Prototyping Web Services based Network Monitoring

Thomas Drevers, Remco van de Meent, Aiko Pras

Abstract— **Web services is one of the emerging approaches in network management. This paper describes the design and implementation of four Web services based network monitoring prototypes. Each prototype follows a specific approach to retrieve management data, ranging from retrieving a single management data object, to retrieving an entire table of such objects at once. We have focused on the interfaces table (ifTable), as described in the IF-MIB.**

This paper also provides some preliminary measurement results, showing the amount of bandwidth required by our prototypes, the effect of compression of the management data, and the effect of having more management data objects. It turns out that the bandwidth usage without compression is between 1 Kbyte to retrieve a single data object, up to 4.2 Kbyte to retrieve the entire ifTable. Compression, however, reduces these numbers significantly: up to 87% less bandwidth usage.

Keywords— **Web services, performance, network management.**

I. Introduction

SEVERAL new approaches to network management are currently emerging. These approaches often focus on the use of the XML language, examples of these approaches are JUNOscript [1] and the work of the IETF network configuration workgroup (netconf) [2].

A special form of XML technology is Web services [3], which are being developed by the World Wide Web Consortium (W3C). Web services seem to be a promising technology for network management [4].

Although several papers have been published on the use [5] and performance [6] of Web services based management, these papers usually focus on the use of a gateway between the Web services, and the Simple Network Management Protocol (SNMP) [7].

In our research we focus on the performance of Web services based network management without the use of gateways. Our intention is to measure the performance of the Web services with respect to bandwidth consumption, agent CPU time consumption, agent memory consumption and round trip time. To determine the impact of different data encodings on the performance, multiple Web services based prototypes will be created, each using a different data retrieval scheme.

All authors are with the Department of Electrical Engineering, Mathematics and Computer Science, University of Twente, PO Box 217, 7500 AE Enschede, the Netherlands. Email: {drevers,meent,pras}@cs.utwente.nl .

A. Contribution

In this paper we present the design and implementation of several Web services based agents for network management. These agents focus on network monitoring, because this is one of the most common tasks of management. The paper also includes some preliminary measurement results; a full comparison study will be described in a future paper.

B. Approach and Structure

The development of Web services based network monitoring agents can be divided into two phases: a design phase, in which the functionality of the agent is defined, and an implementation phase, in which prototypes are build.

The design phase, which is presented in Section II, starts with an analysis of the structure of SNMP's interfaces table (ifTable). We decided to focus on this table, since it contains important data with respect to the usage of the network links. In the traditional SNMP world, the ifTable is probably the most frequently used table for network monitoring purposes. The outcome of the design phase are several WSDL [11] descriptions; each specific for one particular approach to retrieve interface data.

The implementation phase, which is presented in Section III, starts with the selection of a Web services toolkit to handle the encoding and decoding of the Web services messages. This toolkit will be responsible for generation the communication (WSDL) part of the prototypes; additional software will be needed to retrieve data from the kernel. After the toolkit has been selected, the structure of the agent prototypes will be discussed. At the end of the implementation phase, certain additions should be made to facilitate measurements. This section therefore concludes with a discussion of the measurements we want to perform, and the additions needed for such measurements.

Section IV provides some preliminary measurement results. These results give an indication of the performance differences between two types of Web services based network monitoring agents. This section will not discuss all possible performance issues, but only the issue of bandwidth consumption. A full performance study is outside the scope of this paper, but will be the subject of another paper.

II. Design

In this section the design of the Web services based network monitoring agents is discussed. The structure of SNMP's ifTable is analyzed in order to deter-

mine suitable data retrieval schemes, which we will call granularities. For each of the granularities a Web service is designed and described using WSDL.

A. Structure of the ifTable

In this study we have focused on the ifTable, which is part of the Interfaces Group MIB (IF-MIB) [12]. The ifTable contains data objects related to the state of all network interfaces available in the system on which the agent runs.

The ifTable consists of a variable amount of rows, depending on the number of network interfaces in the agent system, and a fixed number of 22 columns. For each network interface in the agent system, both physical (Ethernet, token-ring, etc.) and virtual (loopback, tunnel, etc.), one row is assigned. Every column represents a specific piece of information on the interface, e.g., the index number, or the amount of incoming data octets. See Figure 1 for an excerpt from the entire ifTable.

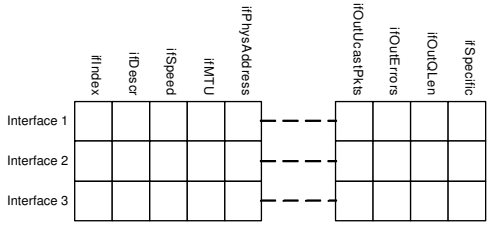

Fig. 1. Excerpt of the ifTable

In our research, we want to make make a full comparison between our Web services based network management agents, and the Net-SNMP agent [13]. Net-SNMP is a widely used open source SNMP implementation. The ifTable represented by Net-SNMP contains only 18 of the 22 columns described in the IF-MIB. To ensure a fair comparison between SNMP and the network monitoring Web services, our prototypes will use the same 18 columns.

B. Granularities

Using the ifTable as partly shown in Figure 1, we have determined 4 distinct data retrieval schemes, each resulting in a different network monitoring agent. All schemes (or granularities) will be able to retrieve any data object from the ifTable; the distinction is that the structure and amount of data objects that are retrieved varies.

The first granularity is to retrieve all data objects separately. This method is comparable to the method used by the SNMP protocol (through the Get operation), although SNMP allows multiple data objects to be contained in a single message. Our Web service that retrieves a single data object at a time is called *GetIfCell*.

The second granularity is to retrieve all data objects in the ifTable in a single operation. The Web service that retrieves all data objects in the ifTable at once is called *GetIfTable*.

The third granularity is to retrieve an entire row of data objects at a time, thus transmitting all available information on a single network interface. The Web service that retrieves a single row of data objects at a time is called *GetIfRow*.

The last granularity we have investigated, is to retrieve an entire column of data objects at a time, thus transmitting the same piece of information for all network interfaces. The Web service that handles one column at a time will be called *GetIfColumn*.

Figure 2 shows examples of the data objects retrieved by one operation of each of the granularities: *GetIfCell* (a), *GetIfColumn* (b), *GetIfRow* (c) and *GetIfTable* (d).

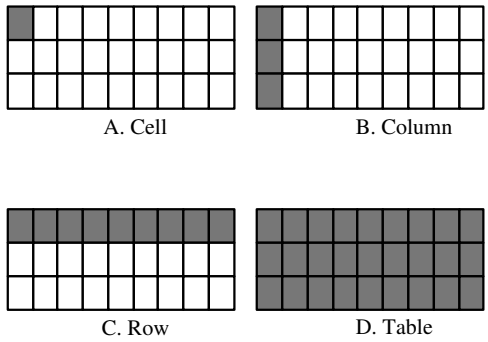

Fig. 2. The interfaces table

C. WSDL descriptions of the ifTable

The WSDL description of each of the four Web services consists of five main (container) elements:

- The <types> element.
- The <message> elements.
- The <portType> element.
- The <binding> element.
- The <service> element.

Each of the five main elements contains a set of elements used to describe a part of the interface for the Web services based network monitoring agents. The purpose and contents of each main element will be explained in the remainder of this section. The complete WSDL descriptions of our Web services can be found at `http://www.simpleweb.org/wsdl-if-mib/` .

C.1 <types> element

There are two types of type-elements used by XML: *simpleType* and *complexType* elements. simpleType elements contain a single value of predefined form. An example of a simpleType is an integer or a string. complexType elements group other elements together. An example of a complexType element is an *Address* element which contains *street*, *postal code*, and *city* (sub)elements.

In the WSDL description of our Web services network monitoring agents, complexType elements are used to group management data objects as defined by the ifTable together. The grouping corresponds to the granularity of the Web service. For instance, all columns of the ifTable can be grouped together,

yielding a complexType definition that can be used to denote an entire column.

For the *GetIfRow* and *GetIfCell* Web services, it may be necessary to retrieve a list of all available network interfaces in the agent system, in order to reference the desired interface in the actual operation. The following WSDL excerpt shows the <complexType> that is used to denote this list.

```
<types>
  <complexType name="interfaces">
   <sequence>
    <element name="index"/>
   </sequence>
  </complexType>
  ...
</types>
```

The *GetIfColumn* Web service uses complexTypes to denote lists of integers and strings in order to serialize the requested data. The definition of these complexTypes is similar to the definition above.

For the *GetIfTable* and *GetIfRow* Web services, we introduce the complexType "ifEntry", grouping columns together as shown below. Note that for the *GetIfRow* Web service, the <sequence> element is left out, as only one row is selected at a time.

```
<types>
  <complexType name="ifEntry">
   <sequence>
    <element name="ifIndex" type="xsd:unsignedInt"/>
    <element name="ifDescr" type="xsd:string"/>
    <element name="ifType" type="xsd:unsignedInt"/>
    <element name="ifMtu" type="xsd:unsignedInt"/>
    <element name="ifSpeed" type="xsd:unsignedInt"/>
    <element name="ifPhysAddress" type="xsd:string"/>
    <element name="ifAdminStatus" type="xsd:unsignedInt"/>
    <element name="ifOperStatus" type="xsd:unsignedInt"/>
    <element name="ifInOctets" type="xsd:unsignedInt"/>
    <element name="ifInUcastPkts" type="xsd:unsignedInt"/>
    <element name="ifInDiscards" type="xsd:unsignedInt"/>
    <element name="ifInErrors" type="xsd:unsignedInt"/>
    <element name="ifOutOctets" type="xsd:unsignedInt"/>
    <element name="ifOutUcastPkts" type="xsd:unsignedInt"/>
    <element name="ifOutDiscards" type="xsd:unsignedInt"/>
    <element name="ifOutErrors" type="xsd:unsignedInt"/>
    <element name="ifOutQLen" type="xsd:unsignedInt"/>
    <element name="ifSpecific" type="xsd:string"/>
   </sequence>
  </complexType>
  ...
</types>
```

C.2 <message> elements

The <message> container elements are used to the describe the information that is being exchanged between a Web service and a user. There are <message> elements for both request (input) messages as well as response (output) messages. For request messages, a <message> consists of zero or more <part> elements that correspond to the parameters of the Web service. For response messages, the <part> elements describe the response data. All <part> elements are associated with a type as defined in the <types> container element.

The *GetIfTable* Web service supports one operation: retrieving the complete table. The following listing shows the message elements the *GetIfTable* Web service uses for this operation. This listing shows a request message containing a "community" string, which is used for authentication similar to the first two versions of SNMP. The response message contains the "ifEntry" element, which was shown in the previous section, and an integer containing the amount of rows in the table.

```
<message name="GetIfTableRequest">
 <part name="community" type="xsd:string"/>
</message>

<message name="GetIfTableResponse">
 <part name="sizeTable" type="xsd:int"/>
 <part name="ifEntry" type="utMon:ifEntry"/>
</message>
```

The other Web services all support more than one operation. The *GetIfRow* Web service supports two operations, i.e., retrieving a specified row and retrieving a list of valid row index numbers (corresponding to the network interfaces in the agent system). The description of *GetIfColumn* Web service defines 18 operations: a distinct operation for retrieving data from each of the columns in the ifTable. Similarly, the *GetIfCell* Web service also has a separate operation for each column. The *GetIfCell* Web service also supports an operation to retrieve a list of valid row index numbers.

The following listing shows the messages for two of the operations used by the *GetIfCell* Web service. The request messages have an index element, which is used to reference the correct cell.

```
<message name="getIfIndexRequest">
 <part name="index" type="xsd:unsignedInt"/>
 <part name="community" type="xsd:string"/>
</message>

<message name="getIfIndexResponse">
 <part name="ifIndex" type="xsd:unsignedInt"/>
</message>

<message name="getIfDescrRequest">
 <part name="index" type="xsd:unsignedInt"/>
 <part name="community" type="xsd:string"/>
</message>

<message name="getIfDescrResponse">
 <part name="ifDescr" type="xsd:string"/>
</message>
```

C.3 <portType> element

A <portType> element describes the operations supported by a specific Web service: the request and response messages associated to each operation, and (optional) documentation of these operations.

As mentioned above, the *GetIfTable* Web service supports a single operation, which is shown in the following listing:

```
<portType name="GetIfTableServicePortType">
 <operation name="GetIfTable">
  <documentation>function utMon__GetIfTable</documentation>
  <input message="tns:GetIfTableRequest"/>
  <output message="tns:GetIfTableResponse"/>
 </operation>
</portType>
```

The other Web services support more operations; the following listing shows part of the <portType> element for the *GetIfColumn* Web service.

```
<portType name="GetIfColumnServicePortType">
 <operation name="getIfIndex">
  <documentation>function utMon__getIfIndex</documentation>
  <input message="tns:getIfIndexRequest"/>
  <output message="tns:uIntArray"/>
 </operation>
```

```
 <operation name="getIfDescr">
  <documentation>function utMon__getIfDescr</documentation>
  <input message="tns:getIfDescrRequest"/>
  <output message="tns:stringArray"/>
 </operation>
 <operation name="getIfType">
  <documentation>function utMon__getIfType</documentation>
  <input message="tns:getIfTypeRequest"/>
  <output message="tns:uIntArray"/>
 </operation>
 ...
</portType>
```

C.4 <binding> element

A <binding> container element provides concrete information on what protocol is being used for the Web service, and how data is encoded and transported. Similar to the <portType> element, it also includes the operations that are supported by the Web service, as well as the request and response messages associated to each operation.

The following listing shows part of the <binding> element for the *GetIfColumn* Web service. In our prototypes, we use SOAP messages on top of the HTTP protocol. The <binding> elements of the other Web services use the same encodings.

```
<binding name="GetIfColumnServiceBinding"
      type="tns:GetIfColumnServicePortType">
 <SOAP:binding style="rpc"
      transport="http://schemas.xmlsoap.org/soap/http"/>
 <operation name="getIfIndex">
  <SOAP:operation soapAction=""/>
  <input>
   <SOAP:body use="encoded" namespace="urn:utMon"
   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
   <SOAP:body use="encoded" namespace="urn:utMon"
   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </output>
 </operation>
 <operation name="getIfDescr">
  <SOAP:operation soapAction=""/>
  <input>
   <SOAP:body use="encoded" namespace="urn:utMon"
   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </input>
  <output>
   <SOAP:body use="encoded" namespace="urn:utMon"
   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </output>
 </operation>
 ...
</binding>
```

C.5 <service> element

The fifth and last container element, i.e. the <service> element, is used to define the location of the Web service and the binding element that this server supports. The following listing shows the service definition for the *GetIfRow* Web service.

```
<service name="GetIfRowService">
 <documentation>GetIfRow service</documentation>
 <port name="GetIfRowService"
 binding="tns:GetIfRowServiceBinding">
  <SOAP:address location="http://yourhost.com"/>
 </port>
</service>
```

III. Implementation

This section describes the implementation of the previously defined Web services. Firstly, a Web services toolkit is chosen to support the handling of encoding and decoding of the Web services messages. This toolkit should preferably also assist in the conversion of the WSDL descriptions into program code. Secondly, an overview is given of the prototype implementation. Finally, the instrumentation of the prototype implemententation to support performance measurements is discussed.

A. Web services toolkits

A toolkit is used to support the prototype implementations with regard to handling WSDL. We have found three toolkits that support the same programming language as Net-SNMP, i.e., the C programming language. It is highly recommended that the toolkit of choice supports the same programming language as Net-SNMP, so that the code of Net-SNMP that retrieves data from the operation system kernel (which keeps track of counters necessary for the ifTable) can be re-used. We consider the following toolkits:

1. easySOAP++ [14],
2. WASP server for C++ [15], and
3. gSOAP [16].

A short description of each of the toolkits will be given next, in order to illustrate their capabilities, and to select one toolkit that suits our needs best.

A.1 EasySOAP++

The easySOAP++ Web services toolkit is an open source initiative intended to create Web services very fast. The toolkit uses a very simple API to create, read, write and remove elements in messages. The toolkit does not do any validity checking of the messages. There are no apparent compatibility problems between easySOAP++ and other Web services toolkits.

The easySOAP++ toolkit does have several drawbacks, however. The first drawback is that this toolkit is only able to act as a server. This means that another toolkit must to be used to create a client that communicates with the easySOAP++ server. The second drawback is that the easySOAP++ toolkit does not support WSDL descriptions in any way. The last drawback is that there is no documentation for the toolkit except for some examples. Together, these three drawbacks make the easySOAP++ toolkit unsuitable for our purposes.

A.2 WASP server for C++

The WASP server for C++ (WASP) is a commercial product. This toolkit features an integrated development environment, which is not available for the other toolkits.

The WASP toolkit can use a WSDL description to create Web services. The toolkit can create both a server and a client, making it possible to use a single toolkit for both manager and agent in our research.

Contrary to easySOAP++, the WASP toolkit comes with extensive documentation, including tutorials, reference guides, etc. Also the WASP toolkit creator, Systinet, mentions an excellent performance [17],

although we have not been able to verify this via an independent source.

There is only one relatively minor problem with the WASP toolkit, i.e., it does not support all Linux versions: just some versions of Redhat and Suse. Concluding, the WASP toolkit is a viable option for implementing our prototypes.

A.3 gSOAP

The gSOAP toolkit is, like easySOAP++, an open source initiative. The gSOAP toolkit can create both a client and a server. This toolkit can be used to create Web services in two ways:

1. A WSDL description is supplied to the toolkit, and the toolkit then generates the necessary code to encode and decode corresponding messages, or
2. An interface description is provided in the C programming language, and this description is then used to create the necessary code and a WSDL description.

Extensive documentation on the use of the gSOAP toolkit is available, making it relatively easy to use. Independent sources report that the performance of gSOAP is good [18]. The gSOAP toolkit supports compression of the messages by using either the ZLIB [19] or deflate [20] algorithm. Compression is not supported by the other toolkits. Because we would like to investigate the influence of data compression on our Web services, this is an advantage of gSOAP over the WASP toolkit.

Because of the advantages of gSOAP compared to the other toolkits, we have chosen the gSOAP toolkit as the basis for our prototype implementation.

B. Agent structure

Figure 3 shows the structure of the Web services prototype implementations. The prototypes are divided into two main areas, i.e., Web services handling and the management data retrieval.

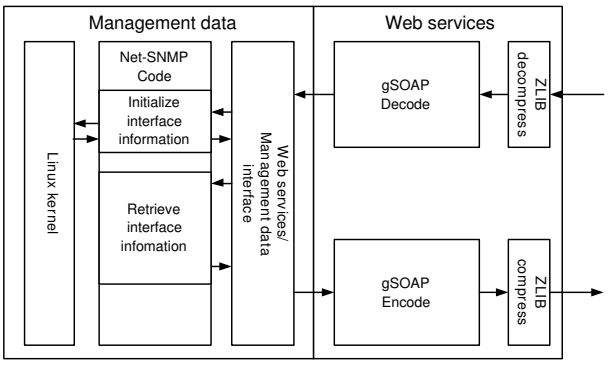

Fig. 3. Structure of the Web services agents

The Web services handling area encompasses all processing that takes care of encoding, decoding, compressing and decompressing of messages, see subsection *C*. The management data retrieval area analyzes each request, retrieves the management data that is requested (e.g., from the operating system kernel), and responds with this management data, see subsection *D*.

C. Web services handling

The Web services handling area provides two sets of functionality: the encoding and decoding of SOAP messages, and the (de)compression of these SOAP messages (when necessary).

As part of the message decoding, the gSOAP toolkit validates the received SOAP messages on their conformance to the WSDL description of the agent. Based on the operation mentioned in the SOAP message, a specific routine is then called to retrieve the requested management data (see next subsection for details). After the routine for management data retrieval is finished, the data returned by the operation is used to create a response message that will be transmitted to the manager.

(De)compression of SOAP messages is handled by ZLIB. The gSOAP toolkit detects whether an incoming message has been compressed or not, and calls the ZLIB decompression routines in case decompression is needed. If a request was compressed, the Web services agent will automatically compress the response message.

D. Management data retrieval

The network management data of the ifTable that is requested through the Web service, is retrieved from the operating system kernel. This area can be divided into three parts, as shown in Figure 3: an interface between the Web service and the actual data retrieval functions, the re-used Net-SNMP program code to retrieve information from the kernel, and the kernel itself.

The first part consists of routines that correspond to the operations as defined in the WSDL descriptions of the agents (see Section II-C.3). These routines call the necessary functions from the Net-SNMP code, to retrieve the requested information. The resulting data is then put in a data structure that the gSOAP toolkit can use to construct the response message.

The second part is the re-used code from Net-SNMP, i.e., program code that retrieves information from the operating system kernel. The Net-SNMP code opens, reads and parses a "virtual file" in Linux' `/proc` filesystem that contains management information of the network interfaces, i.e., `/proc/net/dev`.

The third part is the Linux kernel, which, among all its other tasks, keeps track of, e.g., names of network interfaces and the amount of transmitted data. It updates the "virtual file" that is used by the Net-SNMP code with the latest information every time the file is read.

E. Instrumentation of measurements

In order to compare the performance of our various Web services based network monitoring prototypes with each other, and with Net-SNMP in the future, we will look at the following performance measures:

1. *Bandwidth consumption*: the amount of data transmitted by an operation.
2. *CPU time consumption*: the amount of time an operation takes, and how this time is divided over the different parts of the Web services prototypes.
3. *Memory consumption*: the amount of memory used by the Web services prototypes.
4. *Round Trip Time*: the amount of time that an invoking manager has to wait for a response on a request made to an agent.

The *bandwidth consumption* and *round trip time* can be measured without modifying the network monitoring agents: *bandwidth consumption* can be measured by a network analyzer (such as Ethereal), and the *round trip time* can be measured at the manager side.

To measure the *CPU time consumption*, however, some modifications of the Web services prototypes have to be made for calculating the time difference between the beginning and ending of various functions. To retrieve these time differences, the Linux `gettimeofday` function is used. The `gettimeofday` function returns the current time with a precision of at least ten milliseconds. However, with modern processors the precision is even better, and likely in the order of a few microseconds. We distinguish two aspects: the total time an operation takes to complete, and the time needed for data retrieval from the operating system kernel. By subtracting the time for data retrieval from the total time, the time for message processing can be calculated.

Finally, to measure *memory consumption*, the *dMalloc* library [21] is used, by linking our Web services prototypes against this library. The *dMalloc* library keeps track of all memory allocations, and writes a summary to disk when the application is terminated. Subsequent analysis of this data yields memory consumption information.

IV. Preliminary measurement results

For the present paper we have only investigated the bandwidth consumption of our 4 Web services based network monitoring agents; the other three performance measures will be discussed in future work.

We have focused on bandwidth consumption measured at the application layer and on the IP layer, see Figure 4. The IP layer measurements will give bandwidth consumption values that can be compared to other applications, which may use a different protocol stack (e.g., SNMP over UDP). The application layer measurements will show the bandwidth consumption of the Web services themselves, without the overhead from the HTTP, TCP, and IP protocols.

The Web services based network monitoring agents allow for compression of messages. We have performed our measurements for both uncompressed and compressed messages. To indicate the effectiveness of compression on different message sizes, we have performed measurements of the *GetIfTable* Web service for a varying number of network interfaces (between three and fifteen). For the other measurements, the measurement system was equipped with three network interfaces (Ethernet, local loopback, and tunnel).

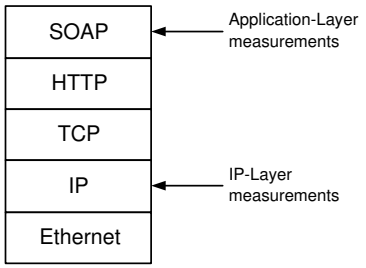

Fig. 4. Protocol stack of the Web services prototypes

A. IP layer bandwidth consumption

Figure 5 shows the IP layer bandwidth consumption of the Web service prototypes. The figure shows that in case of uncompressed data, the bandwidth consumption grows relatively fast if the amount of data increases. To retrieve a single object, approximately 1.8 Kbyte of data is transmitted; to retrieve the entire ifTable, 5 Kbyte is required. Although the *GetIfTable* operation uses more bandwidth than the *GetIfCell* operation in absolute values, the *GetIfTable* operation retrieves 54 data objects at once, while the *GetIfCell* operation retrieves only one data object at a time. Therefore the *GetIfTable* is relatively more efficient per data object than the *GetIfCell* operation.

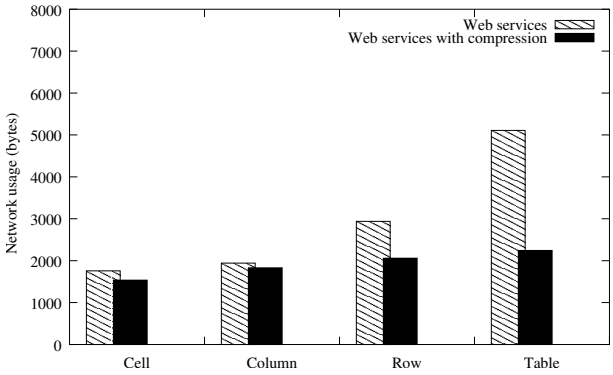

Fig. 5. Network usage on the IP layer

In case compression is used, the absolute differences between the different Web services are much smaller than without compression. For the IP layer the bandwidth consumption is decreased by 13% for the *GetIfCell* operation, up to 56% for the *GetIfTable* operation when compared to the bandwidth consumption of uncompressed data. The compression of the actual SOAP messages is even better, because the data at the IP, TCP and HTTP layers is not compressed and, hence, these layers use the same bandwidth either with or without compression of the Web services.

B. Application layer bandwidth consumption

At the application layer, the bandwidth consumption is measured without the overhead of the underly-

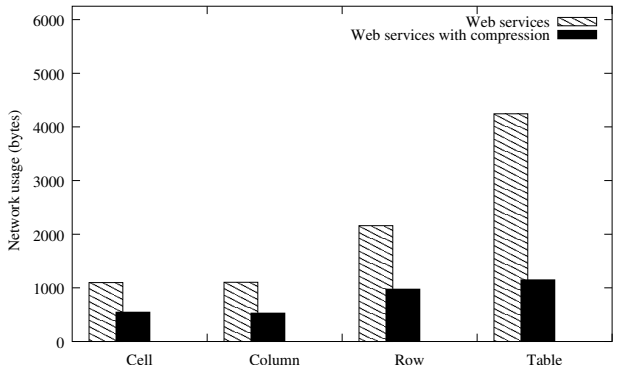

Fig. 6. Network usage on the application layer

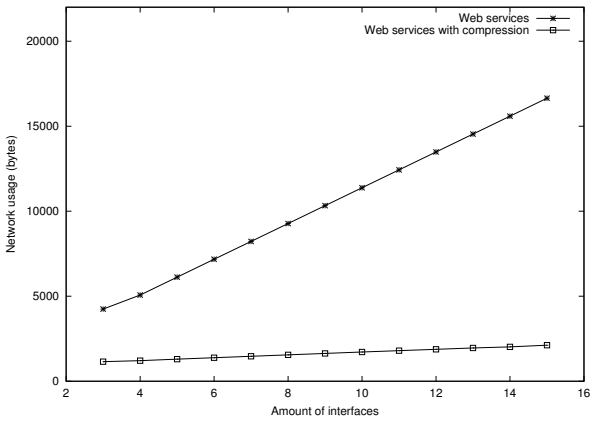

Fig. 7. Network usage of the GetIfTable prototype for different amounts of interfaces

ing protocols. For instance, the TCP protocol introduces overhead due to connection setup, connection termination, and ACK packets. The overhead generated by the underlying protocols does not change much among the different Web services.

Figure 6 shows the bandwidth consumption of the Web services at the application layer. From the figure it can be seen that the (total) size of the SOAP messages to retrieve a single cell is approximately 1 Kbyte. When retrieving the entire ifTable, the size of the SOAP messages amounts to some 4.2 Kbyte.

Comparing Figure 6 to Figure 5, it can be seen that the overhead introduced by the underlying protocols ranges from 500 to 1000 bytes. This implies that the relative difference in bandwidth consumption between the various Web services increases.

The effect of compression is now more apparent, due to the absence of the overhead of the lower level protocols. The compression now reduces the size of the messages to between 50% (GetIfCell) and 27% (GetIfTable) of the size of the uncompressed SOAP messages.

The (absolute) differences between the operations in terms of bandwidth consumption are not very large: the largest operation, i.e., *GetIfTable*, requires only between 3.9 times (no compression) and 2.1 times (compression) times as much bandwidth as the *GetIfCell* operation, while the *GetIfTable* operation transmits 54 times as much management data.

C. Bandwidth usage for larger amounts of interfaces

Figures 5 and 6 have shown that compression reduces larger messages more than smaller messages. In order to determine the effect of compression on even larger amounts of data, we have measured the bandwidth consumption of the *GetIfTable* operation while adding up to 12 (virtual) tunnel interfaces to the measurement system's configuration.

To prevent unrealistic good compression due to a high number of zeros, the values of the "ifInOctet", "ifOutOctet", etc. columns are randomly generated. Figure 7 shows the results of these measurements. The figure shows that compression reduces the bandwidth consumption per data object in case more interfaces are added (which increases the number of rows in the ifTable). In our case the compression reduced the bandwidth consumption to between 27% of the uncompressed variant for three interfaces, and only 13% of the original size for the combination of 3 physical and 12 tunnel interfaces.

V. Conclusion and Future Work

In this paper we have presented the design and implementation of four Web services based network monitoring prototypes. Each prototype follows a specific approach to retrieve management data; the data we have concentrated on is that of the interfaces table (ifTable), as described in the IF-MIB.

The differences in approach to retrieve management data, are based on the granularity of the retrieval operations (WSDL messages). The GetIfCell prototype allows a management application to retrieve only one data object from the ifTable at a time, the GetIfColumn prototype allows the retrieval of one complete column, the GetIfRow prototype allows retrieval of one complete row of the ifTable at a time, and the GetIfTable retrieves the complete ifTable at once.

This paper also provided some preliminary measurement results. These results show the bandwidth needed by our prototypes, the effect of compression, and the effect of having more objects (15 ifTable rows instead of 3).

The bandwidth usage without compression, measured at the application layer (i.e., SOAP messages), is between 1 Kbyte per operation for the GetIfCell prototype, and 4.2 Kbytes for the GetIfTable operation. When compression is used, the required bandwidth decreases with 50% for the GetIfCell prototype (to 0.5 Kbyte), and with 73% for the GetIfTable operation (to 1.1 Kbyte). With a larger ifTable, containing 15 interfaces, the decrease in bandwidth is even 87% (from 16.6 Kbyte to 2.1 Kbyte).

A. Future work

This paper presented some initial performance figures; additional measurements are still needed. Examples of such future measurements are CPU usage, memory usage, and the round trip time for complete

operations. These figures will also be compared to those obtained from traditional SNMP agents. The results of these measurements, as well as the comparison with SNMP agents, will be described in a future paper

References

[1] P. Schafer, "*XML-Based Network Management*", White paper, Juniper Networks, Aug. 2001.

[2] Network Configuration. *netconf* Working Group, 2003. Available at: *http://www.ietf.org/html.charters/netconf-charter.html*. IETF

[3] F. Cubera et al., "*Unravelling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI*", IEEE Internet Computing, vol. 6, Issue 2, pages 86-93, March/Apil 2002.

[4] J. Schönwälder et al., "*On the Future of Internet Management Technologies*", IEEE Communications Magazine, pages 90-97, October 2003.

[5] Y. J. Oh et al. "*Interaction Translation Methods for XML/SNMP Gateway*", 13th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM 2002), pages 54-65, October 2002

[6] R. Neisse et al., "*Implementation and Bandwidth Consumption Evaluation of SNMP to Web Services Gateways*", To be published at NOMS 2004.

[7] J. Case, M. Fedor, M. Schoffstall, and J. Davin. "*A Simple Network Management Protocol (SNMP)*", IETF RFC 1157, May 1990.

[8] Information processing systems - Open Systems Interconnection, "*Specification of Abstract Syntax Notation One (ASN.1)*", International Organization for Standardization, International Standard 8824, December 1987.

[9] Information processing systems - Open Systems Interconnection, "*Specification of Basic Encoding Rules for Abstract Notation One (ASN.1)*", International Organization for Standardization, International Standard 8825, December 1987.

[10] SNMPv2 Working Group, J. Case, K. McCloghrie, M. Rose, S. Waldbusser, "*Structure and Identification of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2)*" RFC 1902, January 1996.

[11] R. Chinnici et al., "*Web Services Description Language (WSDL) Version 1.2 Part 1: Core Language*", W3C Working Draft, June 2003.

[12] K. McCloghrie, F. Kastenholz, "*The Interfaces Group MIB*", IETF RFC 1902, June 2000.

[13] W. Hardaker et al., "*Net-SNMP package*", available from: `http://www.net-snmp.org/`.

[14] D. Crowley et al., "*EasySOAP++ Web services toolkit*", available from: `http://easysoap.sourceforge.net/`.

[15] Systinet, "*WASP server for C++ Web services toolkit*", available from `http://www.systinet.com/products/wasp_cserver`.

[16] Robert A. van Engelen, "*gSOAP Web services toolkit*", 2001, available from: `http://www.cs.fsu.edu/ engelen/soap.html`.

[17] Systinet, "*WASP server for C++ Product Datasheet*", Available from: `http://www.systinet.com/`.

[18] K. Chiu and M. Govindaraju and R. Bramley, "*Investigating the Limits of SOAP Performance for Scientific Computing*", 2002.

[19] P. Deutsch and J. L. Gailly. "*ZLIB compression Data Format Specification version 3.3*", IETF RFC 1950, May 1996.

[20] P. Deutsch, "*Deflate compression Data Format Specification version 1.3*", IETF RFC 1951, May 1996.

[21] G. Watson, "*Debug Malloc Library (dMalloc)*", available from: `http://www.dmalloc.com`.