SNMP Traffic Analysis: Approaches, Tools, and First Results

Jürgen Schönwälder*, Aiko Pras†, Matúš Harvan*, Jorrit Schippers†, Remco van de Meent†
* Computer Science
International University Bremen
Campus Ring 1
28759 Bremen, Germany
† Computer Science
University of Twente
P.O. BOX 217
7500 AE Enschede, The Netherlands

***Abstract*— The Simple Network Management Protocol (SNMP) is widely deployed to monitor, control, and configure network elements. Even though the SNMP technology is well documented and understood, it remains relatively unclear how SNMP is used in practice and what the typical SNMP usage patterns are. This paper discusses how to perform large-scale SNMP traffic measurements in order to develop a better understanding of how SNMP is used in production networks. The tools described in this paper have been applied to networks ranging from large national research networks to relatively small faculty networks. The goal of the research is to provide feedback to SNMP protocol developers within the IETF, researchers working within the context of the IRTF-NMRG, as well as other researchers interested in network management in general. We believe that the results are also valuable for operators and vendors who want to optimize their management interactions or understand the traffic generated by their management software.**

I. Introduction

The Simple Network Management Protocol (SNMP) was introduced in the late 1980s [1] and has since then evolved to what is known today as the SNMP version 3 framework (SNMPv3) [2]. While SNMP is widely deployed, it is not clear which features are being used, how SNMP usage differs in different types of networks or organizations, which information is frequently queried, and what typical SNMP interaction patterns are in real world production networks.

There have been several publications in the recent past dealing with the performance of SNMP in general [3], the impact of SNMPv3 security [4], [5], or the relative performance of SNMP compared to Web Services [6]–[8]. While these papers are generally useful to better understand the impact of various design decisions and technologies, some of these papers lack a strong foundation because authors typically assume certain SNMP interaction patterns without having experimental evidence that the assumptions are correct. In fact, there are many speculations on how SNMP is being used in real world production networks and how it performs, but no systematic measurements have been performed and published so far.

Many authors use the `ifTable` of the `IF-MIB` [9] or the `tcpConnTable` of the `TCP-MIB` [10] as a starting point for their analysis and comparison. Despite the fact that there is no evidence that operations on these tables dominate SNMP traffic, it is unclear how these tables are read and which optimizations are done (or not done) by real world applications. It is also unclear what the actual traffic trade-off between periodic polling and more aperiodic data retrieval is. Furthermore, we do not generally understand how much traffic is devoted to standardized MIB objects and how much traffic deals with proprietary MIB objects and whether the operation mix differs between these object classes or between different operational environments.

This paper describes an effort to collect SNMP traffic traces in order to find answers to some of these questions. Section II discusses possible approaches to collect traces and Section III describes the tools that have been developed to analyze such traces. Section IV discusses the locations from which traces have already been collected. Section V provides some initial results of our analysis; it should be noted that our research is still in progress and more detailed results will be published in a forthcoming paper. Section VI discusses related work and conclusions are finally provided in Section VII.

II. Approaches

The collection of SNMP traffic traces requires the support of network operators. On the technical side, good capturing points have to be defined and configured. On the non-technical side, an agreement has to be defined under which data can be shared and results published.

It is in usually not possible to make SNMP traces which contain complete SNMP messages openly available since they contain sensitive information. In some cases, we have settled on agreements, which make data available to specific researchers for research purposes while in other cases the operators involved keep the data and instead run our analysis software and provide the aggregated results back to us.

A. Sharing Traces

In the first approach, an operator collects traces and subsequently makes them available to researchers for further

processing. Since traces contain data of different levels of sensitivity, operators typically want to exercise some control over the data that is given to researchers. Good examples are SNMP community strings, which are often used as clear-text passwords and hence should be removed. Since removing such data in binary pcap files which contain BER encoded SNMP messages is technically non-trivial and also difficult to verify, there is a need for a human and machine readable representation which makes it easier to (a) identify data to be removed, (b) to actually remove the identified sensitive data, and (c) to verify that the removal was successful.

Next to the filtering of highly sensitive data, some operators also prefer to have data anonymized so that the risk of leaking sensitive information is reduced. Even though strong anonymization is difficult to achieve, it is still often considered useful to achieve at least a level of pseudonymization as a second safety measure to complement a non-disclosure agreement. Anonymization requires applying a filter-in principle where only data for which an appropriate anonymization function exists is retained in a trace. Thus, anonymization can reduce the usefulness of the traces for researchers and it therefore requires some careful planning on the side of the operator involved. Furthermore, the development of suitable anonymization functions is still an ongoing research topic and hence the required software tools are experimental and changing rapidly.

B. Sharing Analysis Software

The alternative approach to sharing traces is to share the analysis software and to ask operators who own the traces to execute the analysis software on behalf of researchers. Of course, operators who execute analysis software provided by researchers have to trust the software and check the results against their privacy requirements. As a consequence, analysis software should be open, portable, and reasonably well documented. Analysis software should be provided in a way, which makes it possible for operators to verify that the code does not contain any unwanted features. This also implies a pragmatic selection of programming languages so that programs are likely to be understood by the operator community.

Experience so far tells us that a combination of both approaches usually works reasonably well. In such cases a researcher obtains potentially filtered and anonymized traces from an operator (typically legally covered by an agreement between the operator and the trace analyst) and the researcher executes analysis software provided by other researchers interested in the trace (assuming that is covered by the agreement).

C. Intermediate Formats

To support the above mentioned approaches, two intermediate formats for SNMP traces have been developed [11]. The first one is an XML format which is intended as a human and machine readable exchange format which is capable to retain all information found in BER encoded SNMP messages. This format is relatively verbose (traces in XML format are typically a factor 7 larger than the original pcap trace file), but this can be mitigated by compression. A large number of tools do exist to process XML files and so ad-hoc transformations are feasible. However, our experience is that many XML tools do not scale very well to large data sets.

The second intermediate format is a simple CSV (comma separated values) format, which only retains the most essential information. It turns out that processing CSV files is usually much faster, especially since many line-oriented tools can be applied directly. The downside of the CSV format is that it is not very flexible and changes in the CSV format typically break tools in surprising ways. Hence it is crucial to get this format right and to keep it stable.

III. Tools

SNMP traces are typically captured using tcpdump and stored in pcap format. Hence, generic tools, which can process raw pcap files, can be used to split and merge the raw traffic traces. In order to analyze the traces, we have developed additional tools, which use the intermediate formats described above. Since traces may become quite large, tools should be relatively efficient and fast.

A. Conversion Tool (*snmpdump*)

A new tool called snmpdump accomplishes the conversion of raw pcap traces into intermediate formats. It reads raw pcap files as input and produces traces in XML or CSV format as output. Since the XML format retains all information, it is also supported as an input format. This allows using snmpdump as a filter. In addition, snmpdump accepts CSV format as input even though this format does not retain all information.

1) Parsing: To extract SNMP messages from raw pcap files, it is necessary to (a) deal with fragmentation by reassembling fragmented datagrams and (b) decode the native BER format into an internal representation. For packet reassembly, we used the libnids library, which essentially contains a user-space version of some portions of the Linux TCP/IP networking code. For the BER decoding, we modified the SNMP packet decoder shipped with tcpdump to construct an in memory representation of an SNMP message.

To parse data stored in the intermediate XML format, we use the libxml reader API. The data provided by the generic XML parser, which automatically checks well-formedness, is used to generate an in memory representation. Although not surprising, it should be noted that parsing raw pcap files is significantly faster and hence pcap remains a good choice for processing large trace files if no filtering or anonymization is needed.

The CSV parser is a straightforward implementation which reads a line, tokenizes it and then generates an in memory representation of the message. The CSV parser is handy when existing CSV files have to be processed further.

2) Data Representation and Control Flow: SNMP messages are represented internally using a nested data structure, which can represent the different SNMP message formats.

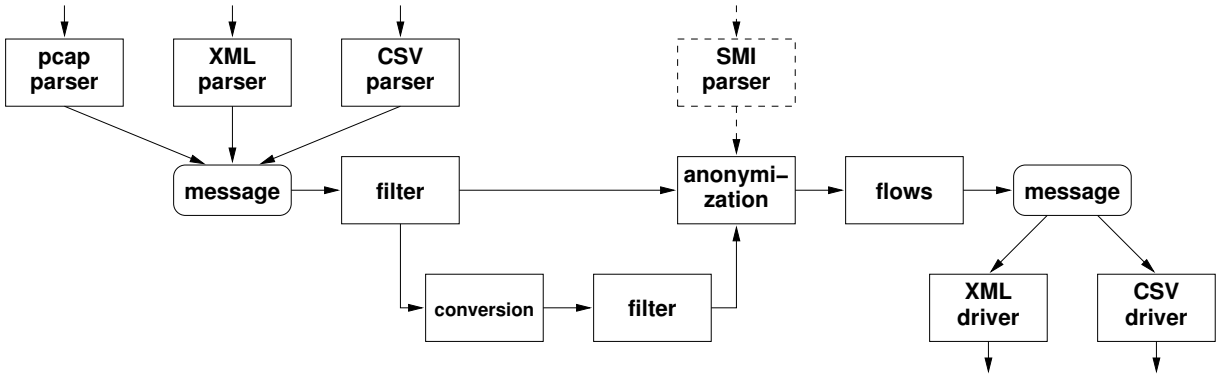

Fig. 1. Data flow within the `snmpdump` data conversion tool.

Every data member carrying the value of a field of an SNMP message has associated attributes. These attributes control memory management and indicate whether a value is present. By adding these attributes to all data members, we are able to pass a decoded SNMP message through several stages of a processing pipeline until the message is finally serialized into one of the output formats. The drivers, which produce different output formats, adhere to a common interface, which makes the implementation extensible. The overall data flow within `snmpdump` is shown in Figure 1.

3) Filtering: The filter module of `snmpdump` is responsible to filter out message fields that should be suppressed, for instance because specific sensitive data must be removed. The message fields that should be suppressed are selected using a regular expression and the suppression essentially changes the attributes of the selected message fields. As a safety measure, the data stored in filtered message fields is cleared or set to some standard "null" value, just in case some other code forgets to check the attributes when accessing message fields.

4) Conversion: The format of the payload of SNMP messages changed when the second version of SNMP was introduced. In particular, the format of unconfirmed traps was changed and harmonized. The coexistence specification [12] defines a conversion procedure, which allows traps in the old format to be translated into the new format and back. The conversion module implements this conversion procedure in order to provide a uniform interface. Note that the conversion module can be bypassed if no conversion is desirable. If conversion has been performed, it is necessary to call the filter module again since the conversion might have filled message fields with values, which were not present before.

5) Anonymization: The anonymization module is responsible for anonymizing message fields. It makes use of a reusable anonymization library called `libanon` [13]. The library provides anonymization functions for standard data types such as signed / unsigned integers and octet strings as well as specific functions for MAC addresses or IP addresses. The anonymization functions support a lexicographic-order-preserving mode in order to preserve SNMP's lexicographic-order property of instance identifiers. A more detailed description of prefix- and lexicographic-order-preserving IP address anonymization can be found in [13].

In order to select the anonymization function for a given message field, it is necessary to have some context information, such as the object descriptor or the object's type name. The anonymization module therefore looks up data definitions by calling the `libsmi` library, an embeddable MIB parser library. Note that these lookups are only performed if anonymization has been requested and the values in question are actually present. The selection of the anonymization function to apply for a given object or a given data type is runtime configurable.

6) Flow Identification: Large traces, which may contain interactions of several managers with hundreds of agents, usually have to be broken into more manageable pieces. A natural choice is to split a combined trace into several traces, each one representing a message flow. An SNMP message flow is defined as all messages between a source and destination address pair which belong to a command generator (CG) / command responder (CR) relationship or a notification originator (NO) / notification receiver (NR) relationship.

The above definition deliberately does not consider port numbers. The reasons are twofold: First, most managed devices include just a single SNMP agent. Even if multiple agents are present, either subagent protocols or proxy mechanisms are usually used to hide this. Even if a device has multiple totally independent SNMP agents, we still consider the device a single logically managed device. Second, many managing systems make heavy use of dynamically allocated port numbers which can change frequently and thus would cause lots of unrelated flows to be generated, even though all the flows are coming from a single management station.

The implementation of the flow identification module requires to deal with reordered messages and to associate responses (and reports) to prior requests since responses do not indicate whether they are send in response to a notification or a data retrieval operation.

B. Analysis Tools

Analysis tools usually read the intermediate format produced by `snmpdump` in the first stage and extract meaningful statistics in a second processing stage.

A Java 1.5 based statistics generator takes an intermediate file as input, parses it sequentially and handles internally each extracted packet to a set of statistics generators, which are responsible for maintaining counters and other data structures. Currently the program is capable of generating basic protocol statistics, like version and protocol operation usage, as well as statistics about MIB usage, relations between managers and agent and statistics about error responses.

In addition to the Java program, a collection of Perl scripts have been developed (part of the `snmpdump` distribution) which analyze intermediate files in `CSV` format. Besides the generation of basic statistics, the scripts are able to analyze SNMP walks (sequences of `GetNext` or `GetBulk` operations) in order to produce data how applications retrieve management information and which strategies are used to implement tables. Additional tools can perform object name (OID) conversion and aggregation on base statistics by reading the MIB identifiers lists that can be created using the `smidump` MIB compiler.

IV. Traces

Traces have been collected at several different locations. We report here analysis results covering eight traces collected at seven different locations. To easily identify the traces and the locations, we use a naming scheme essentially consisting of two numbers: The first number identifies a location while the second number identifies a specific trace collected at that location. For example, a trace name such as *l01t02* refers to the second trace collected at location number 1.

trace	description	start	hours
l01t02	national research network	2005-07-26	162.98
l01t05	national research network	2006-07-10	336.00
l02t01	university network	2006-04-21	294.62
l03t02	faculty network	2006-04-27	159.21
l04t01	server-hosting provider	2006-04-14	4.00
l05t01	regional network provider	2006-04-19	580.60
l06t01	national research network	2006-05-14	222.08
l12t01	point of presence	2006-07-10	208.02

TABLE I
Overview of the SNMP traces

Table I provides an overview of the traces analyzed in this paper. The traces *l01t02* and *l01t05* have been collected at the backbone of SURFnet, which is the research network provider within the Netherlands. Note, however, that these two traces are collected with roughly a year in between and that different attachment points were used to collect the traffic. The first trace was collected near the network operation center (NOC) while the second trace was collected close to a data collection point in the network.

Trace *l02t01* was collected on a University network management VLAN while trace *l03t02* was collected on a faculty network. The relatively short trace *l04t01* was collected at a server-hosting provider. The data was collected by capturing all SNMP traffic originating from or destined to a specific network manager.

Trace *l05t01* was collected at a regional network provider network. The network utilizes many wireless point-to-point links to interconnect research institutions, government institutions and commercial organizations.

Trace *l06t01* was collected on the main network management server of a national research network. Note that there are additional systems generating SNMP traffic in this network and thus the trace only describes the traffic generated by a single management system.

Finally, trace *l12t01* was collected at a point of presence of another national research network.

Network traffic was captured using `tcpdump` and stored in `pcap` format. In some cases, we could capture other management traffic (e.g. SYSLOG) in addition to SNMP traffic. We plan to analyze these traces in the future and related the results to the SNMP analysis we are working on at the moment.

V. Analysis

The purpose of this section is to present some initial analysis results; more traces must be collected and additional analysis methods and scripts must be developed before more comprehensive conclusions on SNMP usage can be drawn. Still it is possible to present some interesting first results.

A. General Characterization

Table II provides a general characterization of the traces. The trace sizes are given in the CSV format that has been used throughout in the analysis. Most traces included all information in the CSV file. Only traces *l04t01* and *l12t01* have been filtered to exclude varbind values and thus the trace file contain less information and are somewhat smaller.

trace	size [MB]	messages	SNMPv1	SNMPv2	SNMPv3
l01t02	6369	51772136	100.0%	-	-
l01t05	14043	40072529	-	100.0%	0.0%
l02t01	77789	258010521	5.5%	94.5%	-
l03t02	130858	871361365	95.0%	5.0%	-
l04t01	10	15099	35.7%	64.3%	-
l05t01	2898	25298667	100.0%	-	-
l06t01	24683	89277889	57.4%	42.6%	-
l12t01	312	2619884	32.3%	67.7%	-

TABLE II
General characterization of the traces

Table II shows the SNMP versions found in the traces. Despite the fact that the status of SNMPv1 and SNMPv2 is *historic* and only SNMPv3 is full *standard*, it turns out that SNMPv3 is not really used in any of our traces. Trace *l01t05* contains a very few SNMPv3 messages but they do not contribute significantly to the trace; it seems that someone was experimenting with SNMPv3 while the traces were collected.

This result did not come as a surprise since earlier discussions with operators already showed that, besides for (DOCSIS) cable modem management, SNMPv3 is still not widely deployed. A second interesting observation is that some trace locations still seem to rely solely on SNMPv1, whereas

the management traffic at other locations is dominated by SNMPv2.

B. Protocol Operations

Table III shows for every trace the usage of the different protocol operations. In general, the overall traffic is dominated by `Get`, `GetNext`, and `GetBulk` operations and their responses. Some traces do not contain any notifications and only trace *l06t01* contains acknowledged notification.

trace	Get	Next	Bulk	Set	Trap	Info	Resp
l01t02	0.0	50.0	-	0.0	-	-	50.0
l01t05	0.0	-	50.0	-	-	-	50.0
l02t01	0.1	2.4	47.1	0.0	0.7	-	49.6
l03t02	0.3	49.8	-	0.0	0.0	-	49.9
l04t01	32.8	3.8	22.9	-	-	-	40.5
l05t01	50.0	0.0	-	-	0.0	-	50.0
l06t01	12.1	31.4	6.5	-	0.0	0.0	50.0
l12t01	1.0	49.0	-	-	0.0	-	49.9

TABLE III
USAGE OF PROTOCOL OPERATIONS (IN %)

The traces *l01t02*, *l02t01*, and *l03t02* contain a `Set` operations but due to the small number, they do not play a significant role in the overall traffic mix. A closer look at trace *l01t02* revealed that all recorded `Set` operations were trying to modify the `sysLocation` scalar with a value of type `Integer32`, which obviously leads to an error response due to a type mismatch, if authentication and access control would have been successful. In trace *l02t01*, we observe `Set` requests to two proprietary MIB modules, which allow to copy configuration files to/from a device. In trace *l03t02*, we found that `Set` operations are used to trigger the download of a VLAN membership policy specification.

Table III also reveals that trace *l04t01* contains significantly more requests than responses. One possible explanation could be packet loss. Upon further investigation and discussion with the network operators, we learned, however, that the day the traces were collected some systems were switched off for maintenance purposes.

trace	Get	Next	Bulk	max-reps	non-reps
l01t02	37.5%	99.3%	-	-	-
l01t05	100.0%	-	100.0%	10/50	0
l02t01	56.3%	99.9%	100.0%	1/10/20/25	0
l03t02	1.6%	99.9%	-	-	-
l04t01	100.0%	100.0%	100.0%	1000	0
l05t01	99.9%	95.6%	-	-	-
l06t01	8.7%	2.6%	0.0%	12	0
l12t01	100.0%	99.9%	-	-	-

TABLE IV
PERCENTAGE OF SINGLE VARBIND GET, GETNEXT, AND GETBULK OPERATIONS AND GETBULK PARAMETERS

Since the traffic is dominated by data retrieval operations, it makes sense to take a closer look how data retrieval is performed. Columns 2, 3, and 4 of Table IV show the percentage of `Get`, `GetNext`, and `GetBulk` requests that contain only one varbind in the varbind list. Except for trace *l06t01*, single varbind `GetNext` and `GetBulk` operations clearly dominate. In four traces, even the `Get` operations tend to be single varbind operations. In trace *l06t01* we find that 86.5% of the `GetBulk` operations contain two varbinds and the remaining 13.5% contain eight varbinds. Furthermore, 85.8% of the `GetNext` operations contain two varbinds. A conclusion from this analysis is that except in trace *l06t01*, table retrieval is usually realized in column-by-column mode.

Columns 5 and 6 of Table IV indicate the max-repetitions (`max-reps`) and the non-repeaters (`non-reps`) parameters of the `GetBulk` operations. The first interesting observation is that none of the `GetBulk` operation used non-repeaters. In traces *l01t05* and *l02t01*, almost 100% of the `GetBulk` requests use 10 `max-reps`. The `GetBulk` requests in trace *l04t01* always use 1000 `max-reps` while the requests in trace *l06t01* always use the 12 `max-reps`.

C. Response Size Distribution

Figure 2 shows the cumulative distribution of the sizes of the response messages. The vast majority of the response messages are about 100 bytes long for traces that do not include the `GetBulk` operation. The corresponding requests are usually even smaller since they do not contain any values. As we will see later, most of the objects retrieved are actually simple numbers whose encoding is relatively compact.

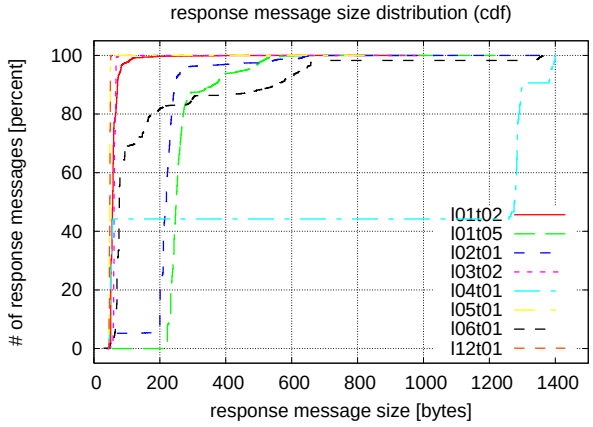

Fig. 2. Response message size distribution (cdf)

The traces that include `GetBulk` requests contain larger response messages. Although almost all `GetBulk` response sizes could be observed, no response message was larger than roughly 1400 bytes. This implies that no fragmentation occurred on a path with an Ethernet PMTU.

When comparing the four `GetBulk` traces, one can observe that trace *l04t01* achieves a much higher percentage of large response messages. This is due to the fact that this particular management applications gives the agent more freedom to generate large response messages by using a large `max-repetitions` parameter (1000). We have observed responses with up to 74 repetitions for *l04t01*. As explained

above, traces *l01t05* and *l02t01* prefer a max-repetitions parameter of 10 and we also observe a matching number of responses with 10 varbinds. According to Figure 2, this leads to most responses being about 300 bytes long. Trace *l06t01* uses the max-repetitions parameter 12 and retrieves multiple varbinds, which leads to some slightly better performance. We conclude that the setting of the max-repetitions parameter can be improved in three out of four locations and we also observe that obviously do not try to dynamically adjust the max-repetitions parameter.

D. Flow Analysis

To further analyze the traces, it is useful to look at individual flows. Table V lists in columns 2 and 3 the number of CG/CR flows and the number of NO/NR flows we have identified in the traces. The remaining columns list the number of CG/CR/NO/NR interfaces that were identified. Note that we identify interfaces and not managers or agents. Hence, it is possible that, for example, multiple CGs are running on a single multi-homed host. From discussions with operators, we know that management systems are indeed often multi-homed (and many routers are by there very nature). In particular, we know that trace *l06t01* was obtained from a single management system.

trace	cg/cr flows	no/nr flows	cg	cr	no	nr
l01t02	203	-	3	178	-	-
l01t05	8	-	2	8	-	-
l02t01	258	197	5	240	197	1
l03t02	42	20	25	20	17	2
l04t01	34	-	3	34	-	-
l05t01	117	2	9	99	2	2
l06t01	288	125	3	260	125	2
l12t01	30	6	5	26	6	1

TABLE V
CHARACTERISTICS OF CG/CR AND NO/NR FLOWS

The traffic is not evenly distributed across the flows. In fact, all traces have a relatively small number of dominating flows. Figure 3 shows the number of messages per minute and the number of bytes per minute exchanged in the various flows of trace *l01t02*. The flows were sorted by descending number of messages per minute.

The solid curve in Figure 3 shows that there are some high volume flows with up to 300 messages per minutes (mpm) but also many flows with less than 20 messages per minutes. The dashed curve shows that the traffic peeks at almost 18000 bytes per minute (bpm) but goes down to less than 1000 bytes per minute for many flows. The figure also indicates that the number of messages and the number of bytes transferred are strongly correlated in this trace. This is not too surprising since we know that most requests are single varbind `GetNext` requests in this trace (and we will see later that most of them retrieve numbers).

The strong correlation between the number of requests and the bandwidth consumed surely cannot be expected for flows that make heavy usage of the `GetBulk` operation. From

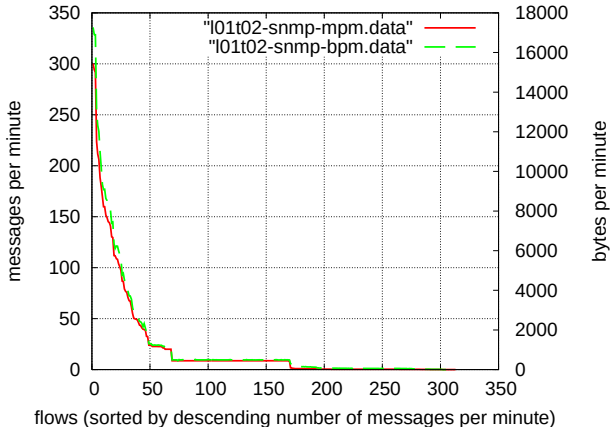

Fig. 3. Traffic intensity of the flows in trace *l01t02*

Figure 2, we already know that trace *l04t01* makes efficient use of `GetBulk` operations by setting the max-repetitions parameter to 1000. The traffic intensity of the flows in this trace shown in Figure 4 prove this.

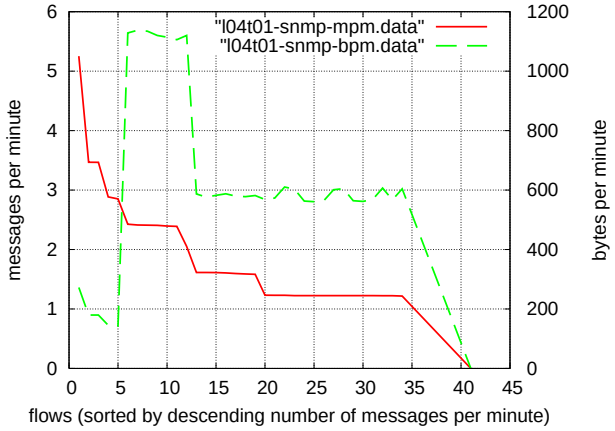

Fig. 4. Traffic intensity of the flows in trace *l04t01*

The computation of the traffic flows for trace *l03t02* revealed that there is one very high intensity flow, which contains 94.3% of all messages in this trace. Figure 5 shows the traffic intensity of the flows in logarithmic scale. This heavy flow generated close to 90,000 messages per minute. By inspecting this flow, we found that a management application got caught in a time-filtered table.

The time-filter mechanism, which was introduced in `RMON2-MIB` [14], can be used to retrieve only those rows of a time filtered table that have changed since a particular time. This is achieved by inserting a time-filter component in the index and excluding rows from the view that have not changed since the time-filter provided in the request. The original time-filter specification favored an implementation style where management applications that do not understand time-filtered tables could potentially run into a long laster or potentially endless loop (if the table changes faster than data can be retrieved). The update of the original time-filter

definition now suggests to implement time-filtered tables in a way that avoids these problems [14].

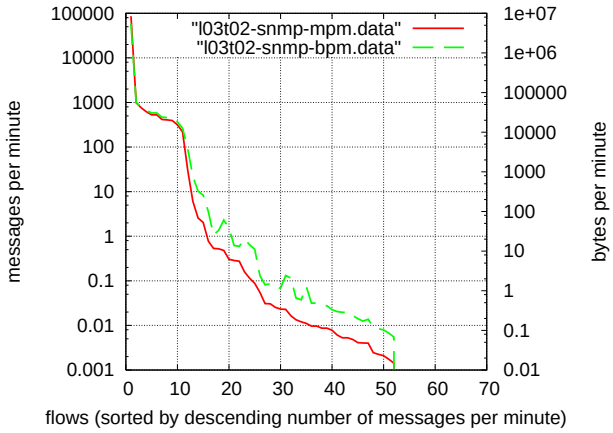

Fig. 5. Traffic intensity of the flows in trace *l03t02*

E. Flow Topologies

Based on the extracted flows, it is possible to draw graphs visualizing the flow topology between CGs, CRs, NOs, and NRs. Since there are typically much more CGs than CRs, we decided that we draw management systems (nodes that hosts CGs and/or NRs) as larger circles and managed elements (nodes that host CRs and/or NOs) as small dots. We indicate CG/CR flows with solid green arrows pointing the the CR and we use dashed red arrows for NO/NR flows. Furthermore, we indicate the relative amount of traffic at a node (as indicated by the number of messages per minute) by its gray level (black means high traffic intensity.

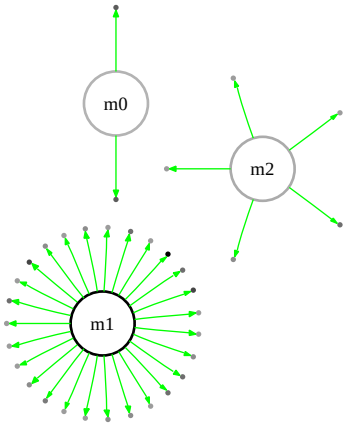

Fig. 6. Flow topology of trace *l04t01*

Figure 6, visualizing the trace *l04t01*, shows a typical simple monitoring flow topology. Management interface $m0$ is monitoring two agents, management interface $m2$ is monitoring 5 agents, and the remaining 27 agents are monitored by $m1$. A more complex monitoring topology is show in Figure 7 where we again see a rather simple low volume setup around management interface $m1$ plus a rather complex setup around the management interfaces $m0$ and $m2$. Surprisingly, a number of managed elements are connected to both $m0$ and $m2$.

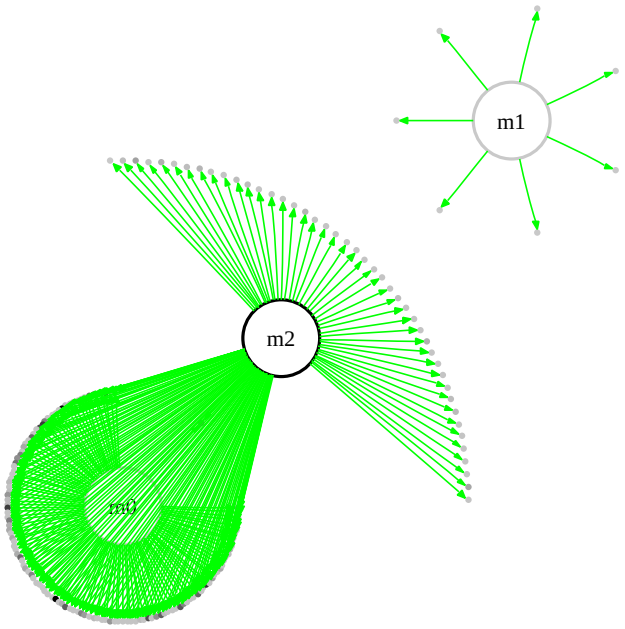

Fig. 7. Flow topology of trace *l01t02*

Another interesting plot which also includes a significant number of NO/NR flows is shown in Figure 8. Management interface $m3$ acts solely as a notification receiver while management interface $m1$ acts as both command generator and notification receiver. The management interface $m0$ acts as a pure commands generator. The single dark dot and the dark coloring of $m0$ indicates that the traffic is dominated by a single flow between $m0$ and the dark managed element. As mentioned above, we know that all management interfaces in trace *l01t02* belong to a single multi-homed machine.

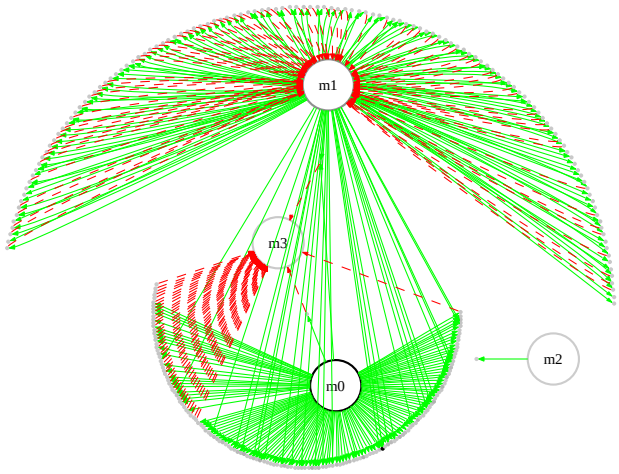

Fig. 8. Flow topology of trace *l06t01*

The flow topology shown in Figure 9 belongs to trace *l03t02* and looks kind of surprising. We see as many as 18

management interfaces talking to a single managed element interface. A closer inspection of the traces reveals that these 18 flows start consecutively, although some of them overlap. A further analysis of the flows reveals that the CRs retrieve information about printer supplies from the network element, which therefore likely is a somewhat important printer in the faculty network. We assume that the printer supply monitoring application is running on systems which get IP addresses assigned dynamically.

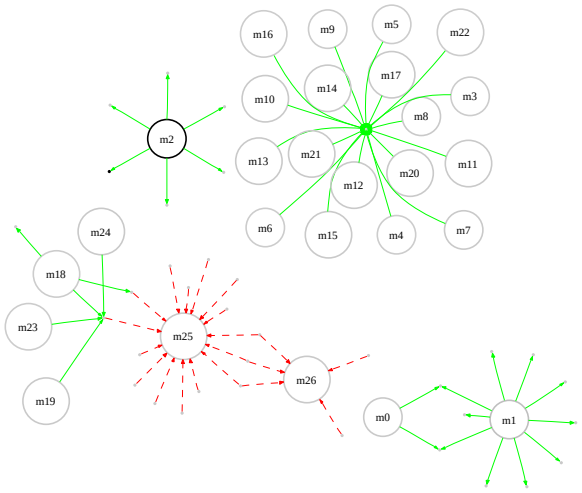

Fig. 9. Flow topology of trace *l03t02*

In Figure 9, we also see two notification receiver interfaces and there seem to be three nodes sending notifications to both notification receiver interfaces. The high volume time-filter traffic is exchanged between management interface $m2$ and the dark dot.

F. Traffic Pattern

It is generally assumed that tools like MRTG [15], which fetch MIB variables at regular intervals, like every 5 minutes, generate most SNMP traffic. If such traffic were plotted as a figure that shows the average number of SNMP packets as function of the time, a straight line would be the result, provided the time interval is taken sufficiently large, for example 24 hours.

While many flows show a quite regular traffic intensity, it is important note that this is not generally true. Figure 10 shows the time series for one of the highest volume flows in trace *l06t01*. While the regular periodic component can easily be identified, we also see significant variations. We know that the network had three major events (fibre cuts, power outages) during the measurement period such events clearly affect the observed monitoring traffic. It is also interesting to see that the average polling intensity seems to have changed after almost five days.

To our surprise, we found that many notification flows also carry periodic traffic, however typically at a much lower rate. Devices seem to regularly report device states (e.g., fan failures, printing engine problems) or unusual protocol

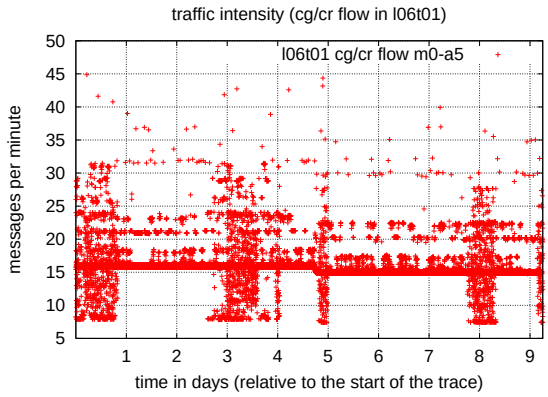

Fig. 10. CG/CR flow traffic intensity over time *l06t01*

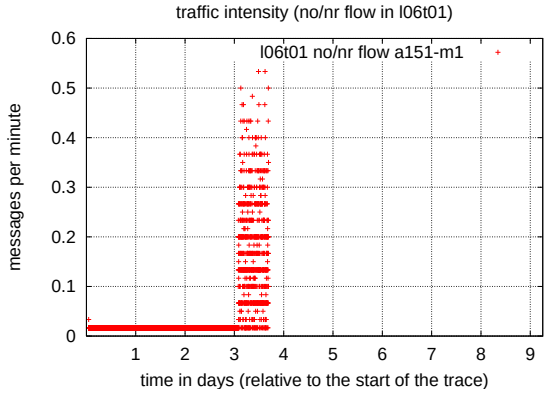

Fig. 11. NO/NR flow traffic intensity over time *l06t01*

states (e.g., PIM routing losses) that are not fixed. Figure 11 shows the traffic intensity of a notification flow which at the beginning is has a constant flow of one notification per minute. After three days, however, significantly more notifications are generated and then the flow terminates. Unfortunately, we do not have access to the notification details. All we know is that this device is an optical fibre system and that there were fibre cuts during the data collection period.

We further investigated how typical manager–agent interactions look like. Figure 12 shows 30 seconds of `GetNext` interactions for a specific CR at in trace *l02t01*. The figure shows that the manager retrieves approximately 170 MIB objects; these objects are lexicographically ordered and represented vertically as numbers 0 to 170. The first observation is that a significant performance gain, in bandwidth as well as response time, would be possible if this `GetNext` sequence would be changed into a `GetBulk` sequence. The second observation is that the same objects are retrieved repeatedly. Further investigation learned that many of these objects are relatively static (such as `ifDescription`); further performance improvements are therefore possible if more intelligent management tools would be used that cache such variables, instead of retrieving them over and over again.

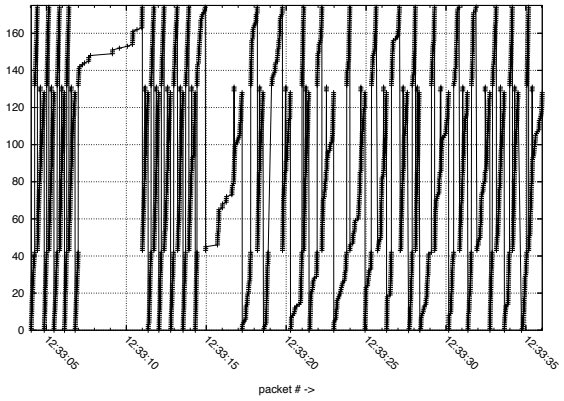

Fig. 12. Typical GetNext sequence at *l02t01*

G. Data Types

Table VI shows the base data types seen in response messages. The column exc counts the number of exceptions that have been seen. Response messages that include a null type are SNMP error messages (actually all noSuchName errors). However, there are also some traces where error responses contain data, especially in trace *l05t01*. A closer inspection revealed that this trace contains many Get requests that have values in the varbind list and it seems like they these requests are simply echoed back when an operation fails. We also noted that almost all requests in trace *l05t01* use 0 as the request identifier - something rather dubious[1].

trace	int32	uint32	uint64	oct	oid	ip	null	exc
l01t02	48.1	3.2	-	39.6	0.3	8.6	0.2	-
l01t05	13.4	21.0	52.7	12.9	0.0	0.0	-	0.0
l02t01	22.4	45.1	18.4	11.7	2.4	0.0	0.0	0.0
l03t02	2.5	95.0	-	2.4	0.1	0.0	0.0	-
l04t01	0.7	0.5	98.8	-	-	-	-	-
l05t01	2.6	80.1	-	17.0	0.0	0.0	-	-
l06t01	37.9	23.8	7.5	30.7	0.0	0.0	0.0	0.0
l12t01	48.3	51.5	0.0	0.1	0.1	0.0	0.0	-

TABLE VI
BASE DATA TYPE USAGE (IN %)

Table VI shows strong dominance of integral data types. This is consistent with our earlier observation that message sizes are relatively small, even when GetBulk operations with 10 or 12 max-repetitions are used. Some traces contain in addition a significant portion of octet string data. However, it seems questionable why some readonly string objects (e.g., ifDescr) are retrieved over and over again.

H. Managed Objects

The varbind names contained in response messages can be classified by matching them to MIB modules with contain the object definitions. Table VII shows the percentage of varbind names that belong to the IF-MIB (IF), the BRIDGE-MIB (BR), the BGP4-MIB (BGP), and the ENTITY-MIB (ENT). The last two columns aggregate names belonging to Cisco (CIS) and Centerpoint (CP) MIB modules. Note that we removed the looping time-filter flow from trace *l03t02* since otherwise the RMON2-MIB would have reached 93.7%.

trace	IF	BR	BGP	HR	ENT	CIS	CP
l01t02	40.1	-	17.6	-	10.3	30.4	-
l01t05	99.7	-	-	0.0	-	-	-
l02t01	93.5	5.5	0.0	-	0.1	0.0	-
l03t02	33.3	65.1	0.0	0.0	0.0	0.1	-
l04t01	99.7	-	-	-	-	-	-
l05t01	80.1	-	-	-	-	-	17.0
l06t01	91.3	0.0	0.0	-	0.0	2.0	-
l12t01	50.4	0.0	-	47.7	-	-	-

TABLE VII
MIB MODULE USAGE IN RESPONSE MESSAGES (IN %)

The IF-MIB clearly dominates in our traces. Still, the traces show differences how the IF-MIB is used. The messages collected in trace *l01t05*, for example, retrieve regularly all 64-bit counters plus the interface names, their types, alias and discontinuity time. In trace *l06t01*, 32-bit counters dominate and the discontinuity indicator is ignored. Trace *l02t01* shows that all columns of the ifTable and the ifXTable are retrieved regularly (including deprecated and index columns). A more precise analysis how data retrieval takes place and how to infer the data retrieval logic of management applications is beyond the scope of this paper.

I. Notifications

Five traces contain traps or inform messages carrying event notifications. In trace *l02t01*, about 52.1% of the notifications are fan failure notifications that are repeated periodically. Some 42.2% are interface up/down notifications while the remaining notifications are HP and Avaya specific. In trace *l03t02*, we found that all notifications were reporting printer problems. Trace *l05t01* contains only Cisco notifications related to TCP session teardowns and configuration changes.

Trace *l06t01* has 26.1% BGP and 8.1% PIM routing related notifications. Some 20.0% are sensor threshold crossing notifications while 13.2% are Cisco notifications related to TCP session teardowns. Note, however, that 21.3% of the notifications in this trace could not be analyzed due to a data conversion error. Trace *l12t01* contains 68.5% authentication failure notifications and 14.8% cold start plus 16.7% shut down notifications.

VI. Related Work

Several recent papers discuss the performance of SNMP [3]–[5], [7], [8], [16]. This work is complementary as it aims at providing empirical data about the usage of SNMP in production network. This data is needed to select realistic scenarios and models for evaluating SNMP performance. Perhaps some of the papers mentioned above need to be revisited once we better understand how SNMP is used in production networks.

A static (compile time) analysis of MIB module definitions is reported in [17]. One conclusion was that MIB modules

[1] We checked the other traces and found that some SNMP stacks use random request identifier while other simply count the request identifier, which means they are predictable.

contain a large number of integral data types. So far, our traces also show a strong usage of integral types. The analysis in [17] also showed that advanced router vendors prefer these days to define 64-bit counters and it seems that 64-bit counters are also preferred in some of the traces. The compiler back-end described in [17] makes certain assumptions about the behavior of SNMP implementations, for example to predict likely encoding sizes. The work reported in this paper helps to provide a basis for these assumptions.

The anonymization performed by the `snmpdump` tool is similar to the approach described in [18] in the sense that we deal with the anonymization of complete payloads. However, in contrast to [18], our tool is specialized to deal with SNMP traffic and we are not interested to regenerate frames.

VII. Conclusions and Future Work

After more than fifteen years of operational experience with SNMP, it is important to capture and analyze SNMP traffic traces to learn how SNMP is used in practice. Such knowledge is valuable for IETF protocol and MIB designers to understand which protocol features are used (or not used) in practice, equipment vendors for optimizing their SNMP implementations, tool developers to identify potential improvements in their tools as well as researchers who compare new management technologies to that of SNMP or analyze modifications of SNMP.

This paper presents an approach to capture and analyze such traces. As part of our research, we have developed some analysis tools, which are openly available, and we collected traces from several different locations. Our results show that SNMP is primarily used for monitoring, and not for configuration purposes. Despite the fact that the IETF declared SNMPv1 and SNMPv2 as *historic* and SNMPv3 as *standard*, we were unable to capture many SNMPv3 packets. Some locations still rely on SNMPv1, whereas others have moved to SNMPv2 to take advantage of the `GetBulk` operation to improve performance and 64-bit data types. Although in many cases the flow of SNMP data is relatively constant, there are also locations where SNMP is used in some very short bursts. We also observed that a significant portion of SNMP walks use a single varbind element and that discontinuity objects are seldom used to deal with counter discontinuities. We found that the `IF-MIB` is by far the most popular MIB module and that integral data types are heavily used.

This paper describes work in progress. More research is needed to develop statistically sound traffic models and investigate, for example, errors, use of advanced protocol options, retrieval of outdated MIB objects and the way current tools implement table retrieval. Preliminary results indicate that significant performance improvements are possible.

The most important step, however, is to collect and analyze more traces. We hope this paper turns out to be useful in convincing operators of the merits of our research and to collect more traces. Partners within the European IST-EMANICS Network of Excellence (NoE) already agreed to collect additional traces; other operators will be approached via the IRTF-NMRG.

VIII. Acknowledgments

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. We would like to thank SURFnet and the other network operators who gave us the possibility to collect and analyze traces. Thanks also to Tiago Fioreze for his support.

References

[1] V. Cerf, "IAB Recommendations for the Development of Internet Network Management Standards," Network Information Center, SRI International, RFC 1052, Apr. 1988.

[2] D. Harrington, R. Presuhn, and B. Wijnen, "An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks," Enterasys Networks, BMC Software, Lucent Technologies, RFC 3411, Dec. 2002.

[3] C. Pattinson, "A study of the behaviour of the simple network management protocol," in *Proc. 12th IFIP/IEEE Workshop on Distributed Systems: Operations and Management*, Nancy, Oct. 2001.

[4] X. Du, M. Shayman, and M. Rozenblit, "Implementation and Performance Analysis of SNMP on a TLS/TCP Base," in *Proc. 7th IFIP/IEEE International Symposium on Integrated Network Management*, Seattle, May 2001, pp. 453–466.

[5] A. Corrente and L. Tura, "Security Performance Analysis of SNMPv3 with Respect to SNMPv2c," in *Proc. 2004 IEEE/IFIP Network Operations and Management Symposium*, Seoul, Apr. 2004, pp. 729–742.

[6] T. Drevers, R. van de Meent, and A. Pras, "Prototyping Web Services based Network Monitoring," in *Proc. 10th Open European Summer School (EUNICE 2004) and IFIP WG 6.3 Workshop*, Tampere, June 2004, pp. 135–142.

[7] A. Pras, T. Drevers, R. van de Meent, and D. Quartel, "Comparing the Performance of SNMP and Web Services based Management," *IEEE electronic Transactions on Network and Service Management*, vol. 1, no. 2, Nov. 2004.

[8] G. Pavlou, P. Flegkas, S. Gouveris, and A. Liotta, "On Management Technologies and the Potential of Web Services," *IEEE Communications Magazine*, vol. 42, no. 7, pp. 58–66, July 2004.

[9] K. McCloghrie and F. Kastenholz, "The Interfaces Group MIB," Cisco Systems, Argon Networks, RFC 2863, June 2000.

[10] R. Raghunarayan, "Management Information Base for the Transmission Control Protocol (TCP)," Cisco Systems, RFC 4022, Mar. 2005.

[11] J. Schönwälder, "SNMP Traffic Measurements," International University Bremen, Internet Draft (work in progress) <draft-irtf-nmrg-snmp-measure-00.txt>, May 2006.

[12] R. Frye, D. Levi, S. Routhier, and B. Wijnen, "Coexistence between Version 1, Version 2, and Version 3 of the Internet-standard Network Management Framework," Vibrant Solutions, Nortel Networks, Wind River Systems, Lucent Technologies, RFC 3584, Aug. 2003.

[13] M. Harvan and J. Schönwälder, "Prefix- and Lexicographical-order-preserving IP Address Anonymization," in *10th IEEE/IFIP Network Operations and Management Symposium*, Apr. 2006, pp. 519–526.

[14] S. Waldbusser, "Remote Network Monitoring Management Information Base Version 2," RFC 4502, May 2006.

[15] T. Oetiker, "MRTG - Multi Router Traffic Grapher," in *Proc. 12th Conference on Large Installation System Administration (LISA XII)*, Boston, Dec. 1998.

[16] V. Marinov and J. Schönwälder, "Performance Analysis of SNMP over SSH," in *Proc. DSOM 2006*. Dublin: Springer LNCS 4269, Oct. 2006, pp. 25–36.

[17] J. Schönwälder, "Characterization of SNMP MIB Modules," in *Proc. 9th IFIP/IEEE International Symposium on Integrated Network Management*. IEEE, May 2005, pp. 615–628.

[18] R. Pang and V. Paxson, "A High-level Programming Environment for Packet Trace Anonymization and Transformation," in *Proceedings of the ACM SIGCOMM 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. ACM, 2003.