Analysis of SNMP usage in the real world

ASSIGNMENT	:	Bachelor Thesis
COMMITTEE	:	Aiko Pras, Remco van de Meent
DATE	:	February 12th 2007
AUTHOR	:	Jorrit Schippers, s0065498

Summary

The Simple Network Management Protocol was designed to make networks and networked devices manageable using a simple protocol and standardized methods of representing management data.

Until now, research on SNMP was based on some widespread assumptions on the usage of SNMP in practice. What the actual usage patterns are was never the subject of research. Recently, research was started on how SNMP is used in practice and what the typical usage patterns are. This research has currently resulted in file formats and tools that make it easier to store and analyse traces containing SNMP traffic.

This thesis discusses a part of this research on SNMP usage, namely the use of traces to generate statistics about the contents of these traces and draw conclusions from these statistics. In particular, this thesis is targeted at the following questions related to SNMP usage:

- Which versions of SNMP are used in practice?
- Which MIB modules, both standard and enterprise, are used in practice?
- What usage patterns of SNMP can be found in the networks of today?
- What kind of errors occur in practice and what pattern can be seen in these errors?
- Which area of SNMP usage van be researched further?
- How can the SNMP protocol be improved to increase usability?

Generating statistics has been accomplished with the development of a couple of tools. The first tool, SNMP Statistics Generator, parses trace files and generates statistics to help answering the research questions. The other tool, Traffic Type Graph, is able to generate graphs which display what protocols are used in a trace.

These tools have been applied to six traces collected at different kinds of networks. The traces vary in time length from four hours up until ten days. The Traffic Type Graph program has been used to give an impression of the usage patterns of SNMP in these traces.
The SNMP Statistics Generator was used to generate a report containing details about the use of SNMP for each trace.
An area for further research on usage of SNMP was found to be the way in which tables are retrieved in SNMP. The standard defines no way to do this, and analysing how programs and people fetch tables in practise could help to improve these programs and to inform people of optimisations.

The conclusions of the paper are that SNMPv1 and SNMPv2c are the domination versions currently used, as SNMPv3 is not used at all. Furthermore, the general assumption that IF-MIB is the dominating MIB module is correct, and other frequently used modules are SNMPv2-MIB and IP-MIB,. The usage pattern occurring most frequently was the pattern where one manager is periodically polling a lot of agents. Errors did occur in the traces, but were most of the time minor in comparison with the amount of packets. Systematic errors were not found in any of the traces.

An improvement to SNMP could be protocol support for the polling usage pattern. It is used in so many networks, that a kind of subscription mechanism would help to lower the amount of traffic, as requests aren't needed.

Table of contents

1. Introduction

1.1 Motivation

SNMP has been the preferred method for at least a part of network management for more than fifteen years. It is currently being used in all kinds of networked devices and applications, such as switches, routers, printers and servers. Many different vendors have developed tools for SNMP agents and SNMP managers, both open-source and proprietary and these tools are used in numerous kinds of networks, from small to large, in both academic and corporate environments. This creates a situation in which every SNMP network environment is unique.

In spite of these heterogeneous environments, research in the past has made assumptions about SNMP interactions [1], such as the assumptions that SNMP is commonly used just to receive statistics and that SNMPv3 is not used in practice compared to earlier versions 1 and 2c.

No research has been done towards charting the patterns of modern day SNMP traffic, while it in fact is a necessary step in the process of improving the protocol and research methods.

The research outlined in [1] is trying to find out how SNMP is used and that the typical usage patterns are. This research is still continuing and has so far resulted in file formats for describing SNMP traces and tools to convert raw packet traces to these file formats.

This thesis uses these tools to find answers to the research questions which are identified and elaborated in section 1.2.

The work reported in this paper was supported in part by the EC IST-EMANICS Network of Excellence (#26854) and the SURFnet Gigaport Research on Networking (RoN) project.

1.2 Research questions

This thesis uses traces captured in networks to analyse the use of SNMP in these operational environments. This thesis will focus on a couple of many possible questions, and in particular the following questions will be answered:

1.2.1 Which versions of SNMP and protocol operations are used in practice?

The first research questions which has to be answered when analysing traces is to find out which protocol versions and operations are used, as well as the number of variable bindings used in a request. This data can then be used to determine which kind of optimizations can be made to the tools used for management.

For instance large amounts of version one getNext requests using just one variable bind could in some cases be optimized by using version two getBulk requests.

1.2.2 Which MIB modules, both standard and enterprise, are used in practice?

In theoretic research about SNMP assumptions are usually made about the typical contents of SNMP messages. The interface table from IF-MIB is frequently used as a basis for analysis, as it is assumed to be one of the most frequently used data sources in practise. From traces it can be easily deducted if these assumptions are correct. Analysing this will automatically give an overview of the relative amounts of standard, enterprise and experimental traffic.

1.2.3 What usage patterns of SNMP can be found in the networks of today?

Another assumption is that SNMP is usually used to monitor networks by polling all devices at a regular interval. Another interesting analysis subject is the amount of agent interfaces which are generally monitored by one management interface.

1.2.4 What kind of errors occur in practice and what pattern can be seen in these errors?

Tools for network management can provide management at such a level that the underlying SNMP interactions are completely hidden from the manager. Without the manager even knowing, erroneous SNMP queries might be done by the application, incidentally or consistently. Analysis of the errors contained in traces can help to detect errors in applications or configurations. BadValue errors should in principle not occur when both client and server tools interpret MIBs correctly. When such errors occur a targeted debug attempt can be made to correct the error.

1.2.5 Which area of SNMP usage can be researched further?

This thesis is by no means complete on the coverage of all areas of SNMP usage analysis. As a preparation for further research, an area in which more research is necessary is elaborated on.

1.2.6 How can the SNMP protocol be improved to increase usability?

Analysis of SNMP usage doesn't have to just result in conclusions about usage itself, it can also be used to steer developments in the SNMP protocol. After the conclusions related directly to analysis are drawn, one or more improvements to SNMP are suggested.

1.3 Approach

The starting point in this research are the traces which have been collected at different networks and transformed in formats that allow for easy processing. To be able to answer the research questions, these files have to be processed into usable statistics. These generators are not available. The first step in this research is therefore to create applications which are able to produce statistics that are needed to answer the questions, statistics, for instance, about the protocol versions and MIB modules used and the errors that occur in traces.

When these statistics have been gathered, conclusions can be drawn for each trace individually. These conclusions are then aggregated to the general conclusions which are the answers to the research questions.

Chapter two elaborates on the tools which are used in answering these questions.

After that, chapter three will show the results of applying these tools at a number of traces.

Chapter four suggests one area on which SNMP usage can be research further.

1.4 Intended audience

The reader is expected to have affinity with SNMP. This thesis is targeted at people who are interested in the usage of SNMP as it provides information about the research on SNMP usage and elaborates on how statistics can be generated from traces and how conclusions can be drawn from these statistics.

Common terms used in relation to SNMP will not be explained, but less common principles will be explained.

Researchers, SNMP tool developers and network operators were the main reader groups which were taken in mind when writing this thesis, and the results of this research should be of interest to these three groups.

1.5 Related work

A general overview of the research of which this thesis is a part is given by [1]. It contains a long list of specific questions of which the answer should be possible to give at the end of the research. It also introduces alternative trace representation methods, which are also used in this thesis.

Deficiencies in SNMP have also been noted in the past. Difficulties in table retrieval have been acknowledged in [2] and [3]. [2] eventually resulted in the getBulk operation in SNMPv2, which did not completely solve table retrieval problems and [3] was a first step to the founding of the Evolution of SNMP Working Group (EOS-WG).

Two groups have been working on SNMP. One of them, the EOS-WG, part of IETF, has been shut down because of lack of progress in the two years it existed. Several drafts about SNMP improvements never made it to standards and are now obsolete. [4]

The other, the Network Management Research Group (NMRG) of the Internet Research Task Force (IRTF), is a longer lasting research group targeted at exploring new technologies for management of the Internet. [5] The research on SNMP usage has recently been introduced in this group with a positive response. [6]

The tools and results described in this paper will also be used for a paper presented at Integrated Management 2007 [9].

2. Analysis Tools

Research on statistics gathered in real life situations requires tools which are able to process large amounts of data efficiently. Recorded trace files are typically several gigabytes in size but tens of gigabytes is not uncommon. Compressing helps, but just to a certain extent, and it has drawbacks like increased processing time.

Part of the solution is developing specialised file formats which just save just the information which is needed for analysis. These formats are discussed in section 2.1 of this chapter.

These specialised trace files have to be generated out of the normal pcap trace files. Tools for this purpose have been developed at the University of Bremen and are described in this thesis to give a complete view of the toolset. This tool, snmpdump, is described in section 2.2.

Section 2.3 describes the main tool developed for this thesis, SNMP Statistics Generator. This tool parses trace files in one of the specialised file formats to generate statistics in various areas. These statistics will be used to answer some of the research questions defined in section 1.2.

The SNMP Statistics Generator was not appropriate to get a quick overview of the traffic contained in a trace. For this purpose a small script was created which can be used to create a quick overview of the different types of traffic inside a trace file by displaying the number of packets for each protocol against the time in a graph. This tool is described in section 2.4.

Figure 1 shows the relationships between the tools.

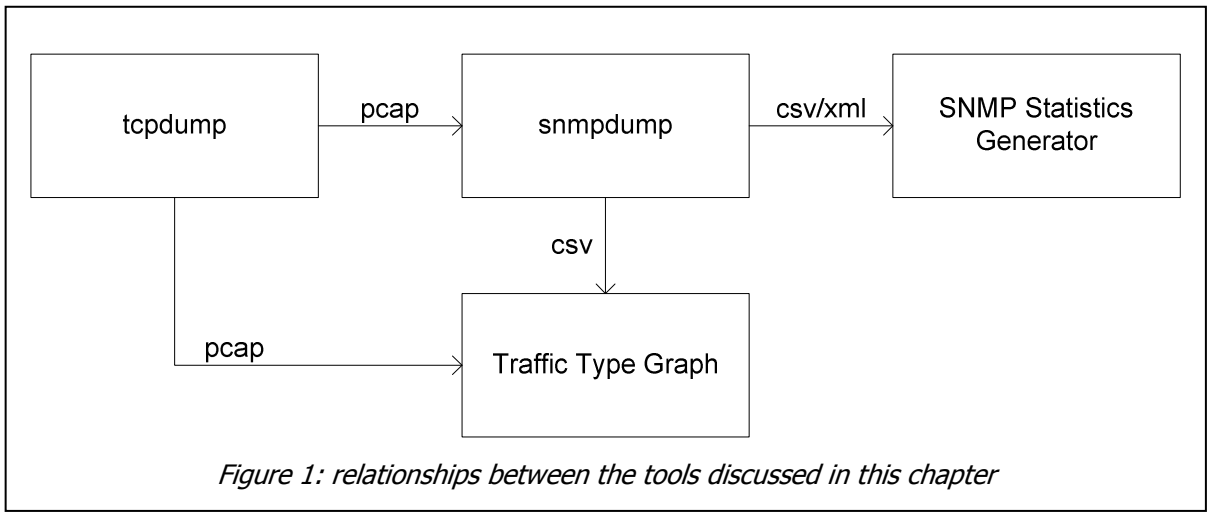


Figure 1: relationships between the tools discussed in this chapter

2.1 Trace representation forms

The current standard storage format for traces containing network traffic is the output of a common library called libpcap, which is used by the popular Unix tool tcpdump (which can also capture traffic other than tcp) [10]. These files contain everything which is transmitted over the wire, also parts that are not important for research on SNMP traffic, like Ethernet, IP and UDP headers.
To be able to analyse large files it was therefore necessary to use a representation targeted at SNMP. This format did not exist before the research on SNMP usage started, which made it an important first step. These formats were available before the research for this thesis started, and are explained below for completeness and are first described in [1].

2.1.1 XML

The first one is XML based. Early assumptions were that given the fact that XML tools are widely spread, it was going to be easy to create analysis programs which used XML as a basis. This appeared to be not the case, because these tools typically can't handle XML files which are several gigabytes in length.

The reason for this is that there are two main categories of XML parsers: DOM based and SAX based. DOM based XML parsers create a object oriented model of the XML tree in memory (hence the limitations) on which a programmer can execute queries like getElementsByTagname and on which advanced XPath queries can be performed.
SAX based parsers parse an XML document in a top down, line by line approach and give the programmer the ability to add hooks when the parser passes the start of an element, a text node, an attribute and so on. This makes SAX memory-efficient and suitable for parsing large files. However, because DOM is easier to program with, most programs use DOM, and the assumption that many programs for analysis could be found appeared to be false.

The disadvantage of XML is that it's quite verbose: fieldnames are repeated twice for each field. Compression of course helps a lot when a lot of repetition is used, but compression increases the processing time of recording and analyzing.

2.1.2 CSV

In many areas where large files containing records with similar structure have to be stored and processed, formats are used using one line per record and a single character as field separator. For instance, log files generated by web servers are typically stored using the W3C log format, which is space separated. For storing SNMP traces the comma separated value (csv) format, which uses comma's as separator, was used.

CSV files are processed in the same way as SAX processes XML files: from the top down, line by line, one line per record. This makes processing the CSV format also memory efficient, as in principle only one record and a pointer to the current record has to be maintained. The advantage of CSV is that the overhead is minimal: there is a single byte separator between every field and each field has to be quoted using one byte in front of the field and one byte after the field, only if the separator appears in the field, otherwise quotation is not needed.

When both are compressed using gzip, the CSV format reduces the file size by about 30%-40%, while using XML increases the file size by approximately this amount.

2.2 Snmpdump

Development on the snmpdump tool was started in the early stage of the research on SNMP usage. It's being developed by Jürgen Schönwälder at the International University of Bremen and it was made to support [1].
For completeness, the program and additional tools provided together with this tool are described in this section. These tools are also described in [9].

2.2.1 The main snmpdump program

Snmpdump is used to convert between the pcap, xml and csv file formats. As the conversion from pcap to xml and csv is lossy, the process can not be reversed, so pcap is only available as input format. However, it is possible to convert between csv and xml and vice versa.

Snmpdump can also manipulate the data while converting. It has basic support for anonymisation, as research on this subject is still progressing [7]. It will try to replace all IP addresses with other IP addresses while preserving the lexicographical order of the address space. The anonymisation functionality is also available separately as a library called libanon and a command line program.

This tool can also, when requested, split the output files into flows according to the sender and receiver information. For every flow between one sender and receiver one output file is created, such that each manager-agent relation can be analysed separately.

In addition to these capabilities, snmpdump can also filter the input file while converting the data. Specific XML attributes or elements can be removed based on a regular expression and pcap files can be filtered using a pcap filter expression when processing pcap files.

2.2.2 Tools distributed with snmpdump

As just a conversion tool is not very useful at itself, some perl applications are provided together with snmpdump. A short overview of these programs will now be given.

- snmpstats.pl
  This script will generate basic statistics about the input csv file, such as protocol versions and operations used.

- snmpflowstats.pl
  This program makes use of the flow separation functionality of snmpdump. It will display statistics like flow duration and number of bytes and messages exchanged in the flow.

- snmpwalks.pl
  This program will extract walk information from csv files. Walks are sequences consecutive getNext or getBulk operations, typically generated by programs like snmpwalk.

- snmpoidstats.pl
  Generates statistics about the object usage in the traces.

2.2.3 Availability

The snmpdump program, anonymisation tools and the perl tools are available at the University of Bremen SVN repository [8].

2.3 SNMP Statistics Generator

To be able to answer the questions listened in section 1.2 statistics have to be generated from the collected traces. SNMP Statistics Generator, written in Java 5.0, provides this functionality and the first part of this section gives an overview of the types of statistics generated by this program, including examples. After that some algorithms used in this program are explained in detail.

2.3.1 Contents of the generated reports

2.3.1.1 Basic statistics

The first part of the output of this program consists of basic statistics like number of packets in the trace, time span covered by the SNMP packets in the trace and time spent by the parser.
After this, a complete breakdown of all packets by protocol, operation and number of variable bindings is given, together with their relative percentages in relation to the total number of packets for that protocol, operation and the total number of packets in the trace. This information will be used to answer the research question presented in section 1.2.1.

An example output is shown in table 1.

The table shows that most traffic is performed using SNMPv1, and a small portion (5%) makes use of SNMPv2. Usually one variable bind is used for both protocols, and it also appears that this environment is making use of traps.

Generator Basic:

version / op.	packets	% of op.	% of version	% of total
v1	827709291	100.00%	100.00%	94.99%
get-request	2849306	100.00%	0.34%	0.33%
1 varbinds	44941	1.58%	0.01%	0.01%
2 varbinds	2804365	98.42%	0.34%	0.32%
get-next-request	411885313	100.00%	49.76%	47.27%
1 varbinds	411883879	100.00%	49.76%	47.27%
4 varbinds	714	0.00%	0.00%	0.00%
8 varbinds	710	0.00%	0.00%	0.00%
13 varbinds	10	0.00%	0.00%	0.00%
set-request	40	100.00%	0.00%	0.00%
1 varbinds	40	100.00%	0.00%	0.00%
response	412973522	100.00%	49.89%	47.39%
1 varbinds	410169158	99.32%	49.55%	47.07%
2 varbinds	2804354	0.68%	0.34%	0.32%
13 varbinds	10	0.00%	0.00%	0.00%
trap	1110	100.00%	0.00%	0.00%
0 varbinds	1110	100.00%	0.00%	0.00%
v2	43652053	100.00%	100.00%	5.01%
get-next-request	21826031	100.00%	50.00%	2.50%
1 varbinds	21826031	100.00%	50.00%	2.50%
response	21826022	100.00%	50.00%	2.50%
1 varbinds	21826022	100.00%	50.00%	2.50%
total	871361344	100.00%	100.00%	100.00%

Table 1: Basic generator output for trace l03t02

2.3.1.2 Replies containing errors

Next, an overview of the errors in the trace is given. The errors are indicated in the response, while it is interesting to know for which request type the error was generated. This is difficult to retrieve, because often requests and responses are stored separately in the traces. The statistics generator contains a lookup buffer which translates from request id to request type, but this buffer can't be of infinite size because of memory and speed constraints. Therefore the output might contain question marks for responses containing an error for which no request could be found.
The statistics indicate the kind of error, the version of the protocol used, the type of request, the OID to which the error index in the response was pointing and the number of occurrences. These statistics will be used to answer the research question identified in section 1.2.4.

An example is given in Table 2.

In those lines where two question marks appear instead of an operation type the program failed to find the corresponding request in the request buffer, as explained in section 2.3.3.1.

The whole trace contained roughly 125 million responses, so a total of 159 erroneous responses is a really low number, but it still might indicate a configuration error, for instance in communication with one particular device or one specific MIB module.

A certain Cisco MIB module (enterprise number 9) seems to be the target of many failed requests.

```
Generator Errors:
error / packets                                          |  packets
---------------------------------------------------------+--------
noSuchName                                               |      155
  v1 get-request (0) 1.3.6.1.4.1.9.2.10.1.0              |        4
  v1 get-request (0) 1.3.6.1.4.1.9.9.47.1.1.1.0          |        6
  v1 ?? (-1) 1.3.6.1.4.1.9.10.62.1.2.1.3.0               |        4
  v1 set-request (3) 1.1.0                               |        6
  v1 ?? (-1) 1.3.6.1.4.1.9.9.171.1.1.1.0                 |        4
  v1 get-request (0) 1.3.6.1.4.1.9.3.6.3.0               |        1
  v1 ?? (-1) 1.3.6.1.4.1.9.10.24.1.1.1.0                 |        4
  v1 get-request (0) 1.3.6.1.4.1.9.9.171.1.1.1.0         |        2
  v1 get-request (0) 1.3.6.1.4.1.9.10.62.1.2.1.5.0       |        2
  v1 get-request (0) 1.3.6.1.4.1.9.10.24.1.1.1.0         |        4
  v1 get-request (0) 1.3.6.1.4.1.9.2.10.4.0              |        4
  v1 ?? (-1) 1.3.6.1.4.1.9.9.42.1.1.1.0                  |        4
  v1 get-request (0) 1.3.6.1.4.1.9.2.10.5.0              |        4
  v1 get-request (0) 1.3.6.1.2.1.47.1.1.1.1.11.1         |       97
  v1 get-request (0) 1.3.6.1.4.1.9.2.10.2.0              |        4
  v1 get-request (0) 1.3.6.1.4.1.9.9.42.1.1.1.0          |        2
  v1 ?? (-1) 1.3.6.1.4.1.9.9.47.1.1.1.0                  |        3
badValue                                                 |        4
  v1 set-request (3) 1.3.6.1.4.1.9.2.9.9.0               |        4
---------------------------------------------------------+--------
total                                                    |      159
```

Table 2: Error generator output for trace l02t01

2.3.1.3 Enterprise module usage

The next overview gives insight in the types of enterprise MIB modules used in the trace. The number of packets as well as the percentage of the total number of packets is given for each enterprise encountered in the packet.

Table 3 shows the output of this generator for a location where APC power management devices are being managed using SNMP.

```
Generator Enterprises:
8990 requests found, containing 8990 oids, 1 different enterprises found

                                    Enterprise |     Oids |      Perc
  ---------------------------------------------+----------+----------
          American Power Conversion Corp. (318) |     2869 |    31.91%
  ---------------------------------------------+----------+----------
                        total enterprise traffic |     2869 |    31.91%
```

Table 3: Enterprises generator output for trace l04t01

2.3.1.4 MIB module usage

After this a breakdown per MIB is given. All packets, from the enterprise and the standard MIB modules, are ordered by MIB as well as possible, and just like the enterprise breakdown the total number of packets and the percentages are given. Together with the enterprise breakdown this part gives information to answer the question defined in section 1.2.2.

Table 4 gives a typical output of this generator.

Besides the usage of IF-MIB, IP-MIB and SNMPv2-MIB, three very common MIBs in network management, also BRIDGE-MB and BGP4-MIB seem to be used, although the BRIDGE-MIB traffic is negligible.

RFC1213-MIB and RFC1155-MIB show up because these MIBs were loaded in the MIB repository and they catch all OID's which can't be fit in MIB's with longer OID bases. Adding more MIB's to the repository, for instance MIB's of the enterprises found in this trace, will make the report more complete.

```
Generator Mibs:
25893575 requests found, containing 27173916 oids, 7 different mibs
found, 0 oids with unknown mibs found

                         MIB |     Oids |      Perc
  ---------------------------+----------+----------
                RFC1213-MIB |  2803284 |    10.32%
                RFC1155-SMI |  8596255 |    31.63%
                     IF-MIB | 10955091 |    40.31%
                     IP-MIB |    38052 |     0.14%
                   BGP4-MIB |  4769149 |    17.55%
                 SNMPv2-MIB |    11746 |     0.04%
                 BRIDGE-MIB |      339 |     0.00%
  ---------------------------+----------+----------
           total known mibs | 27173916 |   100.00%
         total unknown mibs |        0 |     0.00%
  ---------------------------+----------+----------
             total all mibs | 27173916 |   100.00%
```

Table 4: MIBs generator output for tracé l01t02

2.3.1.5 Manager – agent relations

The remaining part of the report is filled with two tables with information about the manager-agent relationships.

The first table shows the relationship between manager and agent as seen from the manager perspective, with for each manager a list of agents it manages and the number of packets sent between them, with separate columns for incoming and outgoing packets, as well as columns showing the percentages of the traffic for the manager and of the total traffic for this specific manager-agent relation. The second table shows the number of managers for each agent. This information is useful for answering the research question outlined in section 1.2.3.

Table 5 lists a couple of rows from the output of this table.

```
Generator Associations:
Number of managers: 2
Number of agents: 169

 manager / agent             | ma->ag | ag->ma | % of man. | % of tot.
-----------------------------+--------+--------+-----------+----------
  [154.163.254.2]:1024 (132)|  38765 |  38595 |   100.00% |     0.68%
    [154.140.1.2]:161        |      6 |      0 |     0.01% |     0.00%
    [154.163.132.18]:161     |    153 |    153 |     0.40% |     0.00%
    [154.163.132.19]:161     |    207 |    207 |     0.54% |     0.00%
    [154.163.132.20]:161     |    191 |    191 |     0.49% |     0.00%
    [154.163.132.21]:161     |    199 |    199 |     0.51% |     0.00%
    [154.163.132.22]:161     |    189 |    189 |     0.49% |     0.00%
    [154.163.132.23]:161     |    187 |    187 |     0.48% |     0.00%
    [154.163.132.36]:161     |    363 |    363 |     0.94% |     0.01%

agent                          | managers
-------------------------------+--------
  [154.140.1.2]:161            |        1
  [154.163.127.2]:161          |        1
  [154.163.127.3]:161          |        1
  [154.163.132.18]:161         |        2
  [154.163.132.19]:161         |        2
  [154.163.132.20]:161         |        2
```

Table 5: Part of the associations generator output for trace l01t03

2.3.2 Program structure

The program is a framework based around two concepts: parsers and generators. Parsers interpret the file formats and convert this information to an internal representation of the SNMP packet. Currently there are parsers for the XML and the CSV format, which seems to be sufficient for the time being.

The generators analyze each packet and generate statistics by updating their internal state for every packet which is analyzed.

The main program accepts multiple filenames as argument, as well as standard input if the filename - is given. Gzip compressed input is also accepted.

For every file the file type is determined and the appropriate parser is called with both the file and a list of generators. The parser will then parse the file and for every packet it reads call each of the generators in the list.

At the end, the main program calls each generator to get a textual representation of the statistics.

2.3.3 Algorithms

A couple of algorithms were invented to make the program work. They are discussed in this section.

2.3.3.1 Request buffer

The Errors generator, which generates statistics about the errors encountered in the traces, needs a buffer to lookup the type of the request so errors encountered in responses can be linked with the original request.

This buffer is implemented using one array and a Map. The array defines the order of the items stored in the buffer, so the first item is removed when the buffer is full and a new item is added. The Map is used for looking up requests based on source ip and port, target ip and port and request id, resulting in the type of the request.

2.3.3.2 OID ordering

The GetNext visualizer needs to sort the OIDs it encounters in lexographical order. This means that OID 1.2.3 has to come before 1.2.3.0, and after 1.2.2.2 for instance.

In Java this is done by defining a custom Comparator and using that comparator to order items in a list.

The comparator used will split both OIDs into an array of integers. After that each integer of the OID with the smallest number of integers is compared with the integer of the other OID at the same position. If they are not equal then this difference decides the order of both OIDs. If after checking all integers no difference is found the length is compared. If both OIDs have an equal length, and all integers are the same, the OIDs are the same as well. If the number of integers is different, but all integers of the shortest OID are equal to the integers of the longest OID, then the shortest OID has to come before the longest OID.

2.3.4 Availability

The SNMP Statistics Generator is available for download at [11].

2.4 Traffic Type Graph

During this research we have worked with a lot of traces, and there became a need for an easy way of displaying an overview of the contents of a trace. In particular there was an interest to easily view the following properties:

- The amount of traffic versus time.
- The types of management traffic in the trace, because some traces which have been given for research contain other kinds of traffic besides SNMP traffic. To get a general view of the kind of management activities occur in the trace the graph should separate common types of management protocols, like SNMP, Telnet, SSH, HTTP and syslog.

For this purpose a small script has been made. It's written in PHP and it uses common unix tools tcpdump and gnuplot to for reading files and writing graphs. It reads both pcap and csv files. An example output containing all traffic types is shown in figure 2.

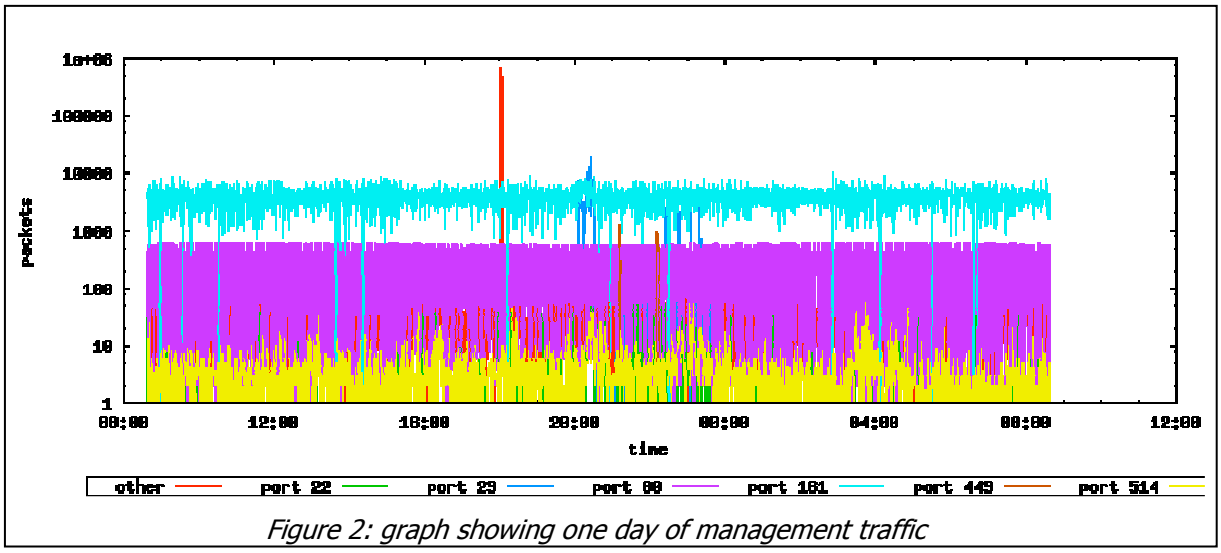


Figure 2: graph showing one day of management traffic

Please take note that the y-axis ha a logarithmic scale, which means that although there is syslog (port 514) traffic, it's minor compared to HTTP (port 80) and SNMP (port 161) traffic. SSH (port 22) and Telnet (port 23) seem to be used only incidentally. The advantage of using gnuplot is that based on the same data files graphs can be enlarged and smaller time intervals can be selected so a better view of the traffic can be given.

The number of packets at the y-axis is the number of packets that were available in the trace in the interval of thirty seconds.

The source of this program is available in the appendix to this thesis and on [11].

3. Analysis Results

With all necessary tools being introduced, the actual measurement can now take place.
In this chapter several traces will be examined and the results will be analyzed and explained.
The first section contains a description of the several traces and locations where these traces were captured.
After this, highlights of the results of the programs listed in chapter two will be listed for these traces.
The chapter is finalized by discussing these results.

3.1 Trace description

Network operators are logically very conservative when it comes to sharing data from their network, especially if it's not only header information, but also complete (SNMP) packets. The first traces which are examined are captured at networks related to universities, as these ones were the easiest to get.

The locations where the traces were captured are:

1. Dutch inter-university network
2. Dutch university network
3. Dutch faculty network
4. Hosting provider network

The length, contents, capture dates and sizes of the traces vary from location to location. Important properties of the traces are listed in table 1. The column contents indicates whether the trace contained only SNMP packets, or that a complete capture of the management traffic was available. The size column lists the uncompressed size of the CSV file containing the SNMP traffic.

For identifying the traces a naming convention is used where the first number indicates the location and the second number indicates the trace number. Trace l02t01 is the first trace recorded at location 2.

Trace	Length	Contents	Date	Size [MB]
l01t01	1 day	Management	July 2005	24
l01t02	7 days	Management	July 2005	6369
l01t03	1 day	Management	April 2006	1386
l02t01	10 days	Only SNMP	April 2006	77789
l03t02	7 days	Only SNMP	April 2006	130858
l04t01	4 hours	Only SNMP	April 2006	24

Table 1: trace contents

3.2 Results of the programs

This section only gives the output of the Traffic Type Graph program. The SNMP Statistics Generator generates a lot of information and it's impractical to list the results for every trace in this chapter. [11] contains all outputs from this program.

The output of Traffic Type Graph consists of graphs showing the number of packets as function of the time for all common management protocols.

Traffic is distinguished by it's port number. Port 22 and 23 are used for remote shell (secure and unsecure), port 80 and 443 for HTTP traffic (insecure and secure) port 514 is used for remote syslog and port 161 and 162 are used for SNMP (normal traffic and traps).

The graphs below are generated with a 30 second peak interval.

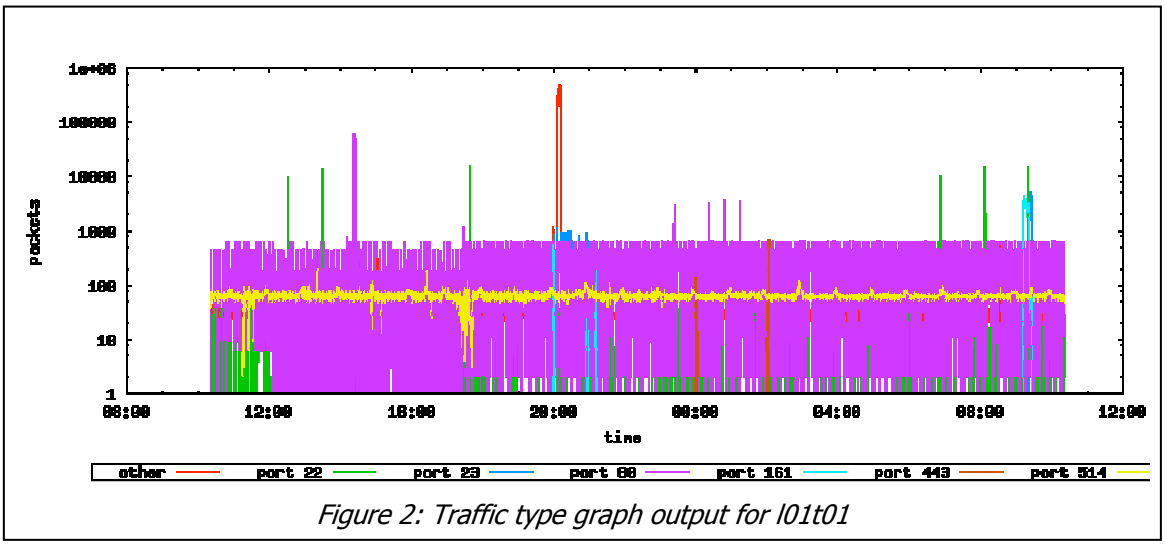


Figure 2: Traffic type graph output for l01t01

Figure 2 shows that at that location and at that moment SNMP was not used systematically, at least not with a common polling interval like five or fifteen minutes. HTTP and syslog, however, appear to be used for certain polling or systematic reporting activities, as these protocols show a constant usage level.

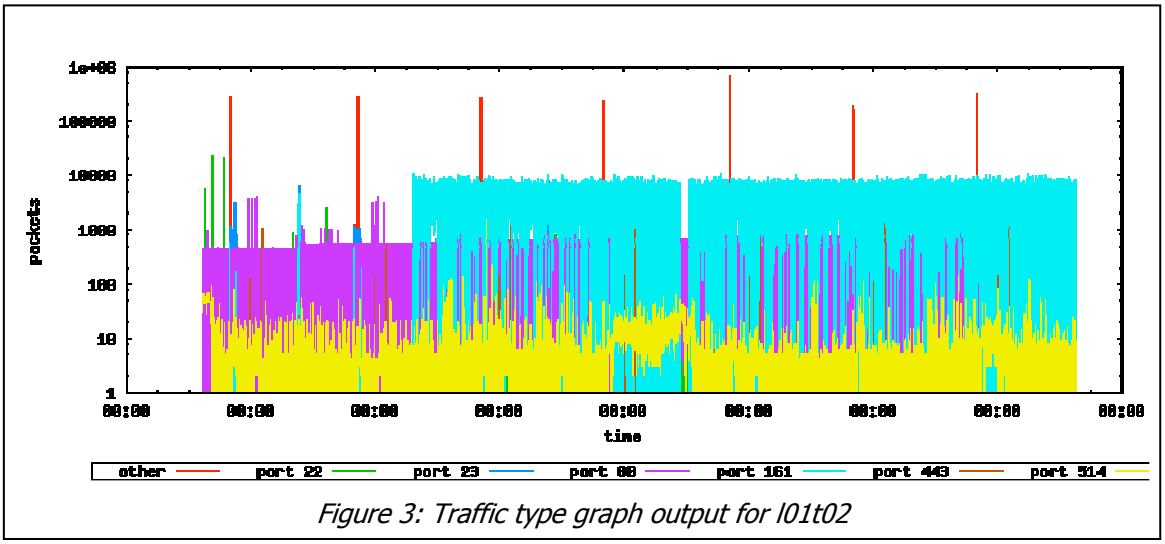


Figure 3: Traffic type graph output for l01t02

Figure 3 shows the same location at a different time. SNMP appears to be used in a polling fashion, but only after one and a half day after recording started. The other protocols seem to be used similarly compared to trace l01t01.

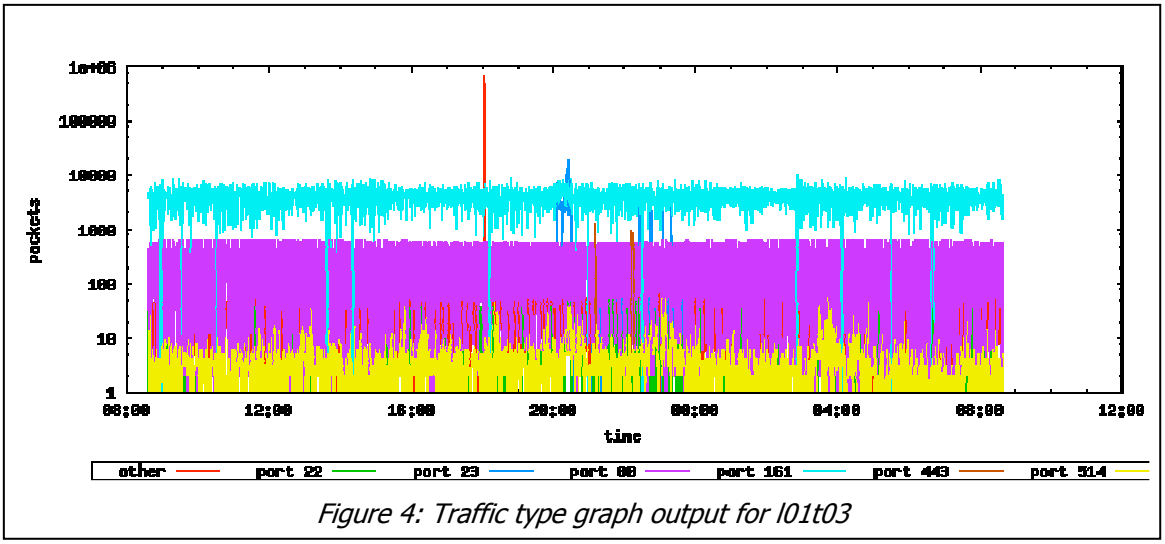


Figure 4: Traffic type graph output for l01t03

Figure 4 is generated from a trace from the same location as figures 8 and 7, only now recorded one year after those two traces. SNMP is now used to constantly poll devices. Meanwhile, HTTP is still used to perform repeating tasks as well, apparently SNMP is not able to replace this, or the other way around, as this HTTP traffic might be generated by web service management solutions.

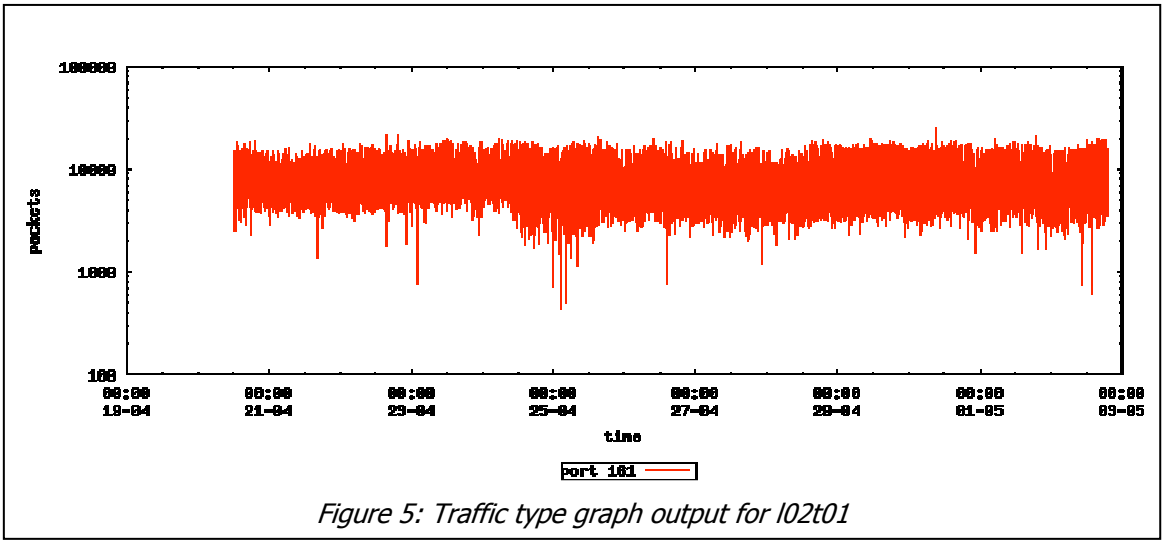


Figure 5: Traffic type graph output for l02t01

The traces supplied by locations 2 and up contain only SNMP traffic. The resulting graph for trace 1 of location 2 is shown in figure 5. The trace shows a typical polling behaviour. The amount of traffic, however, is not very constant.

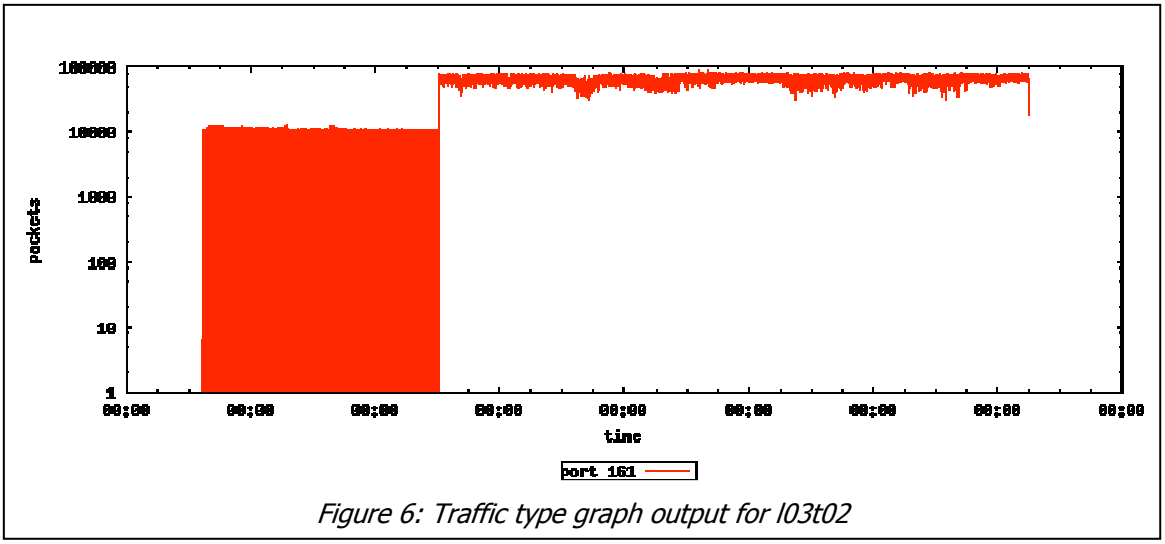


Figure 6: Traffic type graph output for l03t02

Figure 6 shows the output for trace 1, location 3. It also shows typical polling behaviour, but the interesting observation in this trace is the fact that at the third day the pattern changes. A cause could be a changed polling interval. In that case, the polling interval is shorter than in the first part of the graph, as the graph doesn't go back to zero anymore.

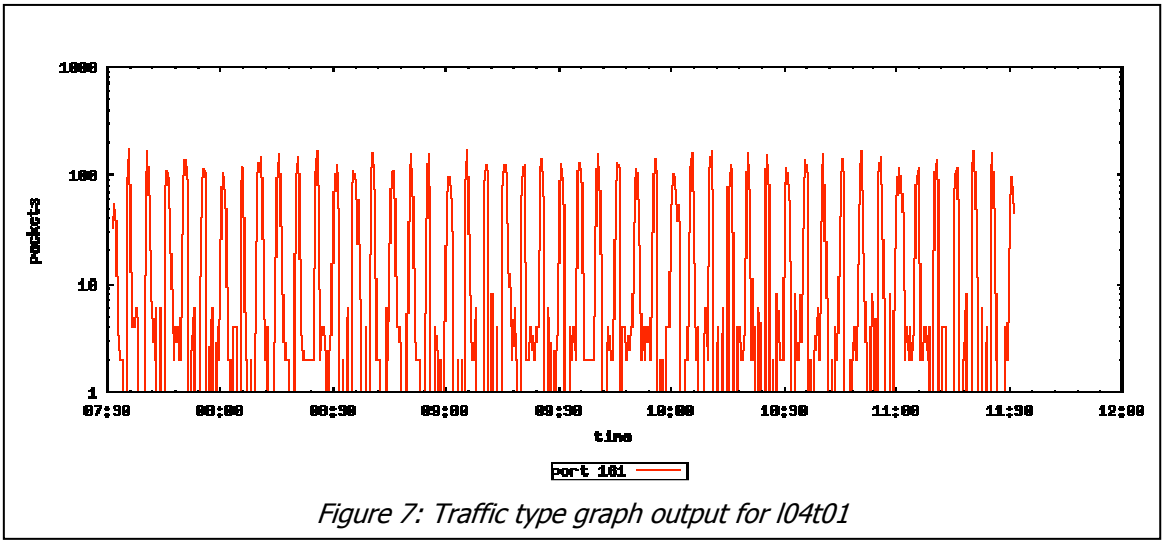


Figure 7: Traffic type graph output for l04t01

The last trace is much shorter than the earlier traces, and the graph in figure 7 is therefore more detailed than the other graphs. A polling behaviour is also observed in this trace.

3.2.1 SNMP Statistics Generator

The SNMP statistics generator generates a lot of information and it's impractical to list the results for every trace in this chapter. Appendix A lists all outputs from this program.

3.3 Discussion of the results

Unfortunately there were only six traces available for analysis. The images generated by the traffic type graph show however that the SNMP behaviour is quite different for each trace. There are even differences between the traces recorded at the same location.

In this section each trace will be examined and for each trace conclusions will be drawn according to the output of the programs. Discussing each trace separately will show how the results of the programs can be explained and how this output gives insight in the structure of the network where these traces were captured.

3.3.1 Trace l01t01

This trace contains only sporadic SNMP traffic, as the traffic type graph shows. In this part of the network apparently no SNMP polling was done at the time this trace was recorded, while the graph shows that there is constant HTTP and syslog traffic.

When looking at the SNMP traffic, it appears that only SNMPv1 is used. Most requests are getNext requests, and SNMP appears to be not only used for retrieving values, as there are also set requests. Multiple variable bindings per request are being used, which is in the case of getNext requests an indication that an efficient way of table retrieving is used.

When looking at the errors contained in this trace, it appears that all set requests have failed because an invalid value was given. The set request is done at SNMPv2-MIB::sysLocation, which can contain a String. When looking at the number of agents in the associations generator output it appears that manager 145.145.254.2 has the same number of agents as there are (failed) set operations. This would suggest that a one-time set operation was performed at all agents, and failed. All other errors occur in get requests for objects in Cisco MIB modules.

The MIB/enterprise usage in this trace remains unclear. One third of the traffic is related to Cisco, one fifth to IF-MIB and forty percent of the traffic is contributed to the general RFC1213-MIB.

When looking at the last part of the output there is one interesting fact: besides the manager mentioned before, there is another manager which accessed seven agents which weren't accessed by the first manager. It sent six messages to each agent, but received no replies, which seems like a problem which has to be solved.

3.3.2 Trace l01t02

The SNMP pattern in this trace is also non standard. In the first part of the trace only incidental SNMP traffic was captured. After one and a half day suddenly constant SNMP traffic is measured, with a small pause around noon at the fifth day.

When looking at the contents of the trace, similarities between this trace and the first trace of this location can be seen. The same MIB modules and enterprises are used, but now IF-MIB is used much more than in the first trace. IF-MIB is a MIB which is usually monitored by polling, so this could relate to the fact that polling seems to be used in this trace.

Again all errors occur in the Cisco MIB, and all set requests fail with a badValue response for sysLocation.

This trace contains the same manager-agent relations as the first trace, with the exception that there is another manager, responsible for 99% of the traffic. This manager is probably the manager performing the polling operations. In this trace we also see that some agents are not responding, and that most agents are both contacted by the manager from trace 1 and the polling manager, which was not available in trace 1.

3.3.3 Trace l01t03

This trace is recorded one year after the first two traces of this location, which is visible from the fact that the erroneous set operations from the previous traces are not present in this trace. Besides that, some SNMPv2 traffic is recorded, although the amount is negligible compared to the amount of SNMPv1 traffic. SNMPv2 appears to be only used for set operations, and SNMPv1 traffic just for get operations. The set operations are probably manual operations performed by operators who used SNMPv2 tools.

SNMP is used in this trace clearly for polling, as there is a constant amount of SNMP traffic present in the trace. HTTP and syslog are the other major components in this trace.

There are quite some errors recorded in this trace, which are almost entirely caused by get request for non-existent rows in ENTITY-MIB. Other errors are related to operations on Cisco MIB modules.

The list of used enterprises and MIB modules and the distribution is comparable to the contents of trace 2, so apparently the polling strategy and the hardware haven't changed in the meantime.

The trace contains two managers, which both handle more or less the same number of agents, with overlap.

3.3.4 Trace l02t01

This trace only contains SNMP traffic, and the graph shows a constant level of traffic. From the detailed report it is immediately clear that this is a different network than location 2.

First of all, SNMPv2 is the dominant version used in this trace, and SNMPv1 has only a minor share in the total number of packets. Moreover the efficient getBulk request operation is frequently used, which indicates the use of optimized tools.

The number of errors is small, but appear all to be caused by operations on Cisco MIB modules.

The number of enterprise specific MIB modules used in this trace is larger than location 1, but the MIB modules are not used as frequently as in location 1, as IF-MIB is by far the most requested MIB modulewith 93.5% traffic, followed by BRIDGE-MIB with 5% traffic.

A total of six managers is active in this trace, of which one manager is acting as a trap receiver. Most agents are being monitored by one manager (excluding the manager traps are sent to) and some are monitored by two managers.

3.3.5 Trace l03t02

The graph shows that during the recording of the trace the polling pattern has changed. The first two days the gaps between the peaks of traffic are large, such that the graph returns to zero between the peaks. After this period not only the number of packets per seconds increases, but also the interval deceased, which causes just a single line indicating the traffic volume.

The trace contains both SNMPv1 and SNMPv2 traffic, with SNMPv2 only accounting for 5% of the total traffic. Most operations are done using the getNext operation, which suggests that there is room for improvement using the getBulk operation.

Errors in this trace are in contrast to previous traces not related to just a couple of MIB modules, but are spread across a large number of modules.

A couple of enterprises is used, but more surprising is the number of MIB modules used in this trace. RMON-MIB, which did not appear in previous traces, is by far the most prominent MIB in this trace, with 94% traffic. BRIDGE-MIB and IF-MIB account for 4% and 2% of the total traffic. All other MIB modules are used incidentally: at least ten other modules are used, but all of them combined account for just 0.1% of the total amount of traffic.

This trace contains 27 managers, of which a lot have just contact with one and the same agent and two of them appear to be trap receivers.

3.3.6 Trace l04t01

Unfortunately this trace is rather short, but it does give insight in the network where this trace was captured.

This network uses both SNMPv1 and SNMPv2, which is not the first case of this combination. But what's new is that both versions are used significantly, neither is dominating. SNMPv1 is used for get and getNext operations, both with just one variable binding, which is usually not very efficient. SNMPv2 is used more efficiently, with a dominance of the getBulk operation and responses containing between 55 and 74 varbinds per response.

This company appears to manage it's power infrastructure using SNMP, as APC's PowerNet-MIB accounts for 31% of the total traffic. SNMPv2-MIB and IF-MIB take up the 6.5% and 61.5%, which looks like a typical polling configuration.

When looking at the managers and agents, it appears that while there are three different managers, no agent is in contact with more than one manager, so we can conclude that this network contains three separated SNMP management areas, possibly for different kinds of devices (power switches, network switches and routers).

4. Area for further analysis

This chapter elaborates on one particular area in which further research on SNMP usage is needed to get a better understanding of the usage of SNMP in practice. The specific area discussed in this chapter is table retrieval. Section 4.1 defines two general ways in which tables can be retrieved in SNMP and why both these methods are not optimal. Section 4.2 explains which specific topics related to table retrieval can be researched.

4.1 Table retrieval methods

SNMP doesn't provide a specific way of fetching tables or parts of tables. You have to use getNext or getBulk to fetch the cells, and this can be done in two ways:

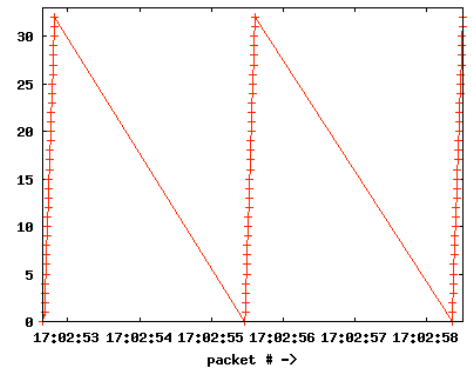
- Column by column: the manager starts by fetching the first cell of the first column and row, and stops when it arrives at the last cell of the last cell in the last row.
- Row by row: the manager requests a complete row per request.

4.1.1 Column by column

Usually managers use the getNext operation on the Object Identifier (OID) of the table to fetch the first cell of the first row, and use the resulting OID to continue to walk through the table. Walking through a table like this results in fetching the table column by column, as the least significant portion of an object identifier of a cell is the identifier of the row. After the walk mechanism has reached the end of the first column, increasing the object identifier results in the object identifier for the first cell of the second column. This continues until the table is completely fetched. This method is called column-by-column, because in this way column one is fetched first, then column two, and so on.

This is not optimal, because you need one operation for each cell. You have to wait for the reply to determine the OID you need as an argument for the next request.

Figure 8 shows typical column by column behaviour using getNext requests. Every cross represents one request, and the ordering on the vertical axis represents the relative lexigraphical ordering between the OID's in the request. The graph represents three consecutive request groups, each with requests to the same row of OID's.



This problem can be partially solved by using the getBulk operation. For some tables the number of rows, and thus the number of OID's, can be determined before fetching the table, because it's provided as a separate value. ifNumber is such a value, and it helps to give the right argument to the getBulk operation. The number of interfaces is also unlikely to change frequently, so in this case getBulk can be efficient.

In other cases this isn't as easy, for instance when fetching the dot1dTpFdbTable in the BRIDGE-MIB, which contains the hardware (MAC) addresses in the MAC table of a network bridge. In this case, the number of rows is not given by a value outside the table, and the number is likely to change a

lot, as entries are dynamically added and deleted. When the number of OID's requested with the getBulk operation is larger than the actual amount of cells, unnecessary data is transmitted. When the argument of the getBulk operation is too low, additional requests are needed to get the complete table.

4.1.2 Row by row

The statistics generated in chapter three show that in many traces getNext operations are usually performed with only one argument. When requesting tables, requesting multiple variable bindings per getNext can be used to fetch one row per response. To do this, the manager needs to transmit a getNext request with all the OID's of the columns he wants to receive, which is the OID of the table appended with the number of the column. The response will contain the cells of the first row. The OID's from this response can be used in the request for the next row, until the response contains values other than table cells, which indicates that the end of the table is reached.

Because in this situation the data is received per row, this method is called row by row.

This can also be implemented by using getBulks, but again the efficiency gain depends on the capacity to know the number of rows beforehand. If you want to receive just a few columns from a wide table, you can use a getBulk request on just those columns, instead of using a getBulk response on the complete table.

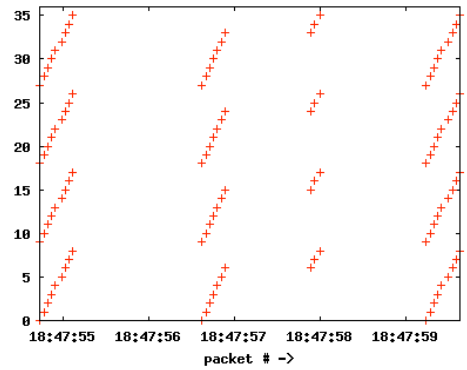


Figure 9: Typical row by row behaviour using getNext requests

Figure 9 shows typical row by row behaviour using getNext requests. Every cross represents one OID in a getNext request. Every getNext request contains four OID's, as the graph shows. The gap in the second row is caused by a retransmission.

4.2 Points of research

There are a number of questions which remain unanswered regarding SNMP tables. These questions can be divided in four categories:

- Table fetch methods.
  In the first chapter a two types of table fetching were identified, and investigation should be done to the preference of managers to use the one method or the other, and which method would have been the best in that situation.
- Black holes are gaps in tables.
  We would like to know if they occur in practice, so tool builders would have to take care of these situations.
- Table starting points.
  A manager can choose between several starting points for fetching a table.
- Standard objects in getBulk
  getBulk can be used to fetch single objects in the same request as the sequences of objects which can replace getNext requests.

4.2.1 Table fetch methods

In the first chapter a distinction has been made between the column by column and row by row table fetching method. Because column by column using getNext is the easiest way of walking through an SNMP tree, it's usually assumed that most managers use this method.

Researchers think that row by row is faster, but this hasn't been confirmed or tested. A possible research goal could be on the one hand checking which methods are being used, and on the other hand checking which is the most efficient.

Specific questions for which an answer has to be found by analysing traces are:

- Which method of table fetching is used?
- Which operation is used?
  Particularly the finding of getNext operations on SNMP version 2 is interesting, because these situations can almost certainly be optimized.
- Which method is the fastest?
- Which method is the most bandwidth efficient?

The results of this research can be useful for toolbuilders and network operators, because improvements in response time and bandwidth usage can allow larger networks to be monitored with less resources or better than before, for instance with smaller polling intervals.

4.2.2 Black holes

The table fetch methods were explained under the assumption that all cells of a row are either all available or all unavailable. It might be the case that this is not true. When using the column by column method, a row might not exist anymore when the agent tries to fetch the next column. Access restrictions might also be the cause of some cells being unavailable to the manager, while other cells in the same column and row are still available.

In bad implementations, this might cause wrong data because the manager things the data in the next cell belongs to the removed row.

Figure 10 shows this. The cell with the red X is inaccessible, for whatever reason. If the manager would assume that the table fetch operation is atomic, it would assume that the value associated with the index 5.2 is 2, which is probably not true. For all other rows it would take the wrong value too, and at the last point in the table the error might be recognized, or even worse, a value from outside the table might be chosen. This cell has been marked with a question mark.

dest(1)	policy(2)	next(3)
2	1	2
3	1	3
5	1	2
5	2	X
7	1	2
8	1	2
9	1	2
		?

Figure 10: black hole. The green rows are the rows that are perceived by the manager

We would like to know if these situations happen in reality, and if managers can cope with them.

Note that the case where rows are deleted while the manager is still busy fetching the columns is interchangeable with the case where rows are added while the manager has already read one or more columns. In both cases partial rows are returned.

4.2.3 Table starting points

When fetching tables in a column by column fashion, there are a couple of options available for the starting point for the walk. You could take the OID of the table (tableOid), the OID of the entry (tableOid.1) or the OID of the first column (tableOid.1.1). tableOid and tableOid.1 are usually not-accessible, and therefore these requests are all the same.

This is one of the properties which can be used to fingerprint managers or management software, and might therefore be a security risk, although a small and theoretic one.

From informal contacts I have learned that Net-SNMP under some circumstances adds .0 to starting points for snmpwalks.

4.2.4 Standard objects in getBulk

getBulk operations have two functions. The first one has already been described, namely the ability to give an OID and the number of consecutive values a manager would like to receive, starting with the given OID. The second property is to request one or more non-repeating values, equal to a get request on that OID. When fetching a table, this could for instance be used to fetch the sysUptime, which stores the time for which the agent has been running, in the same request. This is also an aspect which could be used to fingerprint managers or management software.

5. Conclusion

This section will finalize the thesis by answering the research questions posed in section 1.2 using the traces introduced in section 3.1 and discussed in section 3.3. The number of traces examined in this thesis is not even close to the number of traces necessary to answer the research questions in general, but things can be said in response to these questions when focussed just on the traces discussed in this thesis.

5.1 SNMP versions and operations used in practice

The research question identified in section 1.2.1 is about the usage of different SNMP versions and operations in practice.

According to the analyzed traces, no location was using SNMPv3, and when looking at the usage of SNMPv2 compared to SNMPv1, we see in section 4.3 that at location one SNMPv2 hasn't penetrated at all. Location two seems to have moved to version two almost completely, and moreover, is actually using the new features of that protocol: getNext doesn't appear at all, and all traffic is generated by either get or getBulk requests.
Location three shows only a minor amount of SNMP version two traffic, while location four is still using SNMP version one for a significant amount of traffic.
After these observations it can be concluded that SNMPv3, including it's encryption and authentication features, is not used at all, and that a significant amount of SNMP interactions are still performed using SNMPv1.

5.2 MIB modules and enterprise modules used in practice

The research question listed in section 1.2.2 asks which MIB modules and modules made by which enterprises are used in practice.

When we aggregate the statistics from all traces, it can be seen that the IF-MIB is the most popular MIB: in almost all traces it was one of the major MIBs used, only location three did not use IF-MIB as extensive as other locations. SNMPv2-MIB and IP-MIB are also used in most of the traces, but play only a minor role. BGP4-MIB, RMON-MIB and BRIDGE-MIB are major MIBs in just a couple of traces. Other MIBs only play a minor role in some traces.
Of all enterprises encountered in the traces only MIBs designed by Cisco and APC generate a substantial amount of traffic. Cisco is used by all locations except location 4 and APC is used only by location 4, but plays an important role at this location.

5.3 Usage patterns of SNMP in current networks

The research question mentioned in section 1.2.3 is: what usage patterns are employed in the networks of today?

From the traces examined in this thesis it can be concluded that the main usage pattern is one manager polling a large number of agents. Only in the case of location three we can clearly see multiple managers, as explained in section 4.3.5.

5.4 Errors and error patterns encountered in traces

The research question posed in section 1.2.4 is: What kind of errors occur in practice and what pattern can be seen in these errors?

All traces except the trace recorded at location four contain errors. The percentage of errors compared to the number of packets ranges from 2% (l01t01) to 0,5 ppm (l03t01). All errors associated with Get operations (including GetNext) are NoSuchName errors, and erroneous SetRequest operations (which only occur at location one) are of type BadValue. At location one errors were typically caused by the Cisco enterprise MIBs, but at other locations different MIBs cause errors.
The low number of errors suggests that errors are incidental and that the polling done in most of the traces does not cause repetition of errors. It might be the case that polling incidentally results in erroneous requests or that errors are generated by requests performed by a human operator.

5.5 Area for further research on SNMP usage

The research question outlined in section 1.2.5 is not related to SNMP analysis in practice, but asks to find an area on which SNMP usage can be researched.

Table retrieval in SNMP was found to be an area in which research on SNMP usage could be done. Researching this area gives information about the current usage patterns and could lead to either the conclusion that optimisations can be done, or the conclusion that applications are already retrieving tables efficiently. It could also lead to improvements to the SNMP protocol which allow for more efficient SNMP usage than both table fetch methods discussed in section 4.1.

5.6 Possible improvements to the SNMP protocol

The research question identified in section 1.2.6 asks if, based on the analysis of this thesis, improvements can be suggested which would improve usability of the protocol.

In the graphs in section 3.2.1 and the trace summaries from section 3.3 it can be seen that the general pattern in the examined traces is scheduled information retrieval (polling) using SNMPv1 and/or SNMPv2. From this observation two conclusions can be drawn that could relate to obstacles which tool developers experience when building polling-based software for SNMP. First of all it's strange that, while SNMP appears to be mainly used to periodically retrieve similar sets of information, there is no protocol support for scheduled information retrieval. It could be useful for tool developers to perform a single subscribe action for a certain set of information and have the agent send this information to the manager at the interval specified by the manager. This could be implemented by a MIB or by the protocol itself.

Secondly implementing SNMPv3 seems to be a problem, as mentioned in section 5.1, or there might be no need for it. SNMPv3 actually deserves a new meaning for the S in the abbreviation, as it's not as simple anymore as SNMPv1 (secure would be a better meaning for the letter S). Perhaps implementing these features is not worth the effort. This can be accepted as a fact or research can be done to see if changes to the protocol would make implementing SNMPv3 easier.

If there would ever be a new version of SNMP, it should take into account the usage patterns of SNMP based management in practice. On one hand it will prevent a repetition of history where an SNMP version in the end isn't used, as occurred for version three.
On the other hand working on obstacles from real-world usage will bring the tool developers and protocol designers together and will hopefully create a new version which allows tool developers to create more efficient applications.

6. References

[1] J. Schoenwaelder, "SNMP Traffic Measurements" IETF Internet Draft, <draft-irtf-nmrg-snmp-measure-01.txt>, work in progress, January 2007

[2] M. Rose, K. McCloghrie and J. Davin, "Bulk Table Retrieval with the SNMP", RFC 1187, Performance Systems International, Inc., Hughes LAN Systems, MIT Laboratory for Computer Science, October 1990

[3] R. Sprenkels, J.P. Marin-Flatin, "Bulk Transfers of MIB Data", The Simple Times, Vol. 7, No. 1, March 1999, http://www.simple-times.org/pub/simple-times/pdf/vol7-num1.pdf, as seen on February 6th 2007

[4] Internet Engineering Steering Group, "WG Action: Evolution of SNMP Working Group (eos) to conclude", April 2003, http://psg.com/lists/eos/eos.2003/msg00063.html, as seen on February 6th 2007

[5] Internet Research Task Force, "Network Management Research Group Charter (NMRG)", http://www.ibr.cs.tu-bs.de/projects/nmrg/, as seen on February 6th 2007

[6] O. Festor, "Minutes of the 20th NMRG meeting", IRTF, July 2006, http://www.ibr.cs.tu-bs.de/projects/nmrg/minutes/minutes-020.txt, as seen on February 6th 2007

[7] M. Harvan, "Prefix- and Lexicographical-order-preserving IP Address Anonymization", EECS Seminar presentation, March 2006, http://www.eecs.iu-bremen.de/wiki/images/9/92/Ip-anon.pdf, as seen on February 6th 2007

[8] https://subversion.eecs.iu-bremen.de/svn/schoenw/src/snmpdump/trunk/, as seen on February 6th 2007

[9] J. Schönwälder, A. Pras, M. Harvan, J. Schippers, R. van de Meent, "SNMP Traffic Analysis: Approaches, Tools, and First Results", Proceedings of the10th IFIP/IEEE Symposium on Integrated Network Management (IM 2007), May 2007, to be released

[10] tcpdump/libpcap website, http://www.tcpdump.org/, as seen on February 6th 2007

[11] This thesis' support website, http://wwwhome.cs.utwente.nl/~schippersjk/snmp/, as seen on February 6th 2007

7. Appendix

This appendix contains the Traffic Type Graph script written in PHP. It needs the following command line tools to work:

- tcpdump
- gnuplot
- zcat
- head

It needs two parameters: the source file and the output file. If the output file ends in .eps, an EPS file will be generated, otherwise a PNG image will be written. The gnuplot script and data files will be available after the script has finished. One data file will be generated for each management port. These files can be used to generate more specific graphs.

```
#!/usr/bin/php
<?php
error_reporting(E_ALL);

## PARSING INPUT ##
if($_SERVER['argc'] <= 2) {
        die('usage: '.$_SERVER['argv'][0].' <pcap/csvfile> <output>'."\n");
}

$input = $_SERVER['argv'][1];

$filetype = substr(str_replace('.gz', '', $input), -4) == '.csv' ? 'csv' :
'pcap';

if(!file_exists($input)) {
        die($input.' does not exist'."\n");
}

echo 'Input: '.$input."\n";
echo 'File type: '.$filetype."\n";

## INITIALIZING VALUES ##
$ports = array(22, 23, 80, 161, 162, 443, 514);
$timeframelength = 30;
$basetime = 0;
$timeframe = 0;
$currenttimeframe = 0;
$haspackets = array();
$tsdiff = mktime(0, 0, 0, 1, 1, 2006, 1);

## SETTING UP DATAFILES ##
$files = array();
$filenames = array();
$currentpackets = array();
for($i = 0; $i < count($ports)+1; $i++) {
        $filenames[$i] = ($i == 0 ? 'traffictypegraph_other' :
'traffictypegraph_port'.$ports[$i-1]).'.dat';
```

```
        $files[$i] = fopen($filenames[$i], 'w+');

        echo 'datafile for '.($i == 0 ? 'other traffic' : 'port
'.$ports[$i-1]).' is '.$filenames[$i]."\n";
        $currentpackets[$i] = 0;
}
$output = $_SERVER['argv'][2];

## OPENING SOURCEFILE ##
$handle = open_file($input, $filetype);
if(!$handle) {
        die('unable to open '.$input."\n");
}
echo 'begin: '.strftime('%d/%M/%Y %H:%M')."\n";

## LOOPING THROUGH FILE ###
while(!feof($handle)) {
        if((list($time, $srcport, $dstport) = parse_line(fgets($handle),
$filetype)) == false) continue;

        if($basetime == 0) $basetime = $time;
        $timeframe = round(($time - $basetime) / $timeframelength);

        $port = $srcport < 1024 ? $srcport : $dstport; // $dstport check

        $portindex = array_search($port, $ports);
        if($portindex === false) {
                $portindex = 0;
        } else $portindex++;

        if($timeframe > $currenttimeframe) {
                $n = 0;
                for($i = 0; $i < count($ports)+1; $i++) {
                        $n += $currentpackets[$i];
                        $ts =
($basetime+$currenttimeframe*$timeframelength);
                        $ts -= $tsdiff;
                        fwrite($files[$i], $ts.'000
'.$currentpackets[$i]."\n");
                        if($currentpackets[$i] > 0) {
                                $currentpackets[$i] = 0;
                                $haspackets[$i] = true;
                        }
                }

                // fill in zeroes in gaps
                for($i = $currenttimeframe + 1; $i < $timeframe; $i++) {
                        for($j = 0; $j < count($ports)+1; $j++) {
                                $ts = ($basetime+$i*$timeframelength);
                                $ts -= $tsdiff;
                                fwrite($files[$j], $ts.'000 0 test'."\n");
                        }
                }

                $currenttimeframe = $timeframe;

                echo $currenttimeframe." - ".$n."\r";
        }
        $currentpackets[$portindex]++;
}
pclose($handle);
```

```
echo "\n".'end: '.strftime('%d/%M/%Y %H:%M')."\n";

for($i = 0; $i < count($ports)+1; $i++) {
        $ts = ($basetime+$currenttimeframe*$timeframelength);
        $ts -= $tsdiff;
        fwrite($files[$i], $ts.'000 '.$currentpackets[$i]."\n");
}

# WRITING THE GNUPLOT FILE
$gnuplotfile = 'trafficTypeGraph.gnuplot';
echo 'the gnuplot file is '.$gnuplotfile."\n";
$gnuplot = fopen($gnuplotfile, 'w+');
if(preg_match('/\.eps$/i', $output)) {
        fwrite($gnuplot, '#set terminal png transparent size
840,320'."\n");
        fwrite($gnuplot, 'set terminal postscript eps monochrome'."\n");
        fwrite($gnuplot, 'set xtics rotate by -45'."\n");
} else {
        fwrite($gnuplot, 'set terminal png transparent size 840,320'."\n");
        fwrite($gnuplot, '#set terminal postscript eps monochrome'."\n");
        fwrite($gnuplot, '#set xtics rotate by -45'."\n");
}
fwrite($gnuplot, 'set key below box'."\n");
fwrite($gnuplot, 'set ylabel "packets"'."\n");
fwrite($gnuplot, 'set xlabel "time"'."\n");
fwrite($gnuplot, 'set xdata time'."\n");
fwrite($gnuplot, 'set logscale y'."\n");
fwrite($gnuplot, 'set timefmt "%d/%m/%Y %H:%M"'."\n");
fwrite($gnuplot, 'set format x "%H:%M"'."\n");
fwrite($gnuplot, 'set output "'.$output.'"'."\n");
//fwrite($gnuplot, 'set xrange ["+start.substring(0, start.length()-
3)+".000:"+end.substring(0, end.length()-3)+".000] writeback'."\n");
//fwrite($gnuplot, 'set yrange
["+1+":"+ttg.getMaxPacketsInTimeframe()+"]'."\n");
fwrite($gnuplot, 'plot ');
$first = true;
for($i = 0; $i < count($files); $i++) {
        if(isset($haspackets[$i])) {
                fwrite($gnuplot, '\\'."\n".'  '.(!$first ? ', ' :
'').'"'.$filenames[$i].'" using ($1/1000):2 title "'.($i == 0 ? 'other' :
'port '.$ports[$i-1]).'" with lines');
                $first = false;
        } else {
                unlink($filenames[$i]);
        }
}
fwrite($gnuplot, "\n");
fclose($gnuplot);

echo shell_exec('gnuplot '.$gnuplotfile);

echo 'Created '.$output."\n";


// using popen because files might be too large for fopen
function open_file($filename, $filetype) {
        switch($filetype) {
                case 'csv':
                        if(is_gz($filename))
                                //return gzopen($filename, 'r');
```

```
                                return popen('zcat
'.escapeshellarg($filename), 'r');
                        else
                                //return fopen($filename, 'r');
                                return popen('cat
'.escapeshellarg($filename), 'r');
                case 'pcap':
                        if(is_gz($filename)) {
                                return popen('zcat
'.escapeshellarg($filename).' | tcpdump -tt -q -nr - udp or tcp', 'r');
                        } else {
                                return popen('tcpdump -tt -q -nr
'.escapeshellarg($filename).' udp or tcp', 'r');
                        }
                default:
                        return false;
        }
}

// returns array(time, srcport, dstport) or false
function parse_line($line, $filetype) {
        switch($filetype) {
                case 'csv':
                        $els = explode(',', $line, 5);
                        if(count($els) != 5) return false;
                        return array(
                                intval(substr($els[0], 0, strrpos($els[0],
'.'))),
                                161,
                                1024
                        ); // it's snmp anyway, and anonimized traces
contain weird portnumbers
                        break;
                case 'pcap':
                        $els = explode(' ', $line, 7);
                        if(count($els) != 7 || ($els[5] != 'UDP,' && $els[5]
!= 'tcp')) return false;
                        return array(
                                intval(substr($els[0], 0, strrpos($els[0],
'.'))),
                                intval(substr($els[2], strrpos($els[2],
'.')+1)),
                                intval(substr($els[4], strrpos($els[4],
'.')+1, -1))
                        );
        }
}

// is_gz
function is_gz($filename) {
        $fp = popen('head -c2 '.escapeshellarg($filename), 'r'); // can't
use fopen for laaaarge files
        $magicnumber = fread($fp, 2);
        fclose($fp);
        return (ord($magicnumber{0}) == 31 && ord($magicnumber{1}) == 139);
}
?>
```