SIMPLE NETWORK MANAGEMENT PROTOCOL TRACE ANALYSIS

Mater Thesis

School of Engineering and Science

Jacobs University Bremen

May 2007

Catalin Ciocov

Review Committee: J. Schönwälder, Jacobs University
A. Pras, University of Twente

Abstract

Although the SNMP protocol has been around since the late 80s and it is widely deployed and used in today's networks, there are still a lot of uncertainties about the actual usage of it in production environments. By analysing SNMP traffic traces collected in both large scale and small networks we try to characterize the current usage of the protocol. We propose several analysis methods that aim at extracting relevant information for SNMP developers, network vendors and other researchers.

TABLE OF CONTENTS

1 INTRODUCTION

The Simple Network Management Protocol (SNMP) is an application layer protocol, used to monitor and configure network devices (e.g., routers, switches). The protocol was introduced in late 1980s and has evolved since then to version 3 (SNMPv3). [20]

Although widely deployed, it is still unclear what the dominant implementation features are and how the protocol is used today in real production environments. To determine this and extract useful information about SNMP usage we are collecting traffic traces from large research networks, ISPs or other organizations. By collecting and analysing traffic traces from several locations, we will be able to better understand what versions and features of the protocol are mostly being used, what optimizations (if any) are being done by implementations, either open-source or commercial, and what are the different usage patterns.

Since SNMP is mainly used to retrieve management information that is structured in conceptual tables, we are interested in how those tables are retrieved and how managers deal with things like holes in those tables.

The rest of the paper is structured as follows. Section 2 continues with an introduction of SNMP protocol, the trace files that were collected and the different tools that have been developed to process and analyse these trace files. Section 3 presents some related work in this field, while Section 4 describes our current setup that we use to process and analyse the traces. Section 5 talks about possible analysis that could be performed on these traces in order to extract relevant information about the use of the protocol in production environments. Section 6 is dedicated to SNMP walks and their properties, while Section 7 presents some of the results obtained from analysing our traces. Section 8 gives concluding remarks and identifies possible further work.

2 BACKGROUND

2.1 SNMP DESCRIPTION

A network that uses SNMP for management must contain three components: one or more network management systems (NMS), agents and managed devices. A managed device is a network element (e.g., router, switch, printer) that has an agent running on it. The agent is a software application which has access to the management information available on the managed device and provides this information to a network management system in SNMP format. The NMS is used to monitor and control the network, by polling and/or issuing management commands to the agents. The figure below depicts the components of a managed network.

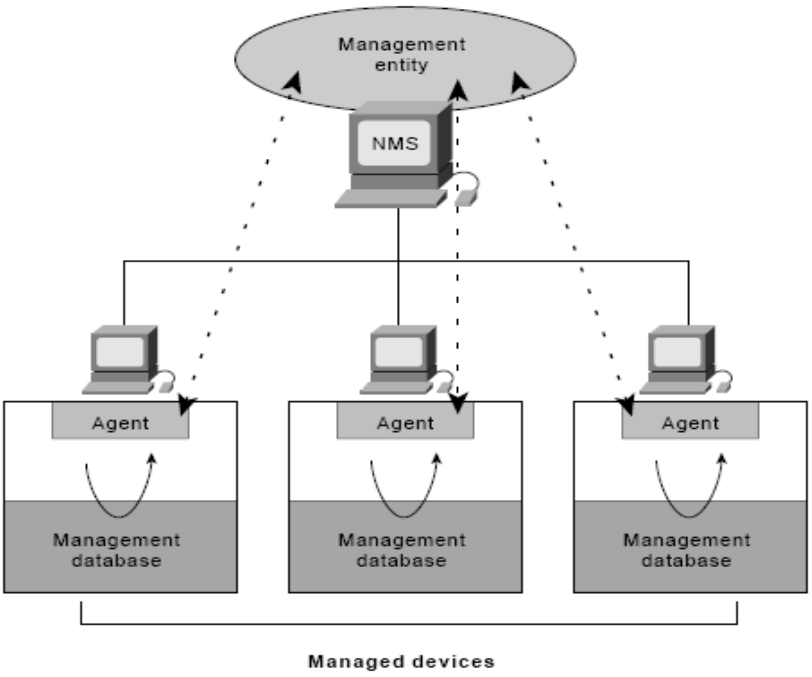


Figure 1: an SNMP managed network [1]

The management information present on a managed device is stored into Management Information Base (MIB). A MIB is a collection of objects that are uniquely identified by an Object Identifier (OID). Objects can either be

scalar (they have just one object instance) or columnar, in which case they have multiple instances that can be thought of as making up a conceptual table.

SNMP provides several protocol operations. Each message exchanged between a sender and a receiver contains a Protocol Data Unit (PDU) which describes the operation that needs to be performed by the receiving SNMP engine. As described in RFC 3411 [11], a PDU may belong to one of the following classes: read, write, response, notification and internal. Read PDUs are used to retrieve data from an SNMP engine; write PDUs define operations that will modify configuration or management data, while response PDUs contain operations that are sent in response to requests. The notification PDUs define notification operations, usually sent from an SNMP agent to a manager (for example a trap).

The following table briefly describes the SNMP protocol operations:

Operation Name	Description
Get	The Get operation is used to retrieve the value of one or more object instances from an agent.
GetNext	The GetNext operation is used to retrieve the next instance of an object, for example in a column or table.
GetBulk	The GetBulk operation was introduced in SNMPv2 and is used to retrieve a larger amount of data, such as multiple table rows. Unlike Get, if some values cannot be provided, the operation will return partial results.
Set	The Set operation can be used to set the value of some object instance on the agent.
Trap/Inform	The Trap/Inform operation is used by an agent

	to report events to a manager.

Table 1: SNMP protocol operations

2.2 TRACES

The first step in analysing SNMP traffic is to collect traces from various locations [4]. Because of the sensitive nature of the information exchanged in SNMP messages, the traces must be anonymized to protect the sensitive data [5].

Initially, all traces are stored in PCAP format, being captured with tools such as "tcpdump". Two additional file formats have been defined in order to facilitate easy access and sharing of trace files [2]. An XML-based format is able to capture all information present in the original PCAP file and has the advantage that it is both human and machine readable. Another available format is CSV (comma separated values), which retains less information than the XML format, but it is much more easy to parse and process (it is also much less in size than XML files). At the moment, most of our analysis has been conducted on CSV files.

Large trace files can be separated into multiple files containing message flows. A flow is defined as all messages exchanged between a source and a destination address pair [4].

2.3 TOOLS

The conversion of raw "tcpdump" files stored in PCAP format to either XML or CSV formats is done with "snmpdump" [4]. This tool can ensure that sensitive information is removed from SNMP traffic and performs IP anonymization [5].

We have developed another tool called "snmpwalk", a PERL script, that uses the CSV output of "snmpdump" and extracts relevant information for our statistics (for more information, please see section 4). This is still work

in progress, so other tools will be developed to analyse the protocol from different points of view and test different theories.

3 RELATED WORK

Although there are very few or no studies at all on SNMP characteristics and usage patterns derived from trace files from real networking environments, such studies have been performed for other protocols and they offer us an insight into analysis methods that can be applied in the case of SNMP. Also, other papers on performance characterization and SNMP protocol implementations are presented in this section.

Looking at other protocols, a DNS performance analysis is presented in [12]. The authors analyse DNS traces and associated TCP traffic with a focus on client perceived performance. Their analysis includes various statistics about lookups, like the number of referrals involved in a typical DNS lookup operation and the distribution of lookup latency. The latency is calculated by using a sliding window of last lookup seen and calculating the time between a lookup in the window and the corresponding response. In our analysis we use a similar concept in calculating the latency of SNMP messages, by calculating the time between requests and subsequent responses.

Other DNS studies [15] look at response time distributions for name servers. The authors use a NeTraMet [16] meter to measure the time between DNS requests and their corresponding responses. NeTraMet is an network accounting meter which builds up packet and byte counts for traffic flows. Each traffic flow is identified by the end point addresses. The results presented in the paper show evidence of multipathing behaviour. The authors also propose a few enhancements to the NeTraMet package to improve data collection quality.

Going into the database realm, in [13] user access patterns are used to characterize the load imposed on a database management system from a user perspective. By analysing the query traces of database engines, a user access graph is computed specifically for each individual application using the database, and based on this graph the authors propose several ways of improving system performance by pre-fetching future queries. It

will be interesting to see if such an approach can also be applied to SNMP and if "manager access patterns" can be computed. Based on this an agent could schedule in advance the tasks needed to respond to future requests from the manager application and thus decrease the response time.

One of the topics in our analysis of SNMP traffic traces is to determine the ratio between periodic and aperiodic traffic. Although SNMP is mostly used for periodically polling network devices for status information, which will generate a mostly periodic traffic function, the protocol allows these devices to asynchronously signal to managers that their state is about to change by issuing a trap. If this "feature" is used in production environments from which we receive our trace files, then we should be able to see some aperiodic traffic components. A first step in analysing this will be to separate or extract the aperiodic traffic.

In [3], the authors present a way of using wavelet analysis to discover network traffic anomalies. If we consider the aperiodic traffic in SNMP traces as "anomaly", because it should differ from the more period traffic generated by polling, we can employ the methods presented in this paper to extract it. The wavelet processing of the signal is done in two steps: analysis/decomposition and the inverse of this, reconstruction/synthesis. In the analysis process the original input signal is decomposed into its components, by convolving the input signal with a filter. A filter that has a smoothing or averaging effect will output a low frequency signal, which should capture the more general aspects of the traffic (i.e.: the periodic traffic). By applying other special filters to the input signal, more discrete properties can be identified and these will belong to the upper strata component (i.e., aperiodic traffic).

In [14], the authors describe a tool called "Tmix" that can be used to generate realistic TCP flows that mimic the workload of specific applications. Their work is useful in our research, as we will try to create a model for NS2 simulator that will mimic network management traffic. Their system takes as an input a trace of packet headers taken from the

network of interest, which is then reverse-compiled into a characterization of each TCP connection found in the trace file. That characterization, which they call a "connection vector", is used as an input to an NS2 module, called "Tmix", which is able to emulate the source-level behaviour of all applications that created the trace in the original network.

Other studies on the performance of SNMP focus on efficient information retrieval from managed devices [6]. In this paper, the authors propose the introduction of a polling layer that could coordinate and manage the polling activities of several managers, especially if they all run on the same network management system. The paper demonstrates that the use of such a polling layer leads to an important gain, especially in monitoring-intensive applications.

In [21] the author proposes a new SNMP operation to address the deficiencies of the GetBulk operation. A new GetRange operation is proposed that will not make use of a max-repetitions parameter, which requires the manager to know before hand of make a guess about how much data is available in a particular table. The author proposes the use of bumper OIDs which should replace the max-repetitions parameter and identify where the retrieval operation should stop.

4 ANALYSIS OF TRAFFIC TRACES

This section presents different areas of interest in traffic trace analysis that answer questions about the use and implementation of the protocol. We can organize our research and look at SNMP characteristics on several layers. As described in [11], an SNMP entity is composed of an SNMP engine, on top of which higher level applications work. Figure 2 below, presents the different layers on which our analysis is focused.

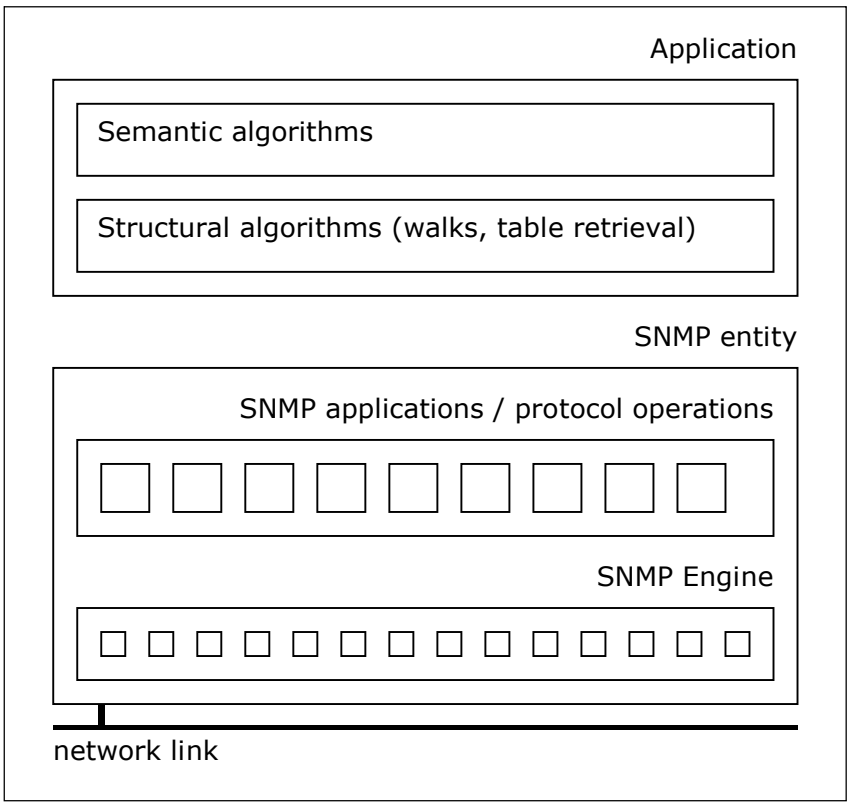


Figure 2: decomposition of a management system

The first layer in our figure represents the network link layer (the physical network media), but our research does not cover this one and starts one layer up, the SNMP Engine messages layer. At this point we are only interested in characteristics of the packet flow, without looking at the contents of these packets. Going up to the SNMP application / protocol operations layer, we look inside each packet and try to determine general characteristics about the usage of SNMP. At the next layer we are

interested in what each flow represents and what are the basic algorithms used by an application to, for example, retrieve a table. At the top most layer we try to characterize higher level functions and algorithms performed by the application and based on this, we try to identify or at least categorize each application.

This rest of this section presents several questions that will be investigated by running analysis on our trace files. We look at basic statistics, such as the protocol version used and most common operations. By looking inside the captured packets and the query pattern, we try to identify the algorithms and the management application that is being used. Analysing and isolating the periodic and aperiodic traffic, we can determine what actions (if any) are taken in case of network failures or notifications received from agents. From an implementation point of view it will be interesting to see what algorithms are being used to retrieve tables and if advanced features, like concurrent table retrieval, are used in production management applications.

The list of possible types of analysis presented in this section is not meant to be final. Other approaches and new analysis methods might be added during our investigations.

4.1 SNMP ENTITY ANALYSIS

This type of statistics presents a general overview on SNMP usage by extracting general characteristics from our trace files. We can identify several layers or levels at which such statistics can be collected, such as message layer and protocol operations layer, as described in the previous section.

At the packet layer, we are only interested in properties of the traffic and not the contents of each packet. Some of the things we would like to measure are packet size distribution, arrival time distribution and number of packets per flow. This information can be used to create SNMP traffic models.

One layer up, on the protocol operations layer, we look inside each packet and try to determine general characteristics about the SNMP usage. Some of the things we look for at this layer are the SNMP version, the protocol operation used, the requested MIB objects and the number of objects per packet.

4.2 APPLICATION ANALYSIS

At the structural algorithms layer, we are interested in determining what are the most common operations exchanged between a manager and an agent. We are particularly interested in identifying the different types of walks (see Section 4.4 for more information on walks) and their characteristics, such as the number of packets or the MIB modules being queried. By looking at the number of packets in each walk and the number of variable bindings we can get an estimate of the relative size of each walk and based on this we can determine the walk size distribution. Furthermore we can look at the total time required for a walk to finish and get an idea about the time needed to get a response for certain requests. Finally we can look at the distribution of walks and determine their properties. This information will also be useful in determining the type of algorithms the manager applications employs in order to retrieve tables.

These statistics will help us get a feeling on what is the current use of SNMP in production environments and what data is of most interest for applications.

4.3 TRAFFIC PERIODICITY

It is assumed that SNMP is mostly used to periodically poll managed devices on the network. It will be interesting to look at how much of the traffic from our traces falls into this category, that is, it is periodic, and what is the proportion of aperiodic traffic. Aperiodic traffic is most likely the result of ad-hoc requests by operators or management applications and it is worth to investigate what percentage of this traffic is caused by

traps and notifications coming from agents on the managed network devices.

Another important aspect would be to determine what the different time scales are where periodic traffic is visible. By identifying and eliminating this traffic we will be able to analyse the aperiodic traffic and determine if there is any correlation between this and periodic traffic.

One method of analysing traffic traces is by employing wavelet analysis [3]. This method decomposes the input signal into several strata, creating different signal components corresponding to different frequency levels. Each of these components has time as its independent variable. The lower strata component contains the low-frequency characteristics of the signal, or in other words, the slow-varying properties of the signal. We will expect this component to reveal large time scale (i.e.: daily) traffic periodicity. In contrast, the upper strata component contains the high-frequency characteristics of the signal, which will point out to more sporadic and possibly short lived properties of the signal.

Another way of looking at the periodicity of the traffic is by analysing it using Fourier Transforms. A Fourier Transform (FT) is a frequency representation of a function that allows us to analyse the function by looking into the frequency domain.

4.4 TABLE RETRIEVAL ALGORITHMS

Table retrieval (or table walk) is an important operation, often performed by SNMP managers, and it is important to analyse how this is implemented. To identify the different types of table retrieval mechanisms, we must first look at what are the possible ways of retrieving a table and how would that be reflected in our trace files.

It is important to make the distinction between a table retrieval algorithm, which specifies how the cells of a conceptual table could be retrieved, and a table walk, which we try to extract from our trace files using the "snmpwalk" tool mentioned in Section 2.3.

A table walk is defined as a sequence of get-next/response or get-bulk/response operations. A mixture of get-next and get-bulk operations is not considered a walk. At least one object identifier in the sequence of requests at the same varbind index must be increasing lexicographically, while all object identifiers at the same varbind index must be non-decreasing. Furthermore, all object identifiers in a response packet must be increasing lexicographically with respect to the object identifiers on the same varbind index of the precedent request packet. Any request packet (except the first one) should include at least one object identifier from the previous response, on the same varbind index. The end of the walk is detected if either a request never sees a response or there is no subsequent request that matches the last seen response within a given time interval.

A strict walk is a walk in which all object identifiers contained in all request packets except the first one are equal to the object identifiers on the same varbind index from the preceding response packet.

A prefix constrained walk is a walk in which all object identifiers on the same varbind index have the same object identifier prefix. This prefix is determined by the object identifiers of the first packet in the walk.

As described in [7], there are three basic retrieval algorithms that can be used to retrieve data from a conceptual table: column by column, row by row and column plus row. A graphical representation of these three algorithms is presented in Figure 3, below.

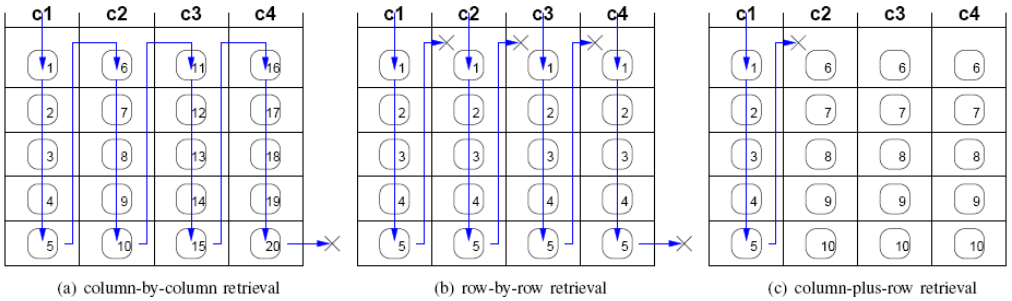


Figure 3: different types of table retrieval algorithms. [7].

In column by column table retrieval, the manager starts by retrieving all the elements of the first column, and then moves to the next one and so on until it reaches the end of the table. This algorithm uses n x m + 1 operations, where n and m are the table dimensions. As you can see, this is the least optimized way of retrieving a table and puts a lot of load on the agent. In our trace files, this algorithm should be reflected by a strict and not prefix constrained walk with only one varbind and n x m + 1 request/response pairs.

In row by row table retrieval, the manager retrieves one row at a time, so it only needs m + 1 operations. This algorithm uses a lot less operations to retrieve a table, but on the other hand it has bigger message sizes, which can be a problem in case of large rows, because SNMP only guarantees 484 bytes messages. In our trace files we should be able to detect such an algorithm by looking at strict and prefix constrained walks with n (the number of columns) variable bindings and m + 1 request/response pairs.

In the third algorithm, column plus row table retrieval, the manager retrieves the first column and then uses a get operation to retrieve each row at a time. This algorithm uses more operations than the row by row algorithm, but less than column by column. The trace file should contain a walk that retrieves the first column and then a series of "get" operations, connected to that walk. It is still not clear how we can properly detect such algorithms in our trace files, but this is something that we will investigate.

Some other table retrieval algorithms have been proposed, such as the pipelined algorithm for bulk table retrieval [8]. This algorithm uses concurrent requests to read several parts of the desired table at the same time, thus speeding up the process but at the same time imposing more load on the agent. Some runtime optimizations are described, where the manager can vary the number of active threads or the retransmission interval based on the network load. It will be interesting to see if such (advanced) algorithms are actually used in production, and if this is the

case, to implement solutions that will enable us to detect such methods in our trace files.

4.5 APPLICATION IMPLEMENTATIONS

We try to identify different implementations of management applications. To the very least, we should be able to categorize these implementations into "smart" and "not-so-smart" groups. We are interested in the type of algorithms used and determine the pattern in the application's behaviour (i.e., if after retrieving table X, the application also retrieves table Y or performs some other operation Z). We will try to categorize applications by looking at measured results from the other statistics we collect, such as data retrieval algorithms (i.e., is the application using advanced table retrieval algorithms that involve concurrent readings of several parts of the requested data?).

By looking at traffic traces for which we know the manager applications, we might be able to identify specific characteristics (i.e., specific values for the parameters used in describing a manager) of those applications and then construct detectors that will identify this manager in other trace files.

5 TRACE PROCESSING AND DATABASE SCHEMA

5.1 NAMING CONVENTIONS

A trace's name is determined by the location (organization) it was taken from and the number of other traces taken at that location. Each location has a number assigned to it, which uniquely identifies it. Formally, a trace name has the following form:

l[location number]t[trace number]

So, for example, the first trace taken on location 6 would have the name "l06t01".

At the next lower level, the flows are named based on the trace they are part of, the type of the flow and the IP addresses of the endpoints. We currently have two flow types: command flows, which are initiated by a command generator or manager and notification flows, which are initiated by a notification originator. Formally, a flow name has the following form:

<trace name>-<cg|no>-<IP of initiator>-<IP of responder>

So, for example, the command flow between 192.168.1.1 and 192.168.1.123 of trace 'l06t01' will be named:

l06t01-cg-192.168.1.1-192.168.1.123

In the database we keep trace and flow information in two fields, named "trace_name" and "flow_name", which are common to most of the tables (see below).

On disk, all files belonging to a trace are kept under the same directory, named after the name of the trace. This directory includes the original trace data in PCAP format and CSV format and additional files generated by the processing scripts, containing statistics or SQL statements that can be imported into the database. Since traces can be split into flows, a "flows" directory is used for storing all flow files.

5.2 DATABASE SCHEMA

We have created a database named "snmptrace" using MySQL as a database server to store the information computed by some of our scripts, like "snmpwalks.pl" and "snmpstats.pl". The major advantage of using a database is that we can easily extract data by running different queries on the data stored.

To keep the information consistent and handle the cases where a script is run more than one time, we label every record from the database with a trace or flow name, so that whenever we need to re-run a script that should update the database information for a particular trace or flow, we know exactly what records to replace.

The diagrams presented in this and the following sections are Database Model Diagrams.

5.2.1 TRACE/FLOW META INFORMATION

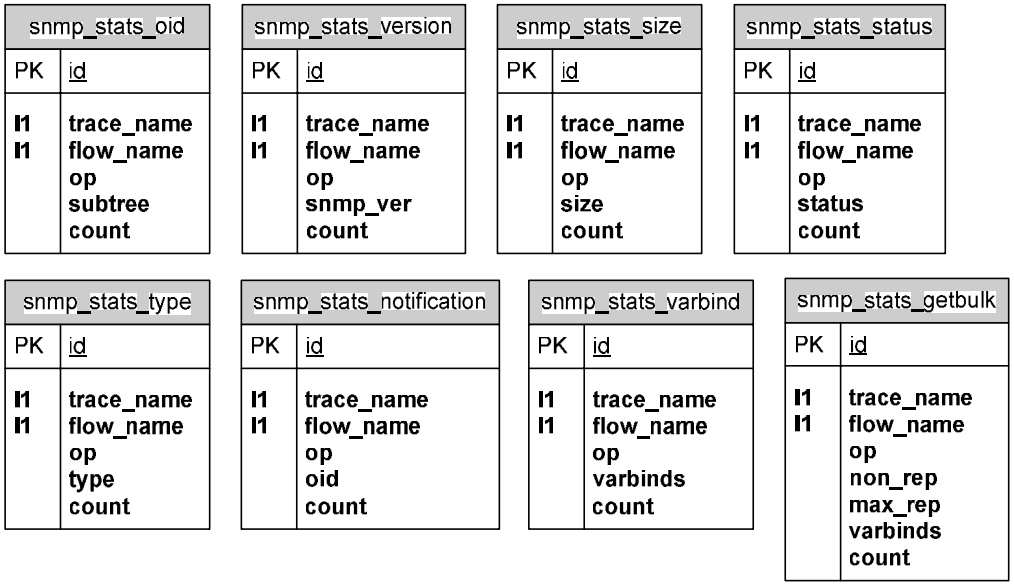
The "snmpstats.pl" script extracts meta information from trace files or flow files. The information we collect at the moment includes start and end timestamps of the trace/flow, the number of messages seen and the number of managers, agents or unknown endpoints discovered (this only makes sense when the script is processing whole trace files). We use the following database table to store this information:

snmp_meta	
PK	id
I1 **I1**	**trace_name** **flow_name** **start_timestamp** **end_timestamp** **messages** **managers** **agents** **unknown**

An empty flow name in this table or any of the tables presented in the next section indicates that the data represents the whole trace rather than a specific flow.

5.2.2 TRACE/FLOW STATISTICS

Additional trace/flow statistics are computed by the "snmpstats.pl" script and stored in the following tables of the database:

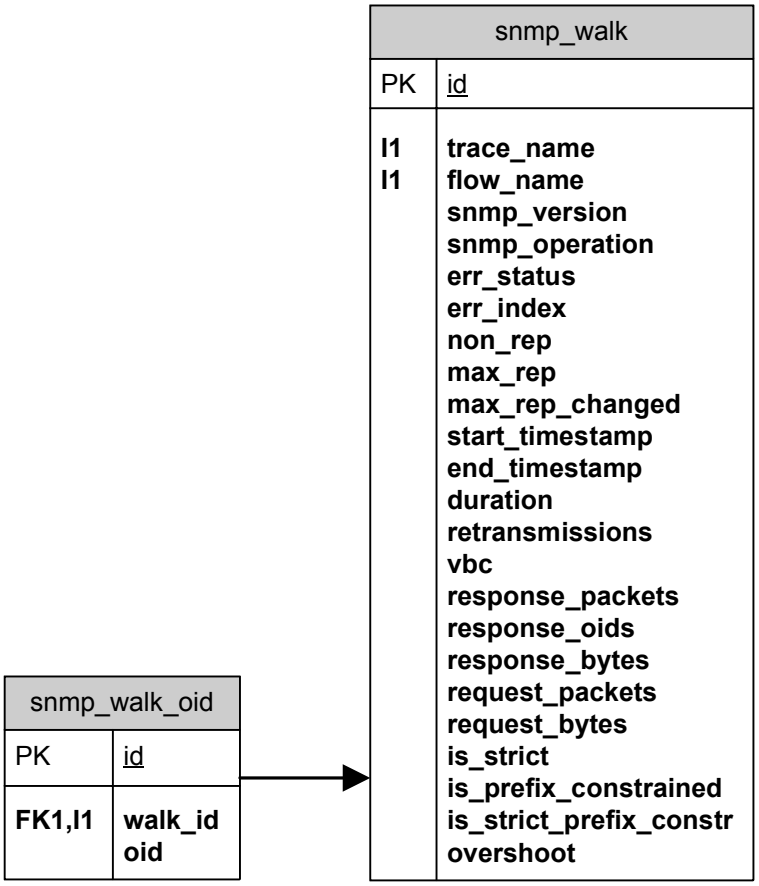


The "snmp_stats_oid" table shows a classification of OIDs based on their prefix, broken down by the SNMP operation. Table "snmp_stats_version" provides information on the SNMP operations, broken down by the SNMP version. The "snmp_stats_size" table stores the message size distribution for each SNMP operation. The distribution of status codes broken down by SNMP operation is stored in table "snmp_stats_status".

The distribution of types used in SNMP operations is stored "snmp_stats_type". The distribution of notifications types broken down by sub-tree is stored in "snmp_stats_notification". The "snmp_stats_varbind" table shows the number of elements in the varbind list for each operation, while table "snmp_stats_getbulk" describes the parameters used in get-bulk operations.

5.2.3 WALK INFORMATION

The "snmpwalks.pl" script currently uses two database tables to store walk information computed from CSV trace files. Please see the model diagram below:



The "snmp_walk_oid" table is linked to the "snmp_walk" table through the column "walk_id", which represents the "id" column from "snmp_walk". It is a one to many relationship between the two tables (one walk has one or more starting OIDs).

Each discovered walk is represented by a record in the "snmp_walk" table. We use a second table, "snmp_walk_oid" to store the starting OIDs for every walk. The table below describes in more details the fields of "snmp_walk" database table:

Field Name	Description
id	Auto increment and primary key for this table.
trace_name	The name of the trace to which this walk belongs to.
flow_name	The name of the flow to which this walk belongs to.
snmp_version	The SNMP protocol version used.
snmp_operation	The SNMP protocol operation used (get-next-request or get-bulk-request).
err_status	The error status of the last response packet from this walk.
err_index	The error index of the last response packet from this walk.
non_rep	The non-repeaters value from the last request packet of this walk.
max_rep	The max-repetitions value from the last request packet of this walk.
max_rep_changed	Indicates if the max-repetitions parameter was changed by the command generator during this walk.
start_timestamp	The timestamp of the first packet from this walk.
end_timestamp	The timestamp of the last packet from this walk.

duration	end_timestamp – start_timestamp
retransmissions	The number of retransmissions detected in this walk.
vbc	The number of varbinds simultaneously retrieved in this walk, excluding any non-repeaters.
response_packets	Total number of response messages in this walk.
response_oids	The total number of objects contained in all response messages of this walk.
response_bytes	The total number of bytes in all response messages.
request_packets	The total number or request (get-next-request or get-bulk-request) messages in this walk.
request_bytes	The total number of bytes in all request messages.
is_strict	Indicates if this walk is strict (0/1).
is_prefix_constrained	Indicates if this walk is prefix constrained (0/1).
is_prefix_constrained_strict	Indicates if the OIDs of this walk never go out of the initial prefix (that is, the walk doesn't reach the end of the table).
hole_detection	If the manager detects holes in this walk this field will be "1", if holes are ignored the field will be "-1", otherwise "0".

5.2.4 QUERIES ON THE DATABASE

This section describes a list of queries that can be executed on this database to extract information about the traces. All the queries presented can either be executed on data from all traces or just one single trace or even flow. All queries below will look at the data for trace 'l12t01'.

5.2.5 TRACE/FLOW RELATED QUERIES

Determine the size of every operation within each flow of a trace:

SELECT flow_name, op, SUM(count * size) AS size from snmp_stats_size WHERE trace_name = 'l12t01' AND flow_name != '' GROUP BY flow_name, op;

Determine the level of usage for every SNMP version within each flow of a trace:

SELECT flow_name, snmp_ver, SUM(count) AS size from snmp_stats_version WHERE trace_name = 'l12t01' AND flow_name != '' GROUP BY flow_name, snmp_ver;

Determine the "busiest" flows in a trace (in terms or number of messages):

SELECT flow_name, messages AS size FROM snmp_meta WHERE trace_name = 'l12t01' AND flow_name != '' ORDER BY size DESC;

Determine the longest flows in a trace:

SELECT flow_name, ROUND((end_timestamp-start_timestamp)/3600, 1) AS size FROM snmp_meta WHERE trace_name = 'l12t01' AND flow_name != '' ORDER BY size DESC;

Determine the most queried MIB sub-trees on the whole trace:

SELECT subtree, SUM(count) AS size FROM snmp_stats_oid WHERE trace_name = 'l12t01' AND flow_name != '' GROUP BY subtree;

Determine the most queried MIB sub-trees in each flow of a trace:

"SELECT flow_name, subtree, SUM(count) AS size FROM snmp_stats_oid WHERE trace_name = 'l12t01' AND flow_name != '' GROUP BY flow_name, subtree;

5.2.6 WALK RELATED QUERIES

Determine which SNMP versions are being used and how many walks use each one:

SELECT snmp_version, count(*) FROM snmp_walk WHERE trace_name = 'l12t01' GROUP BY snmp_version;

Determine which SNMP operations are being used and how many walks use each one:

SELECT snmp_operation, count(*) FROM snmp_walk WHERE trace_name = 'l12t01' GROUP BY snmp_operation;

Determine how common retransmissions are:

SELECT retransmissions, count(*) FROM snmp_walk WHERE trace_name = 'l12t01' GROUP BY retransmissions;

Determine how many walks are strict, prefix constrained or have never reached the end of the table (in other words, they are strict prefix constrained):

SELECT is_strict, count(*) FROM snmp_walk WHERE trace_name = 'l12t01' GROUP BY is_strict;

SELECT is_prefix_constrained, count(*) FROM snmp_walk WHERE trace_name = 'l12t01' GROUP BY is_prefix_constrained;

SELECT is_strict_prefix_constr, count(*) FROM snmp_walk WHERE trace_name = 'l12t01' GROUP BY is_strict_prefix_constr;

What is the distribution of response packets or total number or OIDs retrieved?

SELECT response_packets, count(*) FROM snmp_walk WHERE trace_name = 'l12t01' GROUP BY response_packets;

SELECT response_oids, count(*) FROM snmp_walk WHERE trace_name = 'l12t01' GROUP BY response_oids;

What is the distribution of walks among flows?

SELECT flow_name, count(*) FROM snmp_walk WHERE trace_name = 'l12t01' GROUP BY flow_name;

Are there any managers that vary the number of max-repetitions (in get-bulk requests)?

SELECT max_rep_changed, count(*) FROM snmp_walk WHERE trace_name = 'l12t01' GROUP BY max_rep_changed;

What is the ratio between the size of the requests and size of the responses? (This can further be evaluated on a per flow basis):

SELECT flow_name, SUM(request_bytes)/SUM(response_bytes) FROM snmp_walk GROUP BY flow_name;

What is the distribution of walks based on how many columns are retrieved at the same time?

SELECT vbc, count(*) FROM snmp_walk WHERE trace_name = 'l12t01' GROUP BY vbc;

What are the Top 10 starting positions among all walks?

SELECT t.prefix, count(*) AS c FROM (SELECT GROUP_CONCAT(t2.oid SEPARATOR ', ') AS prefix, count(*) FROM snmp_walk AS t1, snmp_walk_oid AS t2 WHERE t1.id = t2.walk_id GROUP BY t1.id) AS t GROUP BY t.prefix ORDER BY c DESC LIMIT 10;

6 SNMP WALKS

6.1 DEFINITIONS

This section introduces a few notations and definitions to formally describe a walk and its properties. Section 6.3 presents the algorithm that is currently used to extract walks from the traces that we have. First, we introduce a few notations that will help us in the definitions below.

A message belonging to an SNMP flow can either be a request message or a response message. Assuming we have a sequence of messages, we will use $m_{type,\ i}$ to denote the i^{th} message of a certain type in that sequence. A message type can either be "request" or "response". We will also use "*" for types as a shortcut for "any type". For example $m_{request,\ 2}$ is the notation used for the 2^{nd} request message in some sequence.

An SNMP message contains a lot of information including source and destination IP addresses and port numbers, the SNMP operation and version used and the list of object identifiers being requested or received. We will only give notations for the information relevant in defining the walks below.

Each message contains a list of object identifiers that are either requested in a request message or received in a response message. We use $o_{type,\ i,\ k}$ to denote the object identifier at position k in the i^{th} message of type "type". For example $o_{request,\ 2,\ 3}$ is the notation used for the 3^{rd} object identifier in the 2^{nd} request message in some given sequence of messages.

We denote with Source($m_{type,\ i}$) and Destination($m_{type,\ i}$) the transport endpoints associated with a given message m.

Def 1: A walk w is a sequence of requests of the same type (get-next or get-bulk) and response messages with matching source/destination IP addresses and port numbers, within a time interval t from one another, during which all subsequent request object identifiers at the same varbind position are not decreasing lexicographically and at least one object

identifier in a subsequent request message is the same as the object identifier on the same position from the previous response message.

Formally, a sequence $\{m_{request,\ 1}, m_{response,\ 1} \ldots m_{request,\ n}, m_{response,\ n} \mid Destination(m_{request,\ i}) = Source(m_{response,\ i}) \wedge Source(m_{request,\ i}) = Destination(m_{response,\ i})$ for all i} is called a walk iff $o_{request,\ i,\ k} \geq o_{request,\ i-1,\ k}$ for all k and $2 \leq i \leq n$ and exists k such that $o_{request,\ i+1,\ k} > o_{request,\ i,\ k}$ and $o_{request,\ i,\ k} = o_{response,\ i-1,\ k}$.

Another notation that must be introduced now is $Prefix(o_{type,\ i1,\ k1}, o_{type,\ i2,\ k2})$ which denotes that the second object identifier ($o_{type,\ i2,\ k2}$) has the prefix $o_{type,\ i1,\ k1}$.

Def 2: A walk w is prefix constrained iff we have $Prefix(o_{request,\ 1,\ k}, o_{request,\ i,\ k})$ for all $1 \leq i \leq n$ and all k and $Prefix(o_{request,\ 1,\ k}, o_{response,\ i,\ k})$ for all $1 \leq i < n$. In case the last response message is within the same prefix, we say the walk is strictly prefix constrained.

Def 3: A walk w is strict iff we have $o_{request,\ i,\ k} = o_{response,\ i-1,\ k}$ for all $2 \leq i \leq n$ and all k.

We give below additional definitions and notations that will be used in the future sections of this paper, as they are presented in [18]:

Volume of a walk: the volume v(w) of a walk w is the number of varbinds retrieved in all response messages included in the walk.

Start, end and duration of a walk: if w is a walk then, s(w) denotes the start timestamp of w, e(w) denotes the end timestamp of w, and d(w) denotes the duration of w, where d(w) = e(w) – s(w).

Delta-serial walks: two walks w1 and w2 are Δ-serial if s(w2) – e(w1) < Δ.

6.2 ASSUMPTIONS

This section describes the assumptions that we make in the walk extraction algorithm presented in the next section.

One of the assumptions we make is that during a walk, the number of columns retrieved will be the same throughout the whole walk. As a consequence, we also assume that the order of the column retrieved is not going to change during a walk.

Having the above assumption in our algorithm, we will generate two or more walks in case the manager application varies the number of columns retrieved during a walk. This could happen if, for example, the manager detects that a column has no more cells and decides to exclude it from further requests. In this case, our algorithm will produce a new table walk starting at the request that contains a different number of columns.

This case, however, can be detected by further investigating the data produced by the scripts and looking for delta-serial walks with different number of columns retrieved that start one after the other in the OID space.

6.3 WALK EXTRACTION ALGORITHM

This section presents the walk extraction algorithm. Before the algorithm, we present the relevant elements of the data structure used for a walk:

```
Walk {
  op                // the SNMP operation used (get-next, get-bulk)
  version           // the SNMP version
  request_id        // the ID of the last request message
  vbc               // the number of varbinds in this walk
  strict            // flag set for strict walks
  prefix            // flag set for prefix constrained walks
  retransmissions   // the number of retransmissions detected
}
```

The main loop of the walk detection script is presented below:

```
FOR EACH msg IN input file DO
  IF msg.operation IN (‘get-next-request’, ‘get-bulk-request’) THEN
    msg.type        <- ‘request’
    manager-IP      <- msg.source-IP
    agent-IP        <- msg.destination-IP
  ELSE IF msg.operation = ‘response’ THEN
    msg.type        <- ‘response’
    manager-IP      <- msg.destination-IP
    agent-IP        <- msg.source-IP
  ELSE
    GO TO NEXT msg
  END IF

  walk-key   <- manager-IP + ‘|’ + agent-IP
  found-walk <- FALSE

  FOR EACH walk IN open-walks [walk-key] DO
    result   <- Check-Walk (walk, msg)
      IF result = ‘OK’ THEN
        found-walk  <- TRUE
        EXIT LOOP
      ELSE IF result = ‘RETRANSMISSION’ THEN
        GO TO NEXT msg
      END IF
  END FOR EACH
  IF NOT found-walk AND msg.type = ‘request’ THEN
    walk            <- Create-New-Walk(msg, walk-key)
    found-walk      <- TRUE
  END IF

  IF found-walk = TRUE THEN
    Update-Walk(walk, msg)
  END IF
END FOR EACH
```

The function ‘Check-Walk’ above is used to determine if a message belongs to any already discovered walks or not. In case the message is a request message that cannot be assigned to any open walks, a new walk will be created.

```
FUNCTION Check-Walk (walk, msg)
  request-ID        <- packet [request-ID]
  walk-request-ID   <- walk [request-ID]
  msg-OK            <- 0

  IF msg.type = ‘request’ AND msg.op <> walk.op THEN
    RETURN FALSE
  END IF
  IF msg.type = ‘response’ AND msg.request-ID <> walk.request-ID THEN
    RETURN FALSE
  END IF
  IF walk.vbc <> msg.vbc THEN
    RETURN FALSE
  END IF

  // check for retransmissions:
  IF walk.request-ID <> ‘’ AND msg.type = ‘request’ THEN
    all-equal <- 1
    FOR k = 0 TO walk.vbc DO
      IF walk.last-request-OID[k] <> msg.OID[k] THEN
        all-equal <- 0
      END IF
    END FOR
    IF all-equal = 1 THEN
      RETURN ‘RETRANSMISSION’
    END IF
  END IF

  // determine what properties hold for this walk if we add this
  // new packet to it:
  prefix-constrained      <- 0
  strict                  <- 0

  all-prefix-constrained  <- 1
  one-increasing          <- 0
```

```
all-non-decreasing      <- 1
all-equal               <- 1
one-equal               <- 0

FOR k = 0 TO walk.vbc DO
  // prefix check:
  IF NOT (msg.OID[k] HAS PREFIX walk.prefixes[k]) THEN
    all-prefix-constrained <- 0
  END IF

  // checks for request packets
  IF msg.type = ‘request’ THEN
    IF msg.OID[k] > walk.last-request-OID[k] THEN
      one-increasing <- 1
    ELSE IF msg.OID[k] < walk.last-request-OID[k] THEN
      all-non-decreasing <- 0
    END IF

    IF msg.OID[k] = walk.last-response-OID[k] THEN
      one-equal <- 1
    ELSE
      all-equal <- 0
    END IF
  END IF

  // checks for response packets
  IF msg.type = ‘response’ THEN
    IF msg.OID[k] < walk.last-request-OID[k] THEN
      all-non-decreasing <- 0
    END IF
  END IF
END FOR

// determine if the packet belongs to this walk and what
// are the walk properties after adding this packet:
IF msg.type = ‘request’ THEN
  IF one-equal AND one-increasing AND all-non-decreasing THEN
    msg-OK <- 1
    IF all-prefix-constrained THEN
      prefix-constrained <- 1
    END IF
```

```
      IF all-equal THEN
        strict <- 1
      END IF
    END IF
  ELSE IF msg.type = 'response' THEN
    msg-OK <- 1

    // for responses to get-bulk requests, check walk properties
    // through all repetitions:
    IF walk.op = 'get-bulk-request' THEN
      FOR EACH repetition IN msg.repetitions DO
        FOR k = 0 TO walk.vbc DO
          IF NOT (repetition-OID[k] HAS PREFIX walk.prefixes[k]) THEN
            all-prefix-constrained      <- 0
          END IF
          IF repetition-OID[k] < last-response-OID[k] THEN
            all-non-decreasing   <- 0
          END IF
          last-response-OID[k] <- repetition-OID[k]
        END FOR
      END FOR EACH
    END IF
    IF all-prefix-constrained THEN
      prefix-constrained <- 1
    END IF
  END IF

  // if this packet belongs to this walk, update walk properties:
  IF packet-OK = 1 THEN
    IF walk.prefix = 1 AND prefix-constrained = 0 THEN
      walk.prefix <- 0
    END IF
    IF walk.strict = 1 AND strict = 0 THEN
      walk.strict <- 0
    END IF
    RETURN 'OK'
  END IF

  RETURN FALSE
END FUNCTION
```

6.4 TRANSLATION OF TABLE RETRIEVALS TO WALKS

This section describes the types of walks we expect to see for each of the table retrieval algorithms defined.

6.4.1 COLUMN BY COLUMN

This is the most simple table retrieval algorithm in which the manager retrieves a table by first walking the first column, then the second and so on until the end of the table. We are looking for single varbind walks for this type of table retrieval. If the walks start at the top of the table we also expect them to be prefix constrained.

Walks starting on the column header will not be prefix constrained. Our algorithm detects where the prefix constrained property is being lost, so in this case we are looking for walks that loose this property after a few received packets and not right from the first packet (which would signal a walk starting somewhere inside a table).

6.4.2 ROW BY ROW

In this case the manager retrieves more than one object at a time; therefore we expect to see walks with multiple varbinds. If the walks start at the top of the table or column header, they should be prefix constrained.

6.4.3 COLUMN PLUS ROW

In this case we expect to see single column walks followed by multiple columns walks or get-requests that are all Δ – serial. A proper value for Δ can be determined by looking at the distribution of inter-request times of the initial walk.

6.4.4 TABLE HOLES

In case of table holes the result depends on whether the manager detects there is a hole in the table or not. We look for cases where the manager requests an OID that is lexicographically between the last requested OID (or equal to this one) and the last received OID. These cases are presented in Figure 4 below. The walks in case (b) and (c) should loose their strict property, since the new OID requested by the manager is different from the last OID received.

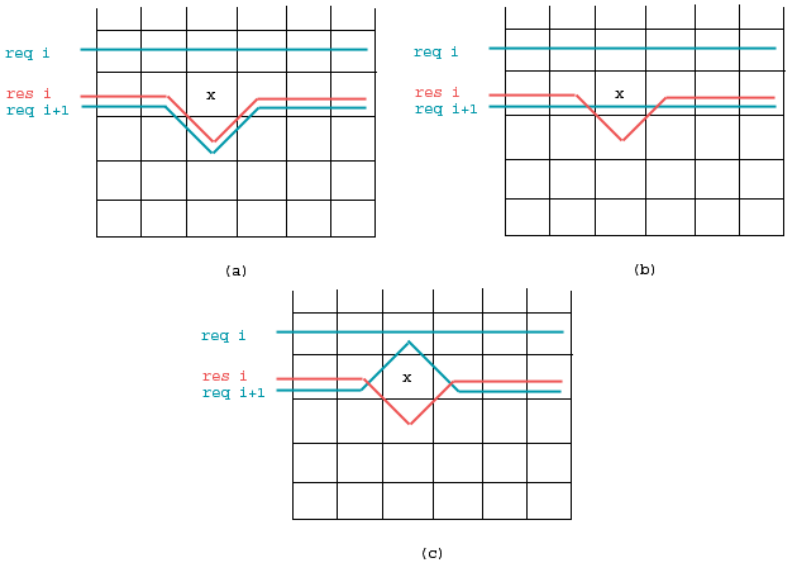


Figure 4: different types of handling a table hole.

In the Figure 4 above, in case (a) the manager does not detect the missing cell from the table and continues the requests using the OIDs received in the last response. In this case our script will generate a strict walk that is not prefix constrained, since, assuming that the table is rectangular, the end of the column that contains the hole will be reached before the end of the other columns.

In the second and third case, presented in the above figure as (b) and (c), we see the manager detecting the hole and using the missing cell (b) or requesting the same OID as in the previous request (c) in the next get-next-request. In this case our algorithm will produce a prefix constrained walk that is not strict, since the second request (req 2) contains an object identifier which is not equal to the object identifier on the same position from the previous response (res 1).

We have implemented a detection mechanism that allows us to flag all walks that contain holes. We do this by looking at every OID received by the manager and compare it with the OID that was requested. If the lexicographical distance between them is more than 1 we assume the table has a hole. Looking at both this flag and the strict flag we can determine how many manager implementations detect holes.

6.5 DROPPING OIDS FROM A WALK

A manager could decide to drop some OIDs from a walk after it determines that, for example, a column that was included in the walk is no longer available (i.e., has no more cells).

Our algorithm has no way of detecting this behaviour and in this case will produce a new walk that has a shorter list of varbinds and starts from somewhere inside the table. The two walks produced in this case should be Δ – serial and to find a proper value for Δ is to look at the inter-request times from the first walk.

7 RESULTS

This section presents some of the results we obtained from analysing the traces. We first describe the metrics that we use and then present different graphs or tables with the data collected.

7.1 METRICS

In determining the general SNMP usage statistics we used the number of messages as a metric, counting all messages with a certain property, like version, the MIB module queried or the number of varbinds retrieved. We aggregate this measurements on a per trace level or for each individual flow within a trace.

By using the same metric across various measurements we are better able to see the correlation between different properties. For example, it is expected that in flows or traces that use mostly get-next-request operations the size of all request packets is similar to the total size of response packets, while in cases where get-bulk operations are used the total size of request packets is significantly less than the total size of response packets. [4]

7.2 SNMP USAGE STATISTICS

We produced some graphs using Tree-maps for visualization to present some of the statistics collected from the traces.

The first graphs, shown on the next page in Figure 5, present the most queried MIB modules in two of the traces: 'l01t02' and 'l06t01'. As you can see from the graphs, the "mib-2/interfaces" branch is mostly used in both of the traces. Trace l01t02 has a more proportional distribution of what is being queried, while in trace l06t01 we see a clear domination of the "mib-2/interfaces" MIB and its extension "mib-2/ifMIB".

Figure 5: distribution of queried MIB in trace l01t02 and l06t01

In terms of SNMP versions being used, trace l01t02 only uses SNMPv1 while trace l06t01 also uses SNMPv2 for get-bulk operations. Figure 6 below show the distribution of SNMP operations being used:

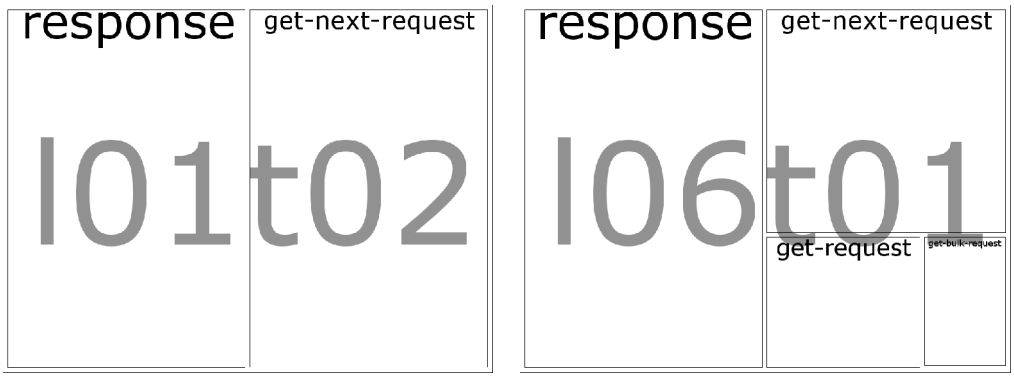


Figure 6: distribution of SNMP operations in l01t02 and l06t01

Figure 7 shows the number of objects retrieved or requested in each operation. Trace l01t02 only contains requests that retrieve one object; however, trace l06t01 shows many variations in the number of objects retrieved.

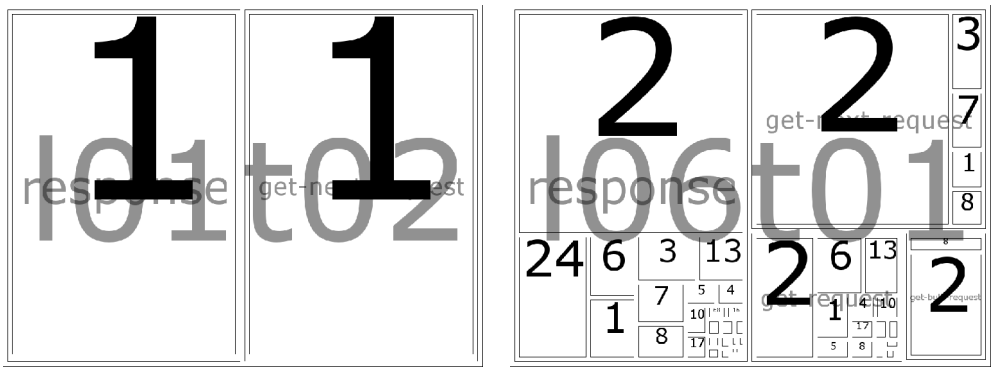


Figure 7: the number of objects retrieved in each operation for trace l01t02 and l06t01

If we take a look at the flows within a trace we observe that in both traces there are a few dominant flows. The following Figure 8 shows the flows in both traces and within each flow, we show the SNMP versions (up) and the operation (down).

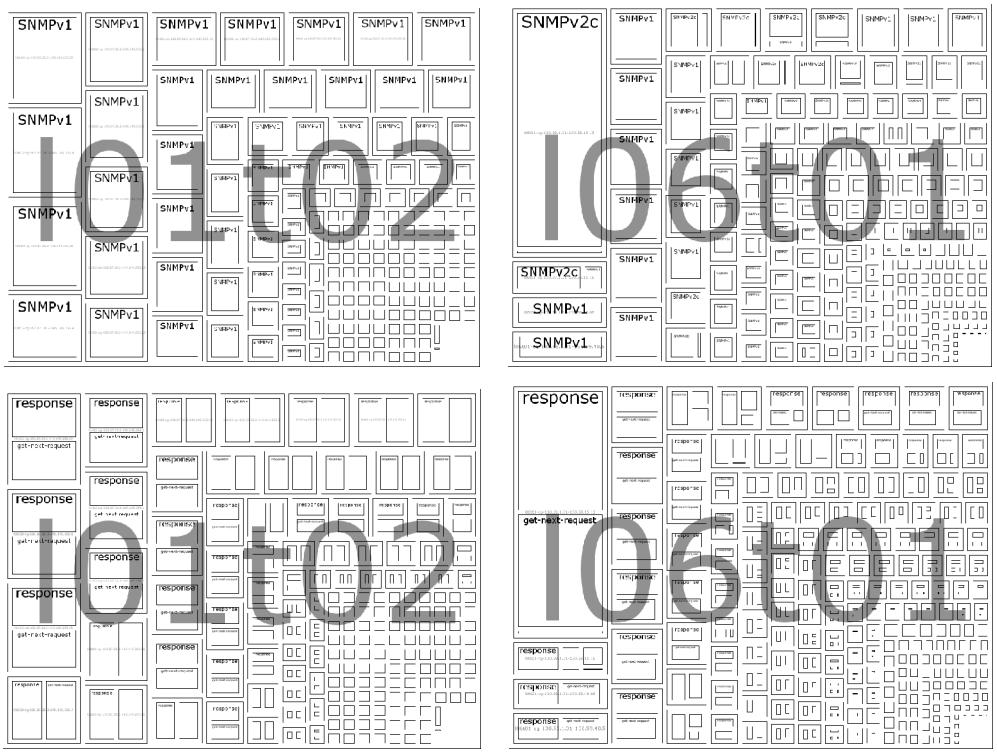


Figure 8: flows in trace l01t02 and l06t01; within each flow the distribution of SNMP versions (up) and SNMP operations (down)

7.3 RESPONSE TIME DISTRIBUTION

We created a script to calculate the response time distribution within a walk, flow or trace by looking at the timestamp difference between every request and its subsequent response. Figure 9 below displays the response time distribution in one of the flows of trace l06t01.

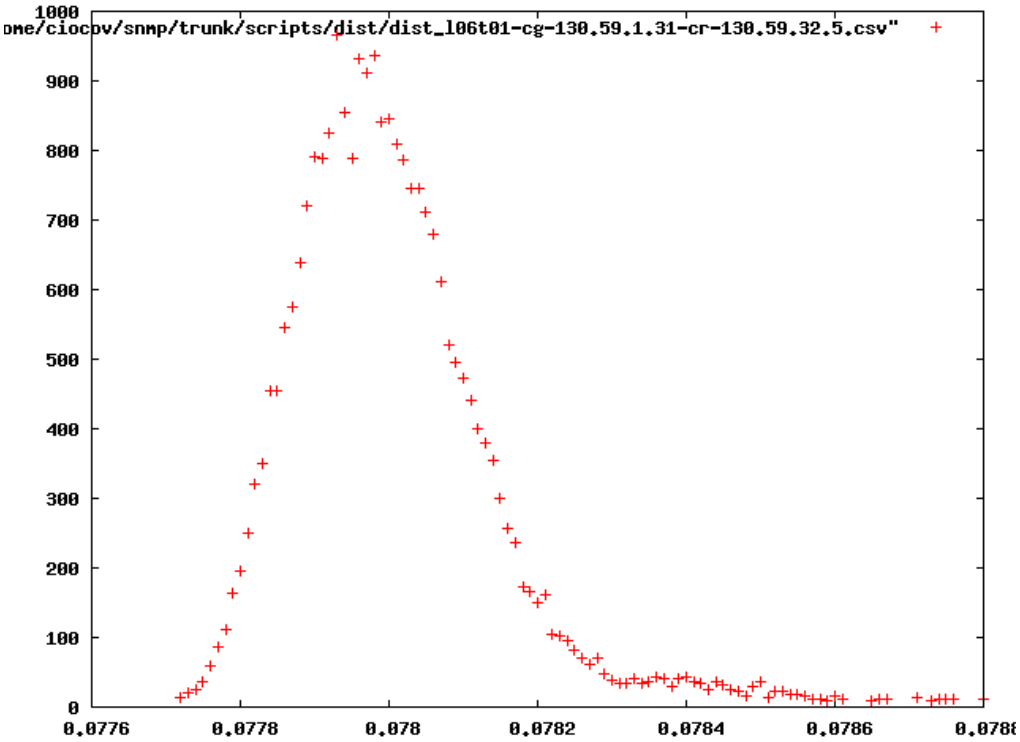


Figure 9: response time distribution (on the x-axis we show the response times in seconds, while on the y-axis the number of occurrences).

The response time distribution in Figure 9 is calculated on a flow that only contains 2-varbind walks and shows one peak around 0.078 seconds. In case of flows that contain more than one type of walk with respect to the number of varbinds retrieved, the response time distribution presents more than one peak, as seen in the following two figures.

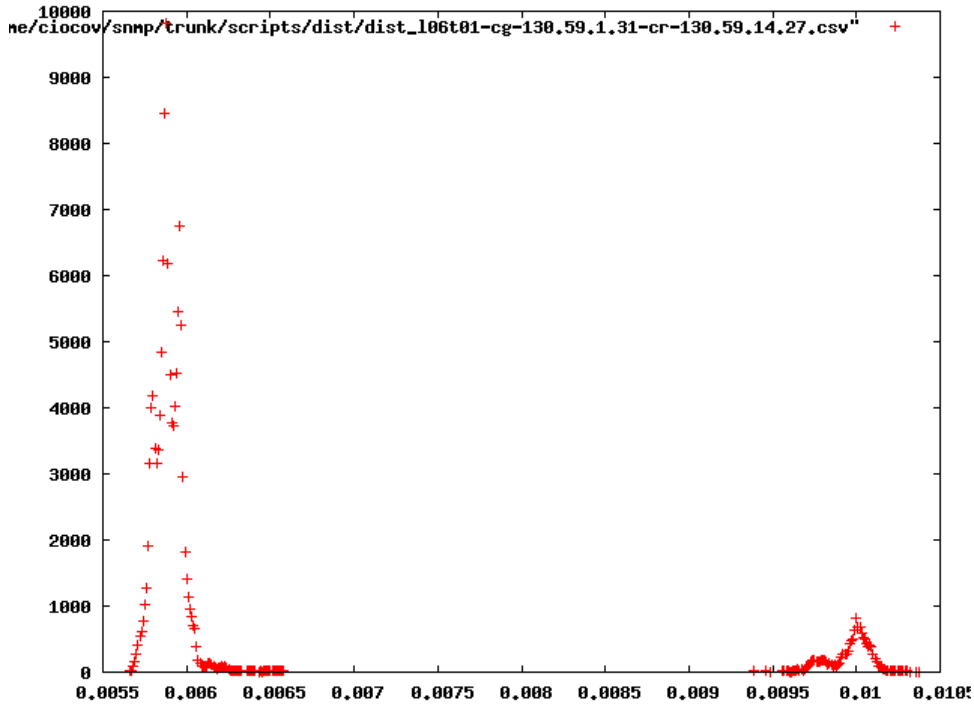


Figure 10: response time distribution (2 types of walks)

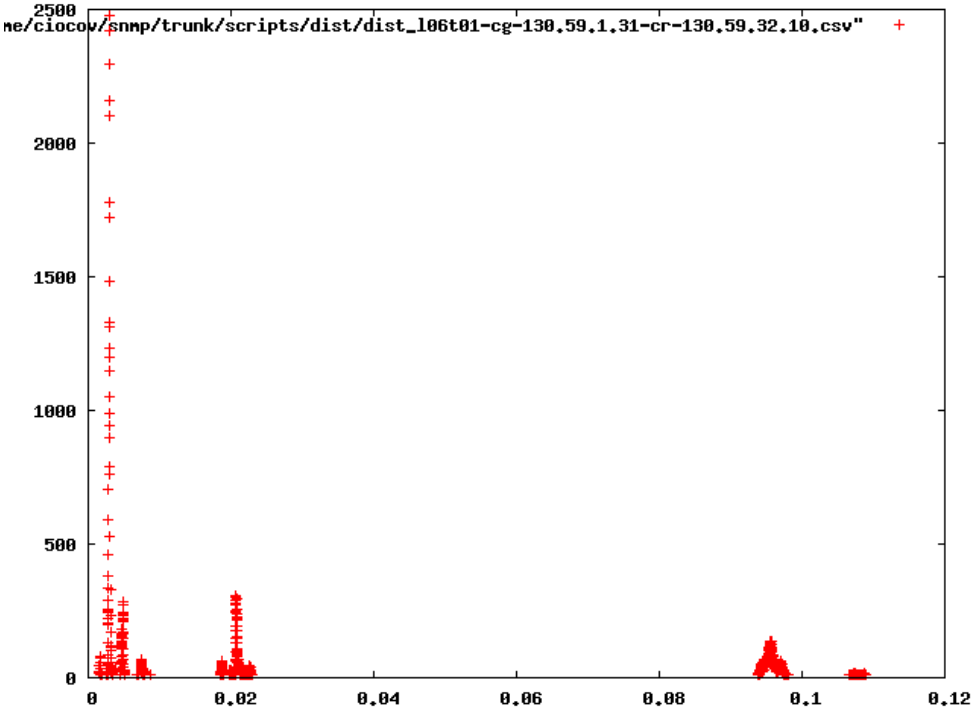


Figure 11: response time distribution (3 types of walks)

7.4 WALK STATISTICS

In this section we present walk related statistics, by counting the number of walks that have a certain property. We look at prefix constrained walks and strictly prefix constrained walks (i.e.: walks that never go out of the starting prefix). We also show the number of walks which were flagged as containing holes.

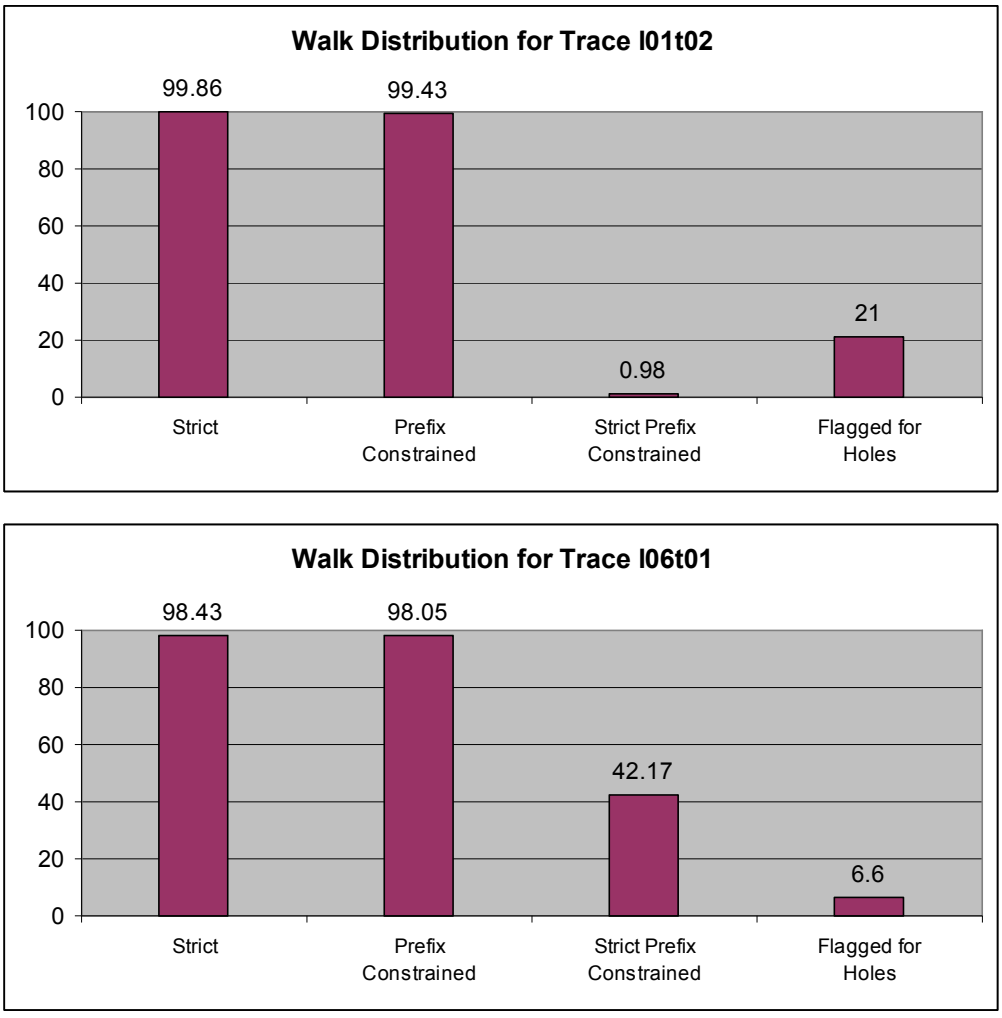


Figure 12: number of walks with certain property for trace l01t02 (up) and l06t01 (down)

We are also interested in the number of walks that were flagged for containing holes that are not strict, as this is an indication of smart

manager implementations that can detect table holes. Although the majority of walks in l06t01 are strict, some flows contain walks that are not strict. On a closer investigation it turns out these walks were used by smart manager implementations to retrieve tables with holes. Out of the total number of 62900 walks that are not strict, 52240 (or 83.05%) were also flagged as containing holes which were detected by the manager.

The table below shows a summary of walks that contain holes in six of the traces and gives an indication on how many of these walks were handled properly my manager implementations (the percentages are calculated from the total number of walks with holes).

Trace	Walks with holes	Detected	Not detected
l01t02	569277 (21%)	1830 (0.32%)	567447 (99.68%)
l01t05	-	-	-
l04t01	-	-	-
l06t01	264147 (6.6%)	52240 (19.77%)	211907 (80.23%)
l11t01	29528 (98.80%)	-	29582 (100%)
l12t01	18358 (26.53%)	-	18358 (100%)

7.5 OVERSHOOT

The overshoot is calculated as the number of retrieved object identifiers that are out of prefix in prefix constrained walks, which can only happen in the last response message of a walk. For walks that are strictly prefix constrained (i.e., they never go out of prefix) the overshoot value is always 0.

In trace l06t01, where we see a lot of get-bulk operations we counted the number of object identifiers retrieved that were out of prefix only for walks

using get-bulk requests that were prefix constrained. Our measurements show that only a small number of these object identifiers were retrieved as indicated in the chart below:

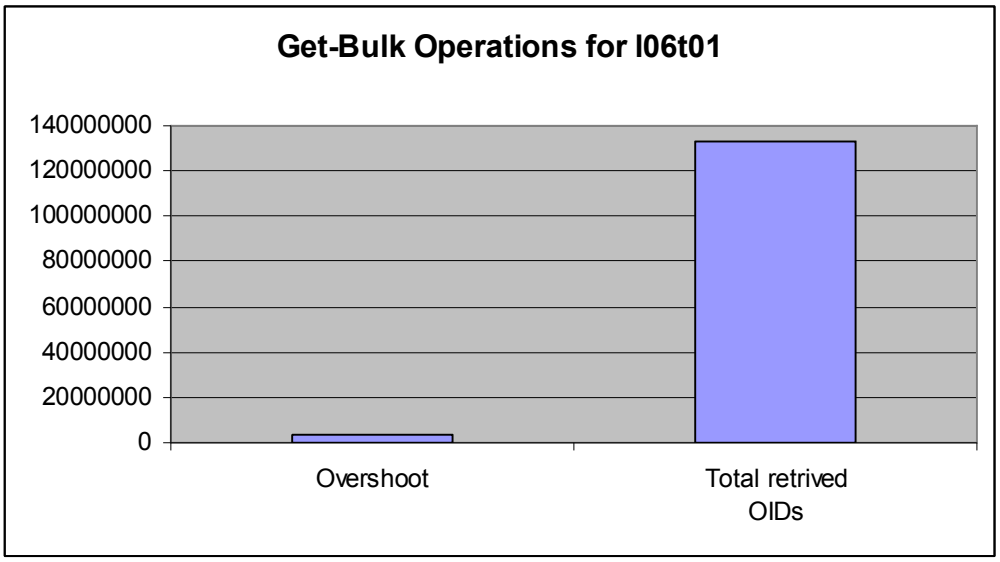


Figure 13: the number of out-of-prefix object identifiers retrieved in the last response of get-bulk prefix constrained walks

Although the overshoot value is small compared to the total number of OIDs retrieved, there are 3,745,848 out-of-prefix objects retrieved using get-bulk operations in trace l06t01, an overhead that can be eliminated by introducing new SNMP operations like GetRange [21].

8 CONCLUSIONS AND FURTHER WORK

It is important to understand how SNMP is used and what the general characteristics of the management traffic are in today's production networks. Such results will be helpful to SNMP researches to correlate their assumptions with "real-life" data or to network vendors in order to better understand the management traffic.

This paper describes several analysis methods that can be applied to our traffic traces. They are meant to give a general overview on the usage characteristics of SNMP and also look deeper into other aspects like table retrieval algorithms and management software implementations.

A new walk extraction script was implemented with support for both get-next and get-bulk requests. It also detects retransmissions and table holes. In case table holes are detected the script tries to analyse the way in which this event is handled by the manager implementation.

A new database schema was introduced to store and further analyse the data extracted by some of the scripts. Using the database, we can extract additional information about the detected walks, such as the relationship between different walk properties (i.e., the number of strict walks among all walks retrieving tables with holes to determine smart manager implementations). Section 5.2.4 gives more queries that can be used to extract data from the database such as the most "usual" starting point for walks. We also use the database to preserve consistency in our graphs by always using the same label to identify a manager or agent within the same trace, even if we re-compile or add new data to our traces.

A new script was created to calculate the response time distribution. Additional and further work could include this information along with other walk parameters such as the number of columns and objects retrieved, to generate artificial SNMP traffic and see how the walk would have performed with some of the parameters modified (eg: increase/decrease the number of max-repetitions in a get-bulk walk).

9 REFERENCES

1. Cisco, Inc. Internetworking Technologies Handbook, ISBN 1587051192, published in 2003.

2. J. Schönwälder. SNMP Traffic Measurements. Internet Draft <draftirtf-nmrg-snmp-measure-01.txt>, January 2007.

3. P. Barford, J. Kline, D. Plonka and A. Ron. A Signal Analysis of Network Traffic Anomalies, Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurements, Marseille, France, pp 71-82, 2002.

4. A. Pras, J. Schönwälder, M. Harvan, J. Schippers, and R. van de Meent, "SNMP Traffic Analysis: Approaches, Tools, and First Results". In 10th IFIP/IEEE International Symposium on Integrated Network Management, Munich, May 2007.

5. M. Harvan and J. Schönwälder. Prefix- and Lexicographical-order preserving IP Address Anonymization. In 10th IEEE/IFIP Network Operations and Management Symposium, Vancouver, Apr. 2006.

6. M. Cheikhrouhou and J. Labetoulle. Efficient Instrumentation of Management Information Models with SNMP. In *Proc. IEEE/IFIP NOMS 2000*, April 2000.

7. J. Schönwälder, A. Pras, C. Ciocov, M. Harvan. "Walk or Crawl". In progress.

8. M. Rose, K. McCloghrie, and J. Davin, "Bulk Table Retrieval with the SNMP," Performance Systems International, Hughes LAN Systems, MIT, RFC 1187, Oct. 1990.

9. Emmanuel Bacry, "Lastwave," – Signal Processing Oriented Command Language. http://www.cmap.polytechnique.fr/~bacry/LastWave.

10. Matlab. http://www.matlab.com.

11. D. Harrington, R. Presuhn, B. Wijnen, "An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks", RFC 3411, Dec. 2002.

12. J. Jung, E. Sit, H. Balakrishnan, R. Morris, "DNS Performance and the Effectiveness of Caching". ACM SIGCOMM Workshop on Internet Measurement (San Francisco, CA, 01–02 November 2001), pages 153–167. ACM Press, 2001.

13. Qingsong Yao, Aijun An. Characterizing Database User's Access Patterns. Database and Expert Systems Applications, 2004.

14. M. Weigle, P. Adurthi, F. Hernandez-Campos, K. Jeffay, F.D. Smith. "Tmix: A Tool for Generating Realistic TCP Application Workloads in ns-2". In ACM SIGCOMM Computer Communications Review, July 2006.

15. N. Brownlee, I. Ziedins. "Response time distributions for global name servers". In Proc. Passive and Active Measurement Workshop (PAM), 2002.

16. NeTraMet website: http://freshmeat.net/projects/netramet.

17. Chen, Peter P., "The Entity-Relationship Model - Toward a Unified View of Data". *ACM Transactions on Database Systems* 1 (1): 9-36, 1976.

18. J. Schönwälder. SNMP Trace Analysis Update (slides). 22 NMRG at 68 IETF, Prague, March 2007.

19. D. Turo, B. Johnson. Improving the visualization of hierarchies with treemaps: Design issues and experimentation. Proc. Visualizations '92, Boston, MA, May 1992.

20. J. Case, R. Mundy, D. Partain, B. Steward. Introduction and Applicability Statements for Internet Standard Management Framework, RFC 3410, December 2002.

21. J. Schönwälder. GetRange Operation for the Simple Network Management Protocol (SNMP). Internet Draft <draft-irtf-nmrg-snmp-getrange-00.txt>, November 2003.