Journal of King Saud University – Computer and Information Sciences (2013) **25**, 35–42

King Saud University

Journal of King Saud University – Computer and Information Sciences

www.ksu.edu.sa
www.sciencedirect.com

ORIGINAL ARTICLE

Feasibility of SNMP OID compression

Hari T.S. Narayanan [a,*], **Geetha Ilangovan** [a], **Sumitra Narayanan** [b]

[a] *Netprowise Consulting, Chennai 600033, India*
[b] *SRM University, Ramapuram, Chennai 600089, India*

Received 18 December 2011; revised 9 May 2012; accepted 29 May 2012
Available online 8 June 2012

KEY WORDS

Protocol;
SNMP;
OID compression;
Design;
OID encoding;
ASN.1;
Net-SNMP;
Experimental research

Abstract Simple network management protocol (SNMP) object identifier (OID) compression can improve bandwidth usage and response time. The current literature includes several OID compression algorithms to reduce redundancy in SNMP protocol data units (PDUs). The overhead of OID compression could outweigh the benefits it offers if its tradeoffs are not well understood. The main objective of this paper is to investigate the OID compression as a viable feature for SNMP libraries. This is done by adding an OID compression algorithm to Net-SNMP, which is one of the popular open source implementations of the SNMP framework. Change to image size, lines of code added, complexity of compression code, the effect of compression on response time, and testing effort required are the parameters presented to understand the viability of OID compression.

© 2012 King Saud University. Production and hosting by Elsevier B.V. All rights reserved.

1. Introduction

Simple network management protocol (SNMP) (Levi et al., 2002; Presuhn, 2002a,b,c; McCloghrie et al., 1999; William, 1998) is a popular application layer protocol used in managing data networks. Almost all networking vendors support SNMP. A number of telecom equipment vendors are also starting to support SNMP in order to enable integrated management. A significant amount of network management activity is currently carried out using the SNMP framework by a large percentage of enterprise networks as well as a sizeable number of service provider core networks. The rest of this section provides a simple introduction to SNMP needed to understand the content of this paper; a comprehensive introduction to SNMP is found in William (1998).

The SNMP framework includes, besides other features, an information model and an application layer protocol. This information model provides an abstract as well as virtual representation of the networking element to be managed; the protocol is used in exchanging requests and responses between network management tasks. The information model of SNMP is graphically represented by an inverted tree—a MIB (management information base) tree. In this tree, every leaf node contains the data items to be Get and Set; every non-leaf node represents a grouping of related data items. Each node in the MIB tree is identified by its path from the root of the tree.

The data items of the information model are carried in a list within the SNMP request and response messages – Fig. 1. This list is referred to as the varbind list. Each data item in the list is referred to as a varbind and represents an element in the information model. Each varbind includes a unique identifier–Object identifier (OID) and a value. An OID is a sequence of sub-identifiers (integers) where each sub-identifier is a natural number associated with a node in a MIB tree. The

[*] Corresponding author. Tel.: +91 44 6565 1234, mobile: +91 9841289525.
E-mail address: ts.hari@gmail.com (H.T.S. Narayanan).

Peer review under responsibility of King saud University.

Production and hosting by Elsevier

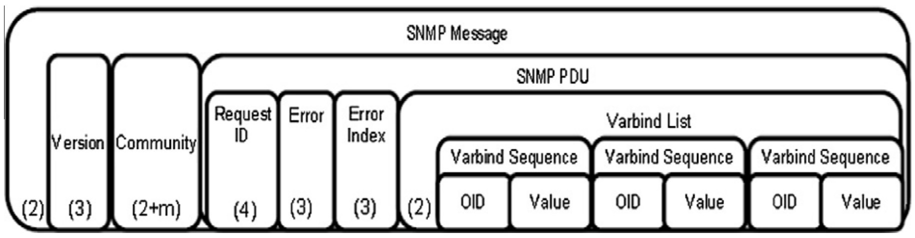


Figure 1 SNMP message format.

entire SNMP message is encoded using Abstract Syntax Notation's Basic Encoding Rules (ASN.1—BER) (ITU-T Recommendation X.690, 2003). The representation of an OID in a varbind list provides an opportunity to save bandwidth when multiple data items appear in a varbind list. This optimization is referred to as SNMP OID compression. This optimization could provide more valuable savings in CPU cycles as compared to bandwidth requirements for management tasks as highlighted in Hari et al. (2010). There are three algorithms proposed for this purpose in the literature (Schoenwaelder, 2001; McLeod et al., 2001; Hari et al., 2011); the effectiveness of such compression algorithms was discussed in Hari et al. (2010). This paper validates the feasibility of one such algorithm by adding the algorithm to one of the open source SNMP framework libraries [Net-SNMP].

This paper is organized as follows. Section 2 provides a comparative overview of various OID compression algorithms. The objective here is to introduce their relative values and not to investigate their relative values in depth; such an investigation is part of Hari et al. (2010). Section 3 describes normal SNMP *OID* encoding. Section 4 describes the OID compression algorithm that is used for this feasibility study. This is one of our own algorithms and it is integrated into Net-SNMP open framework for this investigation. Section 5 provides feasibility results based on the OID compression extension made to Net-SNMP. This section includes image size, response time, and other performance values. Section 6 concludes the paper and lists further work in this area.

2. Other OID compression algorithms

In general, network performance optimization procedures are either suggested for transport layers as in Alnuem (2010) or for application layers. The algorithm presented in this paper belongs to the latter one.

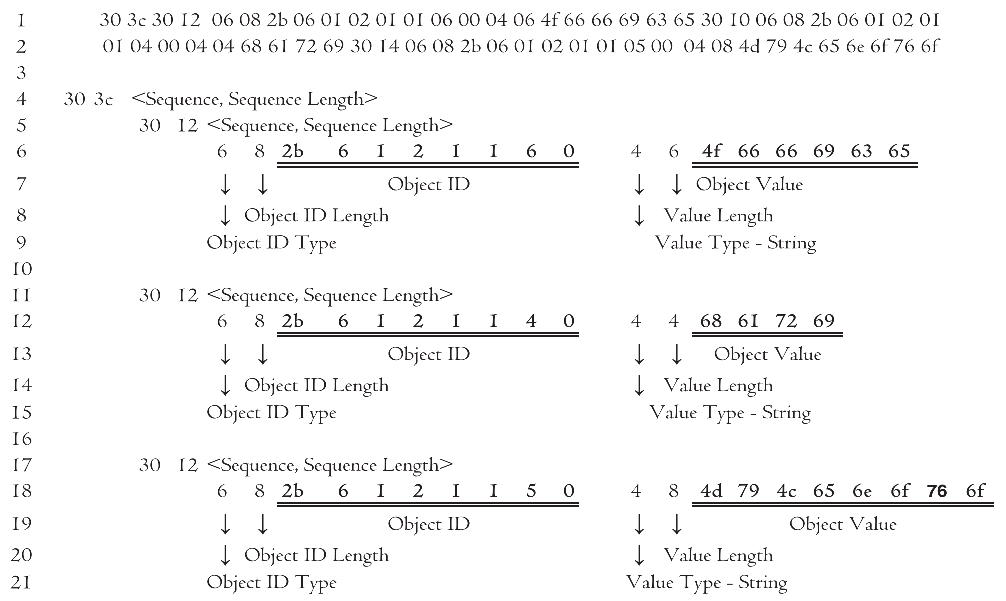
There are three algorithms (Schoenwaelder, 2001; McLeod et al., 2001; Hari et al., 2011) for OID compression in the current literature. All these algorithms code OID compression with respect to the preceding object identifier in a varbind list; the first OID is coded without any compression. The first of the three algorithms (Schoenwaelder, 2001), OID Delta Compression (ODC), uses a combination of the following three different encodings for compressed representation for an OID: *single sub-identifier substitution, range of sub-identifiers substitution, and truncation*. This eliminates more redundancy in a compressed OID than what is suggested in McLeod et al. (2001), Hari et al. (2011). The second of the three algorithms (McLeod et al., 2001) codes only the OID tail replacement with respect to the preceding OID. This scheme fails to eliminate the redundancy in the tail that appears after the point of divergence with respect to anchor OID. However, its compression logic and decompression logic are simple to code and maintain. The OID compression studied in this paper is the algorithm presented in Hari et al. (2011). It encodes only the *range of sub-identifiers substitution* and *truncation*. This is because single sub-identifier substitution is treated as range substitution with a single sub-identifier. The substitution and truncation are represented using a unified encoding structure—location of substitution, span of substitution, and new sub-identifier string to be substituted. The algorithm suggested in Hari et al. (2011) does not include the special optimization suggested in algorithm (McLeod et al., 2001) for OIDs which are longer than 119. The *uniform and simple compressed OID structure* and *lack of special cases* lead to simpler implementation without compromising the compression opportunities listed in Schoenwaelder (2001).

There are two relevant patents applications found in US Patent search. The first one by Ghirardi (2010) describes a compression method for the entire payload of a message. Their compression method is generic in nature and requires extension to the data access protocol to which it is applied, thus making it difficult to develop and deploy. Their method can still be applied to the payload after applying our encoding method. The second patent Aseem (2008) describes a compression method for the storage of hierarchically structured data. Their method is only suitable for data storage. Additionally, the exchange of such encoding is justified only for large volumes of data exchange.

3. SNMP OID encoding

The format of an SNMP message is shown in Fig. 1. The entire message is coded using an ASN.1 derivative. The varbind list appears at the end of this message as an ASN.1 sequence. This sequence (varbind list) contains one or more varbind. Each varbind is a sequence by itself representing a data item from the underlying MIB. This sequence includes an OID and its (object) value. All the items in the varbind list, including the varbind list, are coded using the Type/Tag-Length-Value (TLV) format using ASN.1 BER (ITU-T Recommendation X.690, 2003).

Fig. 2 illustrates the TLV encoding of 3 data items in a varbind list of an SNMP message. Lines 1 and 2 are the hex-dump of the varbind list from an SNMP message with 3 data items in it. Lines 3 to 21 are the parsed output of this varbind list for illustration. Value 30 in line 3 represents an ASN.1sequence type and value 3c (decimal 50) identifies its (varbind list) length in octets. All sequences (in line 4, 10, and 18) are coded the same way—30 to indicate sequence, and followed by the sequence length in number of octets. Lines 5–9 describe the coding of the first data item in the varbind list—06 represents the object identifier type and 04 represents the string type. The string type 04 in line 5 is followed by string length 06 and the actual string in ASCII.



```
30 3c 30 12  06 08 2b 06 01 02 01 01 06 00 04 06 4f 66 66 69 63 65 30 10 06 08 2b 06 01 02 01
01 04 00 04 04 68 61 72 69 30 14 06 08 2b 06 01 02 01 01 05 00  04 08 4d 79 4c 65 6e 6f 76 6f
```

Figure 2 Illustration of varbind list encoding.

All the three objects are of string type in this case and there are a number of other types supported by SNMP. The three object identifiers are: **2b 06 01 02 01 01** 06 00, **2b 06 01 02 01 01** 04 00, and **2b 06 01 02 01 01** 05 00. All the three object identifiers have the same prefix "2b 06 01 02 01 01". In this case, they also differ by one sub-identifier—the 7th one. The OID compression reduces these redundancies to conserve the number of octets used in encoding the varbind list. It is possible to construct the object identifier of the 2nd object to be sent from the 1st object identifier and the difference between the 1st and the 2nd object identifiers. The second object identifier is compressed with respect to its normal form. The receiving side can re-construct the 2nd object's object identifier from the 1st object's object identifier and the encoded difference between the two (i.e., the compressed 2nd object identifier).

4. OID compression algorithm implemented

Table 1 illustrates one of the OID compression algorithms (Hari et al., 2011) described in the literature. This algorithm is implemented and integrated with the Net-SNMP framework for this study. The following terms are used in this illustration:

- The 'Anchor ID' (A) is the Object Identifier of the 1st data that needs to be transmitted.
- The 'Object ID' (O) is the Object Identifier of the 2nd data that needs to be transmitted.
- The term offset refers to the distance (in sub-identifier units) from the **last** sub-identifier of the Anchor ID. The distances to the left of the last sub-identifier (of the Anchor) are considered positive and the distances to the right are considered negative. Since the distance in sub-identifier units between the last sub-identifier and itself is 0, the offset for the last sub-identifier is 0 units.
- The matching prefix is not sent for the 2nd object identifier; only the mismatching part needs to be sent. The mismatch length is also specified in number of sub-identifiers. Length of the mismatch sequence is one of the outputs of the compression method.

Table 1 Compression encoding when length of Object OID (O) is equal to Anchor OID (A).

Type No.	Inputs to compression method: A and O										Case description	Savings in number of subids	Output from encoding method		
		1	2	3	4	5	6	7	8	9			Offset from end	Suffix length	Suffix or embed
		Same length													
1	A	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	A = O				
	O	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9		8	0	0	0
2	A	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	Suffix mismatch				
	O	a_1	a_2	a_3	a_4	a_5	a_6	a_7	o_8	o_9		6	0	2	o_8–o_9
3	A	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	Embedded -mismatch				
	O	a_1	a_2	a_3	a_4	a_5	o_6	o_7	a_8	a_9		6	2	2	o_6–o_7
4	A	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	Scattered mismatch				
	A	a_1	a_2	a_3	a_4	o_5	o_6	a_7	a_8	o_9		3	0	5	o_5–o_9

Note: The table does not show scattering with multiple matching groups. If there are multiple matching groups, the left most group is the one that is considered for encoding.

Typically, sub-identifiers are natural numbers in the range of 1–128. In general, this needs not be true; this algorithm can handle sub-identifiers that are more than one octet. The offset is extra information carried along in all compressed encodings. Thus, if there are five matching sub-identifiers, then compression saves five units of space (since those five matching sub-identifiers are seen as redundant information, they need not be sent), and uses one extra unit of space to code the offset. This results in savings of four units of space. Savings are also calculated in sub-identifier units.

The inputs for the object compression encoding method are the Anchor ID and Object ID. The output from the method includes the offset, mismatch sequence, and mismatch sequence length. Each compressed object encoding includes the offset, mismatch sequence, and mismatch sequence length with respect to the Anchor ID. The encoding can be extended when there are more than two objects in a message. The first object identifier is sent without any compression and as for the second object, only the difference is sent. The second object identifier is constructed at the receiving end from the anchor object's identifier and the encoded difference. If there is a 3rd object, then it can be encoded and decoded with respect to 2nd object identifier, and so on. The mismatch between the anchor and the object identifier to be compressed can be classified into 12 different types. These twelve types are explained in detail in the following paragraphs using Table 1.

4.1. When the Anchor ID and Object ID are of equal length

The following table (Table 1) illustrates the four types of mismatches when the Anchor ID (9 sub-identifiers in length) and the Object ID (also 9 sub-identifiers in length) are exactly the same, meaning that the same node is being transmitted twice. All 9 sub-identifiers of the 2nd object ID need not be transmitted, since there is no mismatch between the Object ID and the Anchor ID. The encoding process for obtaining the Object ID from the Anchor ID is as follows: The offset of the mismatch sequence (which doesn't exist here) is 0. The mismatch sequence length is also 0 since there is no mismatch here.

In the second type, there is a suffix mismatch, meaning that a continuous sequence of sub-identifiers differs only at the end of the two identifiers. From the example in the table, we can see that there is no need to transmit the first 7 sub-identifiers since the first 7 sub-identifiers are exactly the same for both the Anchor ID and the Object ID. The encoding process for obtaining the Object ID from the Anchor ID is as follows: The offset of the mismatch sequence is 0, since the mismatch starts right from the last sub-identifier of the Anchor ID. The mismatch sequence length is 2, since the ending 2 sub-identifiers differ between the Anchor ID and the Object ID.

In the third type, there is an embedded mismatch, which indicates that some inner continuous sequence of sub-identifiers differs between the Anchor ID and the Object ID. In the example from the table, the 6th and 7th sub-identifiers differ between the Anchor ID and the Object ID. Hence, there is no need to transmit the 5 sub-identifiers that occur before the 6th sub-identifier and the two sub-identifiers that occur after the 7th sub-identifier, which is total savings of seven sub-identifiers. The encoding process for obtaining the Object ID from the Anchor ID is as follows: The offset of the mismatch sequence is +2, since the mismatch starts from the 7th sub-identifier. The mismatch sequence length is 2, since it includes the 6th sub-identifier and the 7th sub-identifier.

In the fourth type, there is a scattered mismatch, which means that the mismatches are scattered and do not occur in a continuous sequence. They occur discontinuously within the ID. In the above example, the 5th, 6th, and 9th sub-identifiers differ between the Anchor ID and the Object ID. Hence, there is no need to transmit the first 4 sub-identifiers that occur before the 5th sub-identifier. From the 5th sub-identifier onward, there is a mismatch, so even though the 7th and 8th sub-identifiers in between the 5th and 9th match, they must also be replaced so that the replacement occurs in a continuous sequence. The encoding process for obtaining the Object ID from the Anchor ID is as follows: The offset of the mismatch is 0, since the mismatch occurs right from the ending sub-identifier. The mismatch sequence length is 5 since all sub-identifiers right from the 9th to the 5th are to be replaced.

4.2. When the Anchor ID is of longer length than the Object ID

The following table (Table 2) illustrates the four types of mismatches when the Anchor ID is longer than the Object ID. In the first type of this category, the Object ID is a prefix subsequence of the Anchor ID. This is known as a prefix subsequence match. The Object ID is made up of the first 7 sub-identifiers of the Anchor ID, and those 7 sub-identifiers need not be transmitted. The encoding process for obtaining the Object ID from the Anchor ID is as follows: The offset of the mismatch sequence from the end of the Anchor ID is +2, and the mismatch sequence to be appended to the start of the Anchor ID's offset is a null sequence. The mismatch sequence length is 0 (since it is a null sequence).

The second type of overlap here is a suffix mismatch. The ending sequence of sub-identifiers differs between the Anchor ID and the Object ID. In the above example, the 6th and 7th sub-identifiers are the differing ones. The Object ID also lacks 8th and 9th sub-identifiers which the Anchor ID possesses. The first 5 sub-identifiers of the Anchor ID need not be transmitted, so the savings made here are 5 sub-identifiers. The encoding process for obtaining the Object ID from the Anchor ID is as follows: The offset of the mismatch sequence from the end of the Anchor ID is +2 units, since the mismatch starts from the 7th sub-identifier and also includes the 6th sub-identifier. The mismatch sequence length is 2 units.

The third type of overlap is a special scattered mismatch. Here, the mismatch with respect to the Anchor ID appears to be scattered, whereas with respect to the Object ID, it appears to be embedded. In the above example, the 5th and 6th sub-identifiers of the Object ID differ from those of the Anchor ID's and the Object ID also does not have the 8th and 9th sub-identifiers which are present in the Anchor ID. There is no need to transmit the first 4 common sub-identifiers and the savings made are thus 4 sub-identifiers. The encoding process for obtaining the Object ID from the Anchor ID is as follows: The offset of the mismatch sequence is +2 units, and this includes the matching central sub-identifier (the 7th sub-identifier), apart from the differing 5th and 6th sub-identifiers, in the mismatch sequence. The matching sub-identifier is included to make the encoding simple and uniform. A similar approach is used in other scattered mismatches. The mismatch

Table 2 Compression encoding when length of Anchor OID (A) is longer than Object OID (O).

Type No.	Inputs to compression method: A and O										Case description	Savings in number of subids	Output from encoding method		
		1	2	3	4	5	6	7	8	9			Offset from end	Suffix length	Suffix or embed
		Longer anchor identifier													
1	A	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	O is prefix of A				
	O	a_1	a_2	a_3	a_4	a_5	a_6	a_7	x	x		6	2	0	0
2	A	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	Suffix mismatch				
	O	a_1	a_2	a_3	a_4	a_5	o_6	o_7	x	x		4	2	2	o_6–o_7
3	A	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	Special scattered				
	O	a_1	a_2	a_3	a_4	o_5	o_6	a_7	x	x		3	2	3	o_5–o_7
4	A	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	Scattered mismatch				
	A	a_1	a_2	a_3	o_4	o_5	o_6	o_7	x	x		2	2	4	o_4–o_7

Note: The table does not show scattering with multiple matching groups. If there are multiple matching groups, the left most group is the one that is considered for encoding.

sequence length is 3 units, since it includes the 5th, 6th, and 7th sub-identifiers.

The fourth type of overlap here is a scattered mismatch. Here, the mismatch sequence within the IDs is discontinuous. In the above example, the 4th, 5th, and 7th sub-identifiers differ between the Anchor ID and the Object ID. Also, the Object ID does not contain the 8th and 9th sub-identifiers which the Anchor ID has. The first 3 sub-identifiers which are common between the Anchor ID and the Object ID need not be transmitted, and thereby savings of 3 sub-identifiers are made. The encoding process for obtaining the Object ID from the Anchor ID is as follows: The offset of the mismatch sequence is +2 units, since the mismatch is from the 7th sub-identifier onward. The mismatch sequence length is 4 units, since it includes the 7th sub-identifier, the matching 6th sub-identifier, and the mismatched 5th and 4th sub-identifiers.

4.3. When the Anchor ID is of shorter length than the Object ID

The following table (Table 3) illustrates the four types of mismatches when the Anchor ID is shorter than the Object ID. The first type under this category is, again, a prefix subsequence match, as in the previous category. However, in this type, the Anchor ID is shorter and is made up of the first 7 sub-identifiers of the Object ID, and those 7 sub-identifiers need not be transmitted. The encoding process for obtaining the Object ID from the Anchor ID is as follows: The offset of the mismatch sequence is −2 units, since the 8th and 9th sub-identifiers of the Object ID must now be added to the Anchor ID and for this a distance of 2 units must be traversed to the right from the end of the Anchor ID. The mismatch sequence length is 2 units, since the Object ID contains 8th and 9th sub-identifiers which must be added into the offset locations of the Anchor ID to create the Object ID.

The second type of overlap here is a suffix mismatch, where the ending sequence of sub-identifiers differs between the Anchor ID and the Object ID. In the above example, the 6th and 7th sub-identifiers of the Anchor ID do not match those of the Object ID, and moreover, the Anchor ID does not possess the 8th and 9th sub-identifiers which the Object ID has. It is unnecessary to transmit the first 5 sub-identifiers that are common between the Anchor ID and the Object ID, so the savings made are5 sub-identifiers. The encoding process for obtaining the Object ID from the Anchor ID is as follows: The offset of the mismatch sequence is −2 units, since the Object ID contains 8th and 9th sub-identifiers which must now be appended to the Anchor ID, and for this a movement of 2 units to the right from the end of the Anchor ID is required. The mismatch sequence length is 4 units, since we need to add in the 9th, 8th, 7th, and 6th sub-identifiers.

The third type of overlap is a special scattered mismatch. Here, the mismatch with respect to the Anchor ID appears

Table 3 Compression encoding when length of Object OID (O) is longer than Anchor OID (A).

Type No.	Inputs to compression method: A and O										Case description	Savings in number of subids	Output from encoding method		
		1	2	3	4	5	6	7	8	9			Offset from end	Suffix length	Suffix or embed
		Longer object identifier													
1	A	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	O is prefix of A				
	O	a_1	a_2	a_3	a_4	a_5	a_6	a_7	x	x		6	−2	2	o_8–o_9
2	A	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	Suffix mismatch				
	O	a_1	a_2	a_3	a_4	a_5	o_6	o_7	x	x		4	−2	4	o_6–o_9
3	A	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	Special scattered				
	O	a_1	a_2	a_3	a_4	o_5	o_6	a_7	x	x		3	−2	5	o_5–o_9
4	A	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	Scattered mismatch				
	A	a_1	a_2	a_3	o_4	a_5	o_6	a_7	x	x		2	−2	6	o_4–o_9

Note: The table does not show scattering with multiple matching groups. If there are multiple matching groups, the left most group is the one that is considered for encoding.

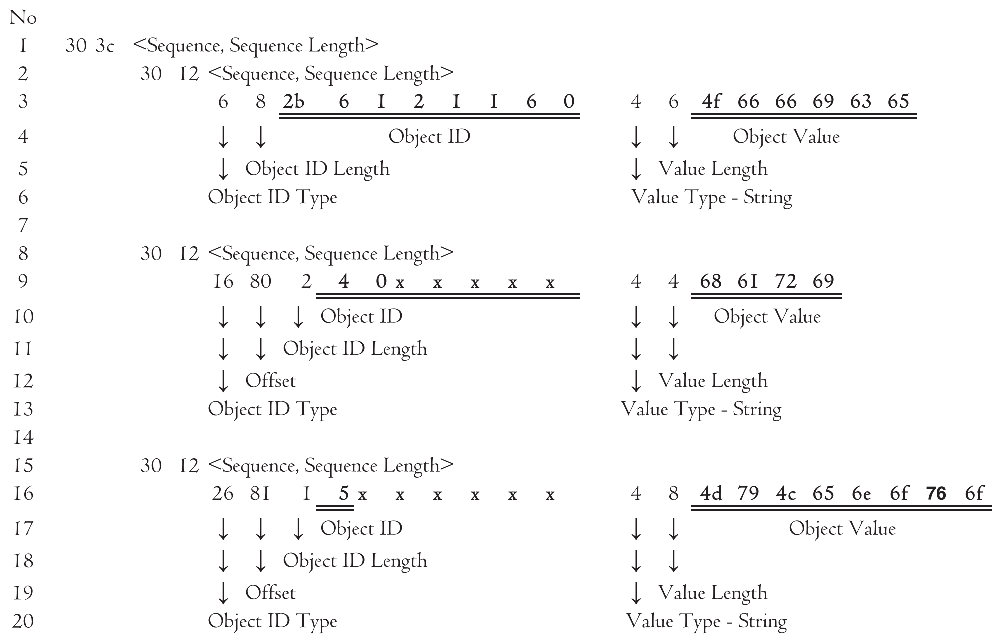

No
1 30 3c <Sequence, Sequence Length>
2 30 12 <Sequence, Sequence Length>
3 6 8 2b 6 1 2 1 1 6 0 4 6 4f 66 66 69 63 65
4 Object ID Object Value
5 Object ID Length Value Length
6 Object ID Type Value Type - String
7
8 30 12 <Sequence, Sequence Length>
9 16 80 2 4 0 x x x x x 4 4 68 61 72 69
10 Object ID Object Value
11 Object ID Length
12 Offset Value Length
13 Object ID Type Value Type - String
14
15 30 12 <Sequence, Sequence Length>
16 26 81 1 5 x x x x x x 4 8 4d 79 4c 65 6e 6f 76 6f
17 Object ID Object Value
18 Object ID Length
19 Offset Value Length
20 Object ID Type Value Type - String

Figure 3 Compressed varbind encoding.

to be embedded, whereas with respect to the Object ID, it appears to be scattered. In the above example, the 5th and 6th sub-identifiers of the Object ID differ from those of the Anchor ID's and the Anchor ID also does not have any 8th and 9th sub-identifiers, which are present in the Object ID. The savings made are 4, since it is not required to transmit the first 4 sub-identifiers which are common between the Anchor ID and the Object ID. The encoding process for obtaining the Object ID from the Anchor ID is as follows: The offset of the mismatch sequence is −2 units, since the Object ID contains 8th and 9th sub-identifiers which must now be appended to the Anchor ID. The mismatch sequence includes the 9th, 8th, 7th, 6th, and 5th sub-identifiers and therefore is 5 units of length.

In the fourth type of overlap, there is a scattered mismatch, which means that the sub-identifiers that differ between the Anchor ID and the Object ID are scattered and do not occur in a continuous sequence. They occur discontinuously within the ID. In the above example, the 4th and 6th sub-identifiers differ between the Anchor ID and the Object ID. The Anchor ID also does not possess the 8th and 9th sub-identifiers which the Object ID has. It is unnecessary to transmit the first 3 sub-identifiers which are common between the Anchor ID and the Object ID. The encoding process for obtaining the Object ID from the Anchor ID is as follows: The offset of the mismatch sequence is −2 units, since the Object ID contains 8th and 9th sub-identifiers which must now be appended to the Anchor ID. The *mismatch sequence* includes the 9th, 8th, 7th, 6th, 5th, and 4th sub-identifiers (even though the 7th and 5th sub-identifiers match, so that the replacement of sub-identifiers may be continuous) and therefore the mismatch sequence length is 6 units.

When an application receives the three output values (offset, mismatch sequence, and mismatch sequence length), it may not be able to differentiate type 3 of Table 1 from types 1 to 4 of Table 2. In type 3, the receiver is supposed to retain the sub-identifiers that appear before the offset, whereas it is supposed to ignore them in types 1–4 of Table 2. In order to alleviate this problem, type 3 of Table 1 is given a special type identifier in the message.

Fig. 3 shows the compressed varbind list corresponding to the one shown in Fig. 2. Although, both the 2nd object and 3rd object in the list could be coded as type 3, for the sake of illustration, the second varbind is coded as type 2, and the third varbind is coded as type 3. Eleven fewer octets are used to encode the three objects in the compressed varbind list. This is not significant savings; however, with a larger number of related fixed size objects, a single compressed message can carry the contents of three normal messages. This is only one of the benefits; the other critical benefit is the number of CPU cycles saved by applications in processing those two extra messages. Two new OID types are to be introduced: type 16 identifies a partial object identifier for all types with the exception of type 2 and type 26 identifies a partial object identifier for type 2. These type values (16 and 26) are used for illustration, eventually, these type values are to be selected suitably and registered with (IANA, 2011). Offsets are coded by adding a bias of 128 (0x80) to convert negative values to positive. Thus, 0 offset is coded as 0x80, 2 is coded as 0x82, −2 is coded as 0x7D. A single octet is sufficient to encode the full range of offset values because the number of sub-identifiers in an OID is limited to 128.

5. Results and analysis

The implementation of OID compression to Net-SNMP provided the following useful results:

1. In all about 800 lines of code (including comments and white space) are added to the library part. Each application (snmpget, snmpbulkget, etc.) required about 50 additional lines of code to offer the new compression as an option. The user can opt for compression or run their application without any compression.

Table 4 Comparing response times of the original code with the OID compression enabled code for SNMP Get Command.

No	Scalar	Tabular	Original code (T_o)	Comp not opted (T_n)	Comp opted (T_c)	Improvement (1) T_o/T_c	Improvement (2) T_n/T_c
1	5	0	1287	1117	681	1.89	1.64
2	25	0	335	341	249	1.35	1.37
3	30	0	1161	1152	764	1.52	1.51
4	60	0	1421	1353	879	1.62	1.54
5	90	0	2738	1393	1022	2.68	1.36
6	128	0	2282	2016	1594	1.43	1.26
7	0	30	6863	6559	6299	1.09	1.04
8	0	60	66074	64586	67076	0.99	0.96
9	0	90	135216	124415	122092	1.11	1.02
10	0	128	175059	168245	163413	1.07	1.03
11	15	15	1166	1162	789	1.48	1.47
12	30	30	7696	7480	7387	1.04	1.01
13	45	45	36227	37637	38038	0.95	0.99
14	60	68	65359	72460	67700	0.97	1.07

2. Image Size: All DLLs, applications, and daemons retain the same size with compression. This is expected because compression is a part of the dynamic SNMP library.
3. Development Effort: About 3 programmer months to develop and test.
4. Code Complexity: Encoding of the compressed object identifier takes $O(n)$ time. This is identical to the current complexity seen in Net-SNMP. However, there is some additional logic required to find offset, length, and suffix/embedded. This is done by comparing anchor's subids against the object's subids in $O(n)$ time, where n is the number of sub-identifiers in an object identifier. The extra effort required to decode and use suffix/embedded identifier is similar to or less than handling a full blown object ID as there is no need to iterate over all redundant sub-identifiers. The extra effort required encoding to find offset and length is compensated to some extent by decoding effort. This results in a response time comparable to the non-compressed OID case. When the more objects are packed in the same PDU, it results in additional savings of CPU cycles at the encoding end as suggested in Hari et al., (2010).
5. Testing Effort: Full-fledged testing has not been performed yet. Code is exercised for typical requests and performance studies.
6. Execution and Response Time: The following test setup is used to compare the response time and execution time, with and without compression.

- Both Net-SNMP agents (the Net-SNMP agent with no compression feature and the Net-SNMP agent with compression feature) are hosted on a Windows XP desktop.
- The modified SNMP agent with compression is assigned the server port 161 and the agent with no compression feature is assigned the server port 1161.
- Client applications with and without compression feature are hosted on the same desktop.
- All other user applications are terminated to reduce any contention to the CPU.
- MS Loop back adapter is setup to capture client–server interaction. This is done to eliminate any network delay variations. Thus, the response time is a good indication of the execution time for both the client and agent side combined.
- Response times are measured using a Tcl script with our packet capture Tcl extension (Hari et al., 2007)
- Tcl script executes each request 10 times and computes the average response time
- Get and Getbulk with multiple numbers of scalar and tabular objects are used in these measurements.
- The times listed are in microseconds.
- Table 4 compares the response times for the Get command. The response time for a Get command with original code (without compression feature) is denoted by To; response time with our compression feature turned on is denoted by Tc; response time with our compression feature turned off is denoted by Tn. These times are computed from the time found from the packet capture—request send time

Table 5 Comparing response times with the compression option is turned off and turned on for Get Bulk Command.

No	Scalar	Tabular	Repetition	Original code (T_o)	Comp not opted (T_n)	Comp opted (T_c)	Improvement (1) T_o/T_c	Improvement (2) T_n/T_c
1	0	5	3	1189	1232	616	1.93	2.00
2	0	4	6	1560	1418	1275	1.22	1.11
3	0	10	3	2626	2743	777	3.38	3.53
4	0	10	6	3694	3552	1956	1.89	1.82
5	0	10	9	1643	1632	445	3.69	3.67
6	0	10	10	54403	55334	52953	1.03	1.04
7	2	1	98	28355	21084	21132	1.34	1.00

and response arrival time. Each row compares response times for a Get with a certain number of scalar and tabular objects. The number of scalar and tabular objects carried in the Get command is identified by columns 2 and 3, respectively. The last two columns list the performance improvement with respect to To and Tn, respectively. Most of the cases show a positive improvement and wherever there is no improvement, the difference in response time is relatively small. All the objects used in these test cases are standard MIB-2 objects with some object duplication.

- Table 5 provides similar comparisons for the GetBulk command. The repetition factor lists the number of rows of tabular objects requested. As expected, Getbulk provides a better performance improvement in comparison to the Get command.

7. The byte-savings due to compression are not presented in this paper. This has been done extensively in our earlier paper (Hari et al., 2010). We did observe IP fragmentation for certain requests listed in Table 4 and Table 5. Our response time calculation has been done based on the arrival time of the last of these fragmented PDUs.
8. Windows services were running when we did these performance measurements. These services could have influenced some of our measurements. To reduce this effect, we ran each test case a few times and ascertained that their variations were low. The improvements listed in the last two columns indicate an overall better performance with OID compression. Absolute values in these columns are not given any importance.

6. Conclusion and future work

The results listed in Section 4 suggest that OID compression is a feasible and viable feature due to its low overhead. We expected to see some improvements in overall response times due to reduction in payload size, transmission time, and processing time. This expectation is confirmed with our results. The effort required for testing is non-trivial; however, it is a one-time activity with subsequent regression testing. There are two aspects to using OID compression—viability of its implementation and the suitability of a given MIB for compression. This paper dealt with the former. We are currently in the process of analyzing various vendor MIBs to find out the latter. About 100 of the 260 MIBs analyzed include 80% or more table objects with average object depth of individual MIBs varying from 9 to 15; about 165 of the 260 MIBs have 60% or more table objects with the average object depth of individual MIBs varying from 9 to 15. These MIBs are highly likely to be conducive for OID compression. However, there are other aspects (key size, number of fixed size objects in a table/group, etc.) to be investigated as well before one can decide whether to use OID compression for a MIB. We are developing some tools and reports in this area which we plan to share in a suitable forum in the future. We are also developing a C library to decode compressed OID varbinds (Premini and Narayanan, 2012). This can be used in frame capture applications like Wireshark.

References

Aseem, Sethi, 2008. Efficient representation of System Network Management Objects Identifiers. US Patent Application: #7343404.

Alnuem, Mohammed A., 2010. An extended review of techniques for enhancing TCP performance, in Elsevier. Journal of King Saud University Computer & Information Sciences 22, 45–61.

Frye, R., Levi, D., Routhier, S., Wijnen, B., 2003. Coexistence between Version 1, Version 2, and Version 3 of the Internet-standard network management framework, RFC 3584.

Ghirardi, Maurizio, 2010. Transferring of SNMP Messages over UDP with Compression of Periodically Repeating Sequence. US Patent Application #20100306414.

Hari, T.S., Narayanan, et al., 2007. A Tcl Language Extension for Accessing and Transmitting Data Link Layer Frames, Tcl 2007 Conference, Chicago, IL.

Hari, T.S., Narayanan, L.S., Prakash Raj, Vasumathi Narayanan, A., 2010. Study on the Effectiveness of SNMP OID Compression, in Springer, Journal of Network and Systems Management 19(4), ISSN 1064–7570.

Hari, T.S., Narayanan, Sumitra Narayanan, A Method For Efficient Encoding /Decoding Of Data Identifiers In Hierarchical Data Models, A Patent filed with the Official Journal of the Patent Office, India, Application No.3782/CHE/2011 Date of filing:04/11/2011 (43) Publication Date: 18/11/2011, International Classification: H04Q.

Internet Assigned Number Authority (IANA): http://www.iana.org/.

ITU-T Recommendation X.690, 2003. "OSI networking and system aspects—Abstract Syntax Notation One (ASN.1)–ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)".

Levi, D., Meyer, P., Stewart, B., 2002. SNMP applications, RFC 3413.

McCloghrie, K., Perkins, D., Schoenwaelder, J., 1999. Structure of Management Information Version 2 (SMIv2), RFC 2578.

McLeod, S., Partain, D., White, M., 2001. SNMP object identifier compression, draft-ietf-eosoidcompression-00.txt.

Premini, Francis, Narayanan, Hari T.S., 2012. Enhancing "Wireshark for Seamless Decoding of Compressed and Non-Compressed SNMP OIDs". In: Proceedings of International Conference On Modelling, Optimisation And Computing (ICMOC-2012), Elsevier.

Presuhn, R., 2002. Version 2 of SNMP protocol operations, RFC 3416.

Presuhn, R., 2002. Transport mappings, RFC 3417.

Presuhn, R., 2002. Management Information Base (MIB) for the Simple Network Management Protocol (SNMP), RFC 3418.

Schoenwaelder, J., 2001. SNMP payload compression, draft-ietf-nmrg-snmpcompression-01.txt.

William, Stallings., 1998. "SNMP, SNMPv2, SNMPv3, and RMON 1 and 2, 3rd edition".