GetBulk`, we used a single Get request, asking for 1, 22 or 66 objects. Not all agents were able to deal with large sized messages; the size of response message carrying 270 objects is such that, on an Ethernet LAN, the message has to be fragmented by the IP protocol.

Since the hardware for the various agents varied, the delay times shown in Table II should be used with great care and only considered as an indication. Under slightly different conditions, delay times might be quite different. For example, the addition of a second Ethernet card may have significant impact on delay times. Also delay times will change if other objects are retrieved than `ifTable` objects. In addition, the first SNMP measurement often takes considerable more time than subsequent measurements.

Despite these remarks, the overall trend is clear. Just as with Net-SNMP, for most agents delay time heavily depends on the number of retrieved objects. It seems that several SNMP agents would benefit from some form of caching and, after 15 years of experience in SNMP agent implementation, there is still room for performance improvements. From a delay point of view, the choice between BER and XML encoding doesn't seem to be of great importance. Our Web services prototype performs reasonably well and, for multiple objects, even outperforms several commercial SNMP agents.

VII. CONCLUSIONS

This paper compared the performance of Web services based network monitoring to that of SNMP. In particular it investigated bandwidth usage, CPU time, memory requirements and round trip delay.

Bandwidth usage of SNMP depends on the specific request-response message pair that is used to retrieve data. From our bandwidth measurements it becomes clear that SNMP is far better in cases where just a single object is retrieved. Also in cases where a large number of objects is retrieved, SNMP remains a factor 2 (`Get`) to 4 (`GetBulk`) better than normal Web services. The conclusion changes, however, when compression is used. If a large number of objects are requested, compressed Web services demands less bandwidth than SNMP. If, for example, `ifTable` data from a DSLAM holding 500 ADSL interfaces is retrieved, compressed Web services will clearly be the winner.

CPU time that is needed for the coding of SNMP (BER) messages, is 3 to 7 times less than needed for the coding of Web services (XML) messages. If Web services messages get also compressed, differences grow further with an additional factor of 3 to 5. One of the surprises of this study, however, is that the CPU time required for coding and compression is still less than the time required to retrieve data from within the system. Since many agents do not seem to cache system calls, data retrieval times go up with the number of retrieved objects. If hundreds of objects are requested, the time needed to code and compress messages can be neglected, compared to the time spent on retrieving data. From a performance point of view, coding is not the issue, but data retrieval.

Memory usage of our Web services prototype is less than that of Net-SNMP. It should be noted, however, that Net-SNMP is much richer in functionality; comparison is therefore not fair and these results should be used with great care. For example, other studies, using different Web services toolkits, obtained opposite results.

Round trip delay of our Web services prototype turned out to be comparable to existing SNMP agents. The disappointing performance of some SNMP implementations can be explained from the fact that data retrieval from within the system is usually more expensive than encoding and decoding, and many SNMP agents do not seem to cache data.

The overall conclusion of our study can be summarized as follows:

- In case a single object is retrieved, SNMP is more efficient than Web services.
- In case many objects are retrieved, Web services may be more efficient than SNMP.
- Coding is less expensive than data retrieval. The choice between BER and XML encoding is not the main factor that determines performance.

From these conclusions it seems that, from a performance point of view, there is no convincing reason to refuse Web services for network monitoring.

Further work is needed before a final conclusion can be drawn whether Web services technologies would be useful for network management purposes. In particular it is important to understand the impact of adding encryption, authentication, authorization (access control) and transactions. As opposed to the SMI, Web services and XML also facilitate the creation of more advanced data structures. Since the introduction of improved data modeling capabilities will have far reaching consequences, further research is needed before standardization of WSDL descriptions for network management purposes should take place [28].

ACKNOWLEDGMENTS

This paper is influenced by discussions at the 11th, 14th and 15th meeting of the IRTF Network Management Research Group. We would like to thank the participants of these meetings for their contributions. We would also like to thank Wes Hardeker for discussions on Net-SNMP, Dave Shields for the ObjectID Prefix Compression (OPC) code, as well as Mark Borst and Jürgen Schönwälder for giving us access to SNMP devices.

REFERENCES

[1] J. Schönwälder, "Overview of the 2002 IAB Network Management Workshop," May 2003, IETF RFC 3535.
[2] J. Schönwälder, A. Pras, and J.P. Martin-Flatin, "On the future of Internet management technologies," *IEEE Communications Magazine*, vol. 41, no. 10, pp. 90–97, October 2003.
[3] "Homepage of the XML Protocol WG," `http://www.w3.org/2000/xp/Group/`.
[4] "Homepage of the Web Services Description WG," `http://www.w3.org/2002/ws/desc/`.
[5] "Homepage of the IRTF Network Management Research Group (NMRG)," `http://www.ibr.cs.tu-bs.de/projects/nmrg/`.
[6] K. McCloghrie, D. Perkins, and J. Schönwälder, "Structure of Management Information Version 2 (SMIv2)," April 1999, IETF STD 58, RFC 2578.
[7] K. McCloghrie and F. KastenHolz, "The Interfaces Group MIB," June 2000, IETF RFC 2863.
[8] C. Pattinson, "A Study of the Behaviour of the Simple Network Management Protocol," in *Proceedings of DSOM2001*, October 2001.
[9] D. Davis and M. Parashar, "Latency Performance of SOAP Implementations," in *Proceedings of IEEE Cluster Computing and the GRID 2002*, 2002, pp. 407–412.
[10] K. Chiu, M. Govindaraju, and R. Bramley, "Investigating the Limits of SOAP Performance for Scientific Computing," in *Proceedings of 11th. IEEE International Symposium on High Performance Distributed Computing*, 2002, pp. 246–254.
[11] A. Ng, S. Chen, and P. Greenfield, "An Evaluation of Contemporary Commercial SOAP Implementations," in *Proceedings of the 5th Australasian Workshop on Software and System Architectures (AWSA 2004)*, 2004, pp. 64–71.
[12] M. Choi, J. Hong, and H. Ju, "XML-Based Network Management for IP Networks," *ETRI Journal*, vol. 25, no. 6, pp. 445–463, December 2003.
[13] M. Choi and J. Jong, "Performance Evaluation of XML-based Network Management," Presentation at the 16th IRTF-NMRG meeting, 2004, `http://www.ibr.cs.tu-bs.de/projects/nmrg/meetings/2004/seoul/choi.pdf`.
[14] R. Neisse, R. L. Vianna, L. Z. Granville, M. J. B. Almeida, and L. M. R. Tarouco, "Implementation and Bandwidth Consumption Evaluation of SNMP to Web Services Gateways," in *Proceedings of NOMS2004*, 2004.
[15] "Zlib homepage," `http://www.zlib.org`.
[16] G. Pavlou, P. Flegkas, and S. Gouveris, "Performance Evaluation of Web Services as Management Technology," Presentation at the 15th IRTF-NMRG meeting, January 2004.
[17] G. Pavlou, P. Flegkas, S. Gouveris, and A. Liotta, "On Management Technologies and the Potential of Web Services," *IEEE Communications*, vol. 42, no. 7, pp. 58–67, July 2004.
[18] T. Drevers, R. van de Meent, and A. Pras, "Prototyping Web Services based Network Monitoring," in *Proceedings of 10th Open European Summer School School and IFIP WG 6.3 Workshop (EUNICE 2004)*, June 2004, pp. 135–142.
[19] R. v. Engelen, "gSOAP web services toolkit," `http://www.cs.fsu.edu/~engelen/soap.html`.
[20] R. v. Engelen and K. A. Gallivany, "The gSOAP Toolkit for Web Services and Peer-To-Peer Computing Networks," in *Proceedings of IEEE Cluster Computing and the GRID 2002*, 2002, pp. 128–135.
[21] "Homepage of the EOS WG," `http://www.ietf.org/html.charters/OLD/eos-charter.html`.
[22] J. Case *et al.*, "A Simple Network Management Protocol (SNMP)," May 1990, IETF RFC 1157.
[23] J. Case *et al.*, "Introduction to Community-based SNMPv2," January 1996, IETF RFC 1901.
[24] R. Presuhn *et al.*, "Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)," December 2002, IETF RFC 3416.
[25] International Organization for Standardization, "Information processing systems - Open Systems Interconnection, Specification of Abstract Syntax Notation One (ASN.1)," December 1987, International Standard 8824.
[26] International Organization for Standardization, "Information processing systems - Open Systems Interconnection, Specification of Basic Encoding Rules for Abstract Notation One (ASN.1)," December 1987, International Standard 8825.
[27] Lawrence Berkeley National Laboratory Network Research, "TCPDump: the Protocol Packet Capture and Dumper Program," 2003, `http://www.tcpdump.org/`.
[28] J. van Sloten, A. Pras, and M. van Sinderen, "On the standardisation of Web service management operations," in *Proceedings of 10th Open European Summer School School and IFIP WG 6.3 Workshop (EUNICE 2004)*, June 2004, pp. 143–150.

Aiko Pras (a.pras@utwente.nl) is associate professor at the University of Twente (UT), the Netherlands. From this university he received in 1995 a Ph.D. degree for the thesis: "network management architectures." His current research interests include network management technologies, Web

Services, network measurements and accounting. He is member of the IRTF NMRG, technical co-chair of the Ninth IFIP/IEEE International Symposium on Integrated Network Management (IM 2005), and TPC member of many international conferences in the area of management.

Thomas Drevers (thomas@drevers.nl) has recently completed his M.Sc. study at the UT. His M.Sc. thesis is called: "Performance of Web services-based network monitoring". He has presented the results of his work at the 14th meeting of the IRTF NMRG, and at the EUNICE 2004 Summer school.

Remco van de Meent (r.vandemeent@utwente.nl) received his M.Sc. in Computer Science from the University of Twente, the Netherlands, in 2001. He currently is working towards his Ph.D. at the department of Electrical Engineering, Mathematics and Computer Science at the University of Twente. His research interests include Internet traffic measurements and traffic modeling, network provisioning and network management.

Dick Quartel (d.a.c.quartel@utwente.nl) is an assistant professor at the University of Twente (UT), the Netherlands. From this university he received in 1998 a Ph.D. degree for the thesis: "Action relations - Basic design concepts for behaviour modelling and refinement". His current research interests include architectural modelling, service-oriented design, service-oriented computing technologies, and context-aware services.