SNMP Trace Analysis Definitions

Gijs van den Broek[1], Jürgen Schönwälder[2], Aiko Pras[1], and Matúš Harvan[3]

[1] Computer Science, University of Twente, Netherlands
j.g.vandenbroek@student.utwente.nl,
a.pras@utwente.nl
[2] Computer Science, Jacobs University Bremen, Germany
j.schoenwaelder@jacobs-university.de
[3] Computer Science, ETH Zurich, Switzerland
mharvan@inf.ethz.ch

Abstract. The Network Management Research Group (NMRG) started an activity to collect traces of the Simple Network Management Protocol (SNMP) from operational networks. To analyze these traces, it is necessary to split potentially large traces into more manageable pieces that make it easier to deal with large data sets and simplify the analysis of the data. This document introduces some common definitions that have been found useful for implementing tools to support trace analysis.

Keywords: simple network management protocol, traffic modeling.

1 Introduction

The Simple Network Management Protocol (SMMP) was introduced in the late 1980s. Since then, several evolutionary protocol changes have taken place, resulting in the SNMP version 3 framework (SNMPv3), published as full standard in 2002 [1,2]. Extensive use of SNMP has led to significant practical experience by both network operators and researchers. However, up until now only little research has been done on characterizing and modeling SNMP traffic.

Since recently, network researchers are in the possession of network traces, including SNMP traces, captured on operational networks. The availability of SNMP traces enables research on characterizing and modeling real world SNMP traffic. Experience with SNMP traces has shown that traces must be large enough in order to make proper observations. A more detailed motivation for collecting SNMP traces and guidelines how to capture SNMP traces can be found in [3].

The analysis of large SNMP traces can take a large amount of processing time. Therefore, it is often desirable to focus the analysis on smaller, relevant sections of a trace. This in turn requires a proper way to identify these smaller sections of a trace. This document describes a number of identifiable sections within a trace which make specific research on these smaller sections more practical.

The rest of the paper is structured as follows. An overview of the definitions is given in the next section and subsequent sections define messages, traces, flows, slices, slice prefixes, and slice types. Related and future work is discussed before the paper concludes.

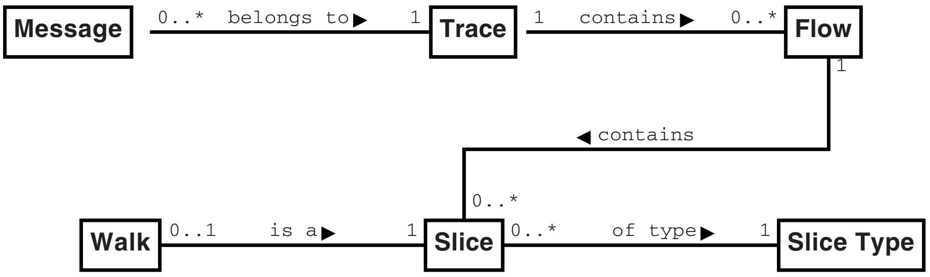

Fig. 1. Relationship between messages, traces, flows, slices and slice types

2 Overview

Fig. 1 shows the various entities associated with an SNMP trace and how they relate to each other.

The most central entity in Fig. 1 is an SNMP trace, consisting of a potentially large set of SNMP messages. An SNMP trace is the result of recording SNMP traffic on a specific network for a specific time duration. Such a trace may, depending on the number of hosts in the respective network, contain SNMP messages exchanged between possibly many different SNMP engines. The messages contained in a trace may be represented in different formats. For the purpose of this document, the simple comma separated values (CSV) format defined in [3] contains sufficient information to split a trace into smaller sections.

The SNMP messages belonging to an SNMP trace may have been exchanged between many different SNMP engines running on different hosts. Therefore, a first obvious way of separating a trace into smaller sets of SNMP messages is the separation of a trace into flows. Each flow contains only those SNMP messages of an SNMP trace that have been exchanged between two network layer endpoints. Such a separation may be necessary in case one wants to analyze specific SNMP traffic characteristics (e.g., number of agents managed by a management station) and wants to rule out network endpoint specific behaviour (e.g., different SNMP management stations may have different polling configurations).

Flows within traces can still be quite large in terms of the number of messages they contain. Therefore, it may be necessary to split a flow into even smaller sections called slices. A slice contains all SNMP messages of a given flow that are related to each other in time and referenced information. Splitting a flow into slices makes it possible to separate SNMP messages within traces that belong to each other.

For example, a slice may contain the SNMP messages exchanged between an agent and a manager, which polls that agent in a single polling instance. The manager may be configured to poll that agent every once in a while. If the requested information from the agent remains unchanged, then the respective slices of SNMP traffic occurring between this manager and agent will be highly comparable. In such a case the slices will be of the same slice type. Similar slices will thus be considered of the same slice type and incomparable slices will not be of the same slice type.

Besides the fact that each slice is of specific slice type, slices can also be of a specific form with respect to the messages encompassing a slice. For example, slices containing a sequence of linked GetNext or GetBulk requests are commonly called an SNMP walk. Note that only a subset of all slices will be walks.

3 Messages

SNMP messages carry protocol data units (PDUs) realizing a small set of well defined protocol operations [4]. The PDUs can be used to classify SNMP messages.

Notation 1. *The properties of an SNMP message M are denoted as follows:*

$M.type$	= *operation type of message M (get, getnext, ...)*
$M.class$	= *class of message M (according to RFC 3411)*
$M.tsrc$	= *transport layer source endpoint of message M*
$M.tdst$	= *transport layer destination endpoint of message M*
$M.nsrc$	= *network layer source endpoint of message M*
$M.ndst$	= *network layer destination endpoint of message M*
$M.reqid$	= *request identifier of message M*
$M.time$	= *capture timestamp of message M*
$M.oids$	= *OIDs listed in varbind list of message M*
$M.values$	= *values listed in varbind list of message M*

These properties of an SNMP message can be easily extracted from the exchange formats defined in [3].

Definition 1. *This definition establishes the following message classes:*

1. *A* ***read request message*** *is a message M containing a PDU of type GetRequest, GetNextRequest, or GetBulkRequest.*
2. *A* ***write request message*** *is a message M containing a PDU of type SetRequest.*
3. *A* ***notification request message*** *is a message M containing a PDU of type InformRequest.*
4. *A* ***notification message*** *is a message M containing a PDU of type Trap or InformRequest.*
5. *A* ***request message*** *is a message M which is either a read request message, a write request message, or a notification request message.*
6. *A* ***response message*** *is a message M containing a PDU of type Response or of type Report.*
7. *A* ***non-response message*** *is a message M which is either a read request message, a write request message, or a notification message.*
8. *A* ***command message*** *is a message M which is either a read request message or a write request message.*

Report messages are treated like Response messages since the SNMPv3 specifications currently use Report messages only as an error reporting mechanism, always triggered by the processing of some request messages. In case future SNMP versions or extensions use Report messages without having a request triggering the generation of Report messages, we may have to revisit the definition above.

Definition 2. *A set of* ***command group messages*** *consists of all messages M satisfying either of the following two conditions:*

1. *M is a command message*
2. *M is a response message and there exists a command message C such that the following holds:*

$$\begin{aligned} M.reqid &= C.reqid \\ M.tdst &= C.tsrc \\ M.tsrc &= C.tdst \\ (M.time - C.time) &< t \end{aligned}$$

The parameter t defines a maximum timeout for response messages.

This definition requires that the response message originates from the transport endpoint over which the request message has been received. This is not strictly required by SNMP transport mappings and in particular the UDP transport mapping allows to send responses from different transport endpoints. While sending response messages from a different transport endpoint is legal, it is also considered bad practice causing interoperability problems, since some management systems do not accept such messages.

It was decided to require matching transport endpoints since doing so significantly simplifies the procedures below and avoids accidentally confusing requests and responses. Implementations responding from different transport endpoints will lead to (a) a larger number of requests without related responses (and likely no retries) and (b) a similarly large number of responses without a matching request. If such behavior can be detected, the traces should be investigated and if needed the transport endpoints corrected. The requirement for matching transport endpoints only affects request / response pairs. It is perfectly fine for a manager to use different transport layer endpoints in different polling instances, or even different operations (i.e., slices) within the same polling instance.

Definition 3. *A set of* ***notification group messages*** *consists of all messages M satisfying either of the following two conditions:*

1. *M is a notification message*
2. *M is a response message and there exists a notification request message N such that the following holds:*

$$\begin{aligned} M.reqid &= N.reqid \\ M.tdst &= N.tsrc \\ M.tsrc &= N.tdst \\ (M.time - N.time) &< t \end{aligned}$$

The parameter t defines a maximum timeout for response messages.

This definition again requires matching transport endpoints for notification group messages.

4 Traces and Flows

Traces are (large) sets of SNMP messages that are the result of recording SNMP traffic using a single traffic recording unit (e.g., using tcpdump) on a network segment carrying traffic of one or more managers and agents. Traces being used in the remainder of this document may be altered as a result of anonymization, which may result in some message information loss.

Traces may contain SNMP messages that have been exchanged between possibly many different network layer endpoints. One way of making an initial separation of such a trace into more manageable pieces is by splitting the messages into flows. Each flow contains only messages that have occurred between two network layer endpoints.

4.1 Trace and Flow Definition

Definition 4. *An SNMP* ***trace*** *(or short "trace") T is an ordered set of zero or more SNMP messages M. All messages M in T are chronologically ordered according to the capture timestamp $M.time$.*

Definition 5. *A* ***flow*** *F is the set of messages of an SNMP trace T with the following properties:*

1. *All response messages originate from a single network endpoint.*
2. *All non-response messages originate from a single network endpoint.*
3. *All messages are either command group messages with parameter t or notification group messages with parameter t.*

Parameter t defines the maximum timeout for response messages. The value of t should be chosen such that only response messages to the respective non-response messages are considered part of the same flow. Analysis of a large number of traces shows that 25 seconds is a proper default value for t.

It is possible that response messages of a trace cannot be classified to belong to any flow. This can happen if request messages triggering the response messages were not recorded (for example due to asymmetric routing), or because response messages were originating from transport endpoints different from the endpoint used to receive the associated request message.

This definition of a flow indicates that it can be either unidirectional (e.g., a manager sending non-response messages to a non-responding agent), or bidirectional (e.g., a manager reading a table from an agent). This is different from other flow definitions, like the NetFlow definition [5]. The flow definition is mostly consistent with the definition of an SNMP flow used in [6]. The difference is that the tool used to generate the data reported in [6] did only require that the network layer source endpoint of the response messages matches the destination network layer endpoint of the associated request messages.

Definition 6. *A **flow initiator** is the network layer endpoint of the two endpoints involved in a flow, which is responsible for sending the first non-response message.*

Notation 2. *The properties of a flow F are denoted as follows:*

$F.type$	= *type of the flow F (command/notification)*
$F.nsrc$	= *network layer source endpoint of F*
$F.ndst$	= *network layer destination endpoint of F*
$F.start$	= *timestamp of the first message in F*
$F.end$	= *timestamp of the last message in F*
$F.init$	= *initiator of the flow F*
$F.t$	= *parameter t of F (maximum timeout for response messages)*

Subsequently, flows containing only command group messages are called command flows. Similarly, flows containing only notification group messages are called notification flows.

4.2 Trace and Flow Example

Table 1 shows an example of a trace consisting of SNMP messages that were exchanged between different network layer endpoints. The network layer endpoints are represented by A, B, C and D.

The first flow F_1 consists of SNMP messages that have been exchanged between network layer endpoints A and B, where all response messages originate from B and all non-response messages originate from A. The minimum value of parameter t for this flow is 0.07 seconds, since that is the longest time between a request and its subsequent response message.

The second flow F_2 contains SNMP messages exchanged between network layer endpoints C and D, where all response messages originate from D and all non-response messages originate from C. The minimum value of parameter t for this flow is 0.04 seconds.

The third flow F_3 contains the remaining SNMP messages of the trace that occurred between network layer endpoints C and D. In this case the non-response message originates from D. There is no parameter t applicable to this flow, because there are no response messages.

Table 1. Example trace containing two flows

Message	Time [s]	Direction	Type	Reqid	Flow
0	0.00	$A \rightarrow B$	GetNext	1	F_1
1	0.04	$C \rightarrow D$	Get	10	F_2
2	0.05	$B \rightarrow A$	Response	1	F_1
3	0.08	$D \rightarrow C$	Response	10	F_2
4	0.11	$A \rightarrow B$	GetNext	2	F_1
5	0.15	$B \rightarrow A$	Response	2	F_1
6	0.18	$A \rightarrow B$	GetNext	3	F_1
7	0.22	$D \rightarrow C$	Trap	14	F_3
8	0.25	$B \rightarrow A$	Response	3	F_1

5 Slices

Flows can still contain a large amount of SNMP messages. A flow should therefore be split up into even smaller sets of messages. One way of identifying meaningful subsets of messages of a flow would be by considering the behavior of managers and agents. In the case of managers, they are usually configured to perform regular polling instances. In such a polling instance, the manager might poll a number of agents. Since a flow contains only the messages exchanged between two network layer endpoints, a flow therefore probably consists of only a subset of the messages that are part of a polling instance. So, one option of finding smaller, meaningful subsets of messages within flows, would be by looking for messages that belong to a particular polling instance. Such a smaller set of messages is called a slice.

5.1 Slice Definition

Definition 7. *A **slice** S with parameter e is a subset of messages in a flow F for which the following properties hold:*

1. *All messages are exchanged between the same two transport endpoints (a single transport endpoint pair).*
2. *All non-response messages must have a PDU of the same type.*
3. *All messages with a PDU of type Get, Set, Trap, or Inform must contain the same set of OIDs.*
4. *Each GetNext or GetBulk message must either contain the same set of OIDs as the preceding request or it must be linked to a response of the last previously answered request (i.e., the request must contain at least one OID that has been contained in the (repeater) varbind list of a preceding response message of the last answered request message).*
5. *All Response messages must follow a previous request message that is part of the same slice.*
6. *For any two subsequent non-response messages $Q1$ and $Q2$ with $Q1.time < Q2.time$, the following condition must hold:*

$$(Q2.time - Q1.time) < e$$

The first item in the slice definition requires that the messages of a single slice are exchanged between a single transport layer endpoint pair. This is different from the flow definition, which requires a single network layer endpoint pair. The choice of looking at the transport layer endpoints in the case of slices is based on the assumption that, for instance, multiple managers and agents might be operating from the same respective network layer endpoint. Another assumption is that a manager and an agent will only use a single transport layer endpoint respectively when they communicate for the duration of a slice (or even a polling instance). A previous section already mentioned some issues when a manager or agent uses different transport layer endpoints within a single polling instance.

The parameter e defines the maximum time between two non-response messages that belong to a slice. This parameter should be chosen such that unrelated non-response messages within a flow are not considered to be of the same slice. Unrelated non-response messages are those that, for instance, belong to different polling instances. The parameter e should therefore be larger than the retransmission interval in order to keep retransmissions within a slice and smaller than the polling interval used by the slice initiator.

The value of parameter e might be closely related to parameter t of the respective flow the slice is part of. For instance, if parameter e is very large, than t is also likely to be very large and vice versa. Also, if parameter e is very small, then t is probably also very small. However, it is not possible to strictly state that e and t are always closely related to each other, because parameter e is specific for a slice. This is in contrast with parameter t which is specific for a much larger set of messages, a flow.

Definition 8. *A* ***slice initiator*** *is one of the two transport layer endpoints involved in a slice, which is responsible for sending the chronologically first non-response message.*

Notation 3. *The properties of a slice S are denoted as follows:*

$S.type$	=	*type of non-response messages in S*
$S.tsrc$	=	*transport layer endpoint of initiator of S*
$S.tdst$	=	*transport layer endpoint of non-initiator of S*
$S.start$	=	*timestamp of the chronologically first message in S*
$S.end$	=	*timestamp of the chronologically last message in S*
$S.init$	=	*initiator of slice S*
$S.e$	=	*parameter e of S (maximum time between two non-response messages)*

5.2 Slice Example

Table 2 shows an example of a flow containing messages exchanged between transport layer endpoints A, B, and C. Considering the timing of the messages, a proper value of e should be $0.14 \leq e \leq 299.82$ seconds. Such a value for parameter e will separate the flow into two apparent polling instances, which each contain the same set of messages.

The first slice S_1 consists of a Get request and its subsequent response. A similar request is recorded later in slice S_3 but since we assume e as discussed above, the slices S_1 and S_3 are distinct. The second slice S_2 also contains a Get request and its subsequent response. This slice is different from S_1 since a different OID is requested. The slice S_4 consists of a Set request that has not been answered. (A potential reason is that the SNMP engine listening on the transport layer endpoint C did not grant write access and dropped the message.)

The last slice S_5 contains a sequence of linked GetNext requests. The GetNext request message 10 is likely a retransmission of the GetNext request message 9. This example demonstrates that retransmissions are recorded in the slice that contains the original request.

Table 2. Example flow containing three slices

Message	Time [s]	Direction	Type	Reqid	OIDs	Slice
0	0.00	$A \rightarrow B$	Get	1	alpha.1	S_1
1	0.06	$B \rightarrow A$	Response	1	alpha.1	S_1
2	0.12	$A \rightarrow B$	Get	2	beta.1	S_2
3	0.17	$B \rightarrow A$	Response	2	beta.1	S_2
4	300.00	$A \rightarrow B$	Get	3	alpha.1	S_3
5	300.05	$B \rightarrow C$	Set	4	gamma.1	S_4
6	300.07	$B \rightarrow A$	Response	3	alpha.1	S_3
7	300.14	$A \rightarrow B$	GetNext	5	beta	S_5
8	300.19	$B \rightarrow A$	Response	5	beta.1	S_5
9	300.32	$A \rightarrow B$	GetNext	6	beta.1	S_5
10	300.52	$A \rightarrow B$	GetNext	7	beta.1	S_5
11	300.58	$B \rightarrow A$	Response	7	delta.1	S_5

6 Slice Signature and Prefix

As noted in the beginning of this document, it is desirable that slices can be tested for equality/comparability. This is where the slice prefix comes in. The slice prefix provides one of the means to compare slices. Using the slice prefix and a few other parameters of a number of slices, one can determine which slices should be considered "equal" and which of them are incomparable. This will assist in the process of finding potentially other relations.

The slice prefix is a set of OIDs. This set is constructed from the messages that make up a single slice. So, for example, a slice that is the result of a manager requesting the contents of a particular table (with OID alpha) on an agent using a simple single varbind GetNext walk, starting at the table OID alpha, shall yield a slice prefix which consists of the OID alpha.

Because the aim is to compare various slices using the slice prefix (along some other characteristics of a slice), this implicitly suggests the need to know whether a number of slices are the result of the same behaviour (i.e., specific configuration) of the initiating party of these slices. For example, one may want to know whether a number of slices that involve a single manager and a single agent were the result of just one specific configuration of that manager. Multiple slices, that may all be initiated by that same manager and each slice possibly occurred in different polling instances, may in fact be the result of the same specific configuration of that particular manager. So, since in this case the specific configuration of the manager is only relevant for determining the behaviour, the slice prefix should be constructed based on OIDs in messages originating from that manager only. More generally, only the messages within slices that are sent by the initiating party (the non-response messages) are considered for the determination of the respective slice prefix of a slice.

6.1 Slice Signature and Prefix Definition

Definition 9. *A **slice signature** S.sig of a slice S is a set of OIDs derived from the OIDs contained in the non-response messages of a slice. Let $r(S)$ denote the set of response messages of slice S and $n(S)$ the set of non-response messages of S. Then the set S.sig consists of the following OIDs:*

$$S.sig = \begin{cases} \bigcup_{n \in n(S)} (n.oids) \setminus \bigcup_{r \in r(S)} (r.oid) & S.type \text{ is GetNext or GetBulk} \\ \bigcup_{n \in n(S)} (n.oids) & \text{otherwise} \end{cases}$$

The slice signature summarizes which OIDs have been carried in a slice and is straightforward to compute. However, there are situations in some GetNext or GetBulk sequences where the signature might contain some unwanted OIDs as will be demonstrated by an example below.

To further condense signatures, it is necessary to introduce a prefix relationship between OIDs. This prefix relationship can then be used to reduce a slice signature to a slice prefix.

Definition 10. *An OID $a = a_1.a_2...a_n$ is a **prefix** of OID $b = b_1.b_2...b_m$ if and only if $n < m$ and $a_i = b_i$ for $1 \leq i \leq n$.*

Definition 11. *The **slice prefix** S.slice is the set of all OIDs o in S.sig for which there is no p in S.sig such that p is a prefix of o.*

6.2 Slice Signature and Prefix Example

The following example demonstrates how a slice prefix is determined. Consider the case that a single manager A is set to poll a specific agent B. Manager A is programmed to retrieve some values from B. A single slice may contain the messages shown in Table 3.

The slice shown in Table 3 has a number of interesting properties. First, not all columns in the retrieved table have an equal length. Second, the manager

Table 3. Example slice for calculating a slice prefix

Message	Direction	Type	OIDs
0	$A \rightarrow B$	GetNext	alpha, beta
1	$B \rightarrow A$	Response	alpha.1, beta.1
2	$A \rightarrow B$	GetNext	alpha.1, beta.1
3	$B \rightarrow A$	Response	alpha.2, beta.3
4	$A \rightarrow B$	GetNext	beta.2, alpha.2, sysUpTime
5	$B \rightarrow A$	Response	beta.3, alpha.3, sysUpTime.0
6	$A \rightarrow B$	GetNext	beta.3, alpha.3
7	$B \rightarrow A$	Response	gamma.1, alpha.4
8	$A \rightarrow B$	GetNext	alpha.4
9	$B \rightarrow A$	Response	delta.1

is set to request the sysUpTime on an irregular basis (i.e., every few requests). Third, the manager attempts to fill "holes" in the table and finally the order of referenced OIDs in GetNext messages changes.

All of these properties do not influence the process for determining the slice signature. The slice prefix is constructed as follows:

1. The union of all OIDs in non-response messages is the following set:

$$\begin{aligned} N = \{ \; & alpha, beta, alpha.1, beta.1, beta.2, alpha.2, \\ & sysUpTime, beta.3, alpha.3, alpha.4 \; \} \end{aligned}$$

2. The union of the OIDs in response messages is the following set:

$$\begin{aligned} R = \{ \; & alpha.1, beta.1, alpha.2, beta.3, alpha.3, \\ & sysUpTime.0, gamma.1, alpha.4, delta.1 \; \} \end{aligned}$$

3. Subtracting the two sets results in the slice signature $S.sig$:

$$S.sig = N - R = \{ \; alpha, beta, beta.2, sysUpTime \; \}$$

   The element beta.2 exists, because the manager was trying to fill a "hole" in a table. Since these "holes" reside in the tables on the agent side and may change dynamically, they do not really help in describing the behavior of the initiating party.
4. Since beta is a prefix of beta.2, the slice prefix becomes the following set:

$$S.prefix = \{ \; alpha, beta, sysUpTime \; \}$$

   The slice prefix does not include beta.2 anymore and thus a manager retrieving the same columns alpha and beta with and without "holes" will produce slices with the same slice prefix.

7 Slice Types

As described previously, the slice type allows for comparing slices. This means that any number of slices that are of the same slice type may be considered an equivalence class and may therefore be considered to be the result of the same behaviour of the slice initiator.

7.1 Slice Type Definition

Definition 12. *Two slices A and B satisfy the binary* ***slice equivalence*** *relation $A \sim B$ if the following properties hold:*

1. *All messages in A and B have been exchanged between the same network layer endpoints.*
2. *All read request messages, write request messages, and notification messages in A and B originate from the same network layer endpoint.*

3. *All non-response messages in A and B are of the same type.*
4. *The slices A and B have the same prefix, that is $A.prefix = B.prefix$.*

It can be easily seen that the relation $\sim$ is reflexive, symmetric, and transitive and thus forms an equivalence relation between slices.

Definition 13. *Let S be a set of slices, then all slices in the equivalence class*

$$[A] = \{s \in S | s \sim A\}$$

with $A \in S$, are of the same ***slice type****.*

7.2 Slice Type Example

The flow shown in Table 4 contains two slices. The first slice S_1 contains messages that have been exchanged between transport layer endpoints A and B while the second slice S_2 contains messages that have been exchanged between transport layer endpoints C and D. However, the network layer endpoints of this slice are the same as the first slice and all non-response messages in both slices originate from the same network layer endpoint.

Table 4. Example for slice equivalence and slice types

Message	Direction	Type	OIDs	Slice
0	$A \rightarrow B$	GetNext	alpha, beta	S_1
1	$B \rightarrow A$	Response	alpha.1, beta.1	S_1
2	$A \rightarrow B$	GetNext	alpha.1, beta.1	S_1
3	$B \rightarrow A$	Response	alpha.2, beta.2	S_1
4	$A \rightarrow B$	GetNext	alpha.2, beta.2	S_1
5	$B \rightarrow A$	Response	gamma.1, delta.1	S_1
6	$C \rightarrow D$	GetNext	alpha, beta	S_2
7	$D \rightarrow C$	Response	alpha.1, beta.1	S_2
8	$C \rightarrow D$	GetNext	alpha.1, beta.1	S_2
9	$D \rightarrow C$	Response	gamme.1, delta.1	S_2

As can be verified easily, the slices S_1 and S_2 satisfy the slice equivalence relationship, that is $S_1 \sim S_2$ and they form an equivalence class under $\sim$, which we call a slice type.

8 Related and Future Work

The performance of SNMP has been the subject of several studies. Some papers such as [7,8] compare the performance of centralized SNMP management to distributed management approaches while other papers compare the performance of the SNMP protocol with middleware systems such as CORBA or Web Services [9,10,11]. The impact of security protocols on SNMP performance has been studied in [12,13,14]. Some authors have formulated models for the SNMP protocol [15,16].

All these studies have in common that they make assumptions how SNMP is used in practice, due to a lack of commonly accepted models how SNMP is used in practice. To address this issue, some researchers active in the Network Management Research Group (NMRG) of the Internet Research Task Force (IRTF) started an effort to collect traces from operational networks and to build the necessary tools to analyze them [3]. First results were published in [6] and it became clear that precise definitions of basic concepts, such as those provided in this paper, are needed in order to produce meaningful and comparable results. The authors are currently using and extending these definitions in order to study the periodicity of SNMP traffic [17] and table retrieval algorithms.

9 Conclusions

The analysis of SNMP traces requires a collection of common and precise definitions in order to establish a basis for producing meaningful and comparable results. This paper provides such a collection of basic definitions that have been developed over time and are also being reviewed and discussed within the NMRG of the IRTF. This paper is a condensed summary of a more detailed document [18] submitted to the NMRG and written to provide an early stable reference while the work in the NMRG continues and to foster discussion within the broader network management research community.

Acknowledgement

The work reported in this paper is supported by the EC IST-EMANICS Network of Excellence (#26854).

References

1. Case, J., Mundy, R., Partain, D., Stewart, B.: Introduction and Applicability Statements for Internet Standard Management Framework. RFC 3410, SNMP Research, Network Associates Laboratories, Ericsson (December 2002)
2. Harrington, D., Presuhn, R., Wijnen, B.: An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks. RFC 3411, Enterasys Networks, BMC Software, Lucent Technologies (December 2002)
3. Schönwälder, J.: SNMP Traffic Measurements and Trace Exchange Formats. Internet Draft (work in progress) <draft-irtf-nmrg-snmp-measure-04.txt>, Jacobs University Bremen (March 2008)
4. Presuhn, R.: Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP). RFC 3416, BMC Software (December 2002)
5. Claise, B.: Cisco Systems NetFlow Services Export Version 9. RFC 3954, Cisco Systems (October 2004)
6. Schönwälder, J., Pras, A., Harvan, M., Schippers, J., van de Meent, R.: SNMP Traffic Analysis: Approaches, Tools, and First Results. In: Proc. 10th IFIP/IEEE International Symposium on Integrated Network Management (May 2007)

7. Zapf, M., Herrmann, K., Geihs, K.: Decentralized SNMP Management with Mobile Agents. In: Proc. 6th IFIP/IEEE International Symposium on Integrated Network Management, Boston, May 1999, pp. 623–635 (1999)
8. Fuggetta, A., Picco, G., Vigna, G.: Understanding Code Mobility. IEEE Transactions on Software Engineering 24(5), 342–361 (1998)
9. Gu, Q., Marshall, A.: Network Management Performance Analysis and Scalability Tests: SNMP vs. CORBA. In: Proc. 2004 IEEE/IFIP Network Operations and Management Symposium, Seoul, April 2004, pp. 701–714 (2004)
10. Pras, A., Drevers, T., van de Meent, R., Quartel, D.: Comparing the Performance of SNMP and Web Services based Management. IEEE Transactions on Network and Service Management 1(2) (November 2004)
11. Pavlou, G., Flegkas, P., Gouveris, S., Liotta, A.: On Management Technologies and the Potential of Web Services. IEEE Communications Magazine 42(7), 58–66 (2004)
12. Du, X., Shayman, M., Rozenblit, M.: Implementation and Performance Analysis of SNMP on a TLS/TCP Base. In: Proc. 7th IFIP/IEEE International Symposium on Integrated Network Management, Seattle, May 2001, pp. 453–466 (2001)
13. Corrente, A., Tura, L.: Security Performance Analysis of SNMPv3 with Respect to SNMPv2c. In: Proc. 2004 IEEE/IFIP Network Operations and Management Symposium, Seoul, April 2004, pp. 729–742 (2004)
14. Marinov, V., Schönwälder, J.: Performance Analysis of SNMP over SSH. In: State, R., van der Meer, S., O'Sullivan, D., Pfeifer, T. (eds.) DSOM 2006. LNCS, vol. 4269, pp. 25–36. Springer, Heidelberg (2006)
15. Pattinson, C.: A study of the behaviour of the simple network management protocol. In: Proc. 12th IFIP/IEEE Workshop on Distributed Systems: Operations and Management, Nancy (October 2001)
16. Chen, T., Liu, S.: A Model and Evaluation of Distributed Network Management Approaches. IEEE Journal on Selected Areas in Communications 20(4), 850–857 (2002)
17. van den Broek, J.G.: Periodicity of SNMP Traffic. BSc Thesis (August 2007)
18. van den Broek, J.G., Schönwälder, J., Pras, A., Harvan, M.: SNMP Trace Analysis Definitions. Internet Draft (work in progress) <draft-schoenw-nmrg-snmp-trace-definitions-02.txt>, University of Twente, Jacobs University Bremen, ETH Zurich (April 2008)