Received January 9, 2018; | Revised April 17, 2018; June 26, 2018; | Accepted July 24, 2018

DOI: 10.1002/nem.2044

RESEARCH ARTICLE

On the design and performance evaluation of automatic traffic report generation systems with huge data volumes

Carlos Vega*[1,2] | Eduardo Miravalls-Sierra[2] | Guillermo Julián-Moreno[1] | Jorge E. López de Vergara[1,2] | Eduardo Magaña[1,3] | Javier Aracil[1,2]

[1]NAUDIT, High Performance Computing and Networking, S.L., Parque Científico de Madrid C/Faraday 7, 28049 Madrid, Spain

[2]Departamento de Tecnología Electrónica y de las Comunicaciones, Escuela Politécnica Superior, Universidad Autónoma de Madrid, C/Francisco Tomás y Valiente 11, 28049 Madrid, Spain

[3]Departamento de Automática y Computación, Universidad Pública de Navarra, Campus Arrosadia, 31006, Pamplona, Navarra, Spain

Correspondence
*Carlos Gonzalo Vega Moreno, Email: carlos.vega@naudit.es, carlosgonzalo.vega@predoc.uam.es

Present Address
Parque Científico de Madrid. Calle Faraday, 7. 28049. Madrid

Summary

In this paper we analyze the performance issues involved in the generation of automated traffic reports for large IT infrastructures. Such reports allow the IT manager to proactively detect possible abnormal situations and roll out the corresponding corrective actions. With the ever-increasing bandwidth of current networks, the design of automated traffic report generation systems is very challenging. In a first step, the huge volumes of collected traffic are transformed into enriched flow records obtained from diverse collectors and dissectors. Then, such flow records, along with time series obtained from the raw traffic, are further processed to produce a usable report. As will be shown, the data volume in flow records turns out to be very large as well and requires careful selection of the Key Performance Indicators (KPIs) to be included in the report. In this regard, we discuss the use of high-level languages versus low-level approaches, in terms of speed and versatility. Furthermore, our design approach is targeted for rapid development in commodity hardware, which is essential to cost-effectively tackle demanding traffic analysis scenarios. Actually, the paper shows feasibility of delivering a large number of KPIs, as will be detailed later, for several TBytes of traffic per day using a commodity hardware architecture and high-level languages.

KEYWORDS:
Traffic analysis, Network management, Automatic reports.

1 | INTRODUCTION

Service outages are one of the primary concerns of any datacenter or network manager. Using a sample of 69 datacenters in 43 institutions, it turned out that the average cost per minute of datacenter outage was 7,908 USD[1]. To prevent such incidents, traffic analysis is a fundamental activity in datacenter or network management, which proactively helps identifying potential sources of trouble, before they happen.

Actually, the high-speed nature of current networks, together with the ever-increasing amount of new systems and applications, makes traffic analysis a difficult task. As it turns out, it is not a matter of analyzing a few GBytes worth of traffic with a traffic sniffer any longer, but digesting hundreds of GBytes of traffic and providing a quick diagnosis of the incident or proactive analysis. Such a digestion process primarily consists of funneling the incoming packets into flow records, which, in turn, serve to produce a usable report.

Such flow records are not restricted to the usual netflow, IPFIX or sFlow [2] (sampled flow) parameters, but also include transport and application layer statistics. The former are mostly network and transport layer parameters related to volumetry, such as number of bytes transmitted between two IP endpoint through given TCP ports. The latter also include the HTTP and DNS response time or the number of TCP zero window announcements from the client or server, which have an impact in the perceived Quality of Service (QoS).

As the network bandwidth increases, so does the amount of flow records and the computing power required to process and summarize them into a human readable report, thus becoming a very challenging task. In the past, the amount of flow records was manageable in a standard desktop and the analyst would simply use scripts to process the flow records, build the corresponding graphs and tables and finally produce the report. Nowadays, the amount of flow records has grown so large that automatic traffic reporting systems running on powerful servers are in order.

We believe that generation of traffic reports is cornerstone for traffic analysis. Needless to say, huge distributed databases, such as Hadoop systems, can be used for this purpose. However, such approach comes at a large hardware cost and is unpractical to deploy in a real network, taking into account that rack space and power are limited. Consequently, we advocate for a vertical scalability approach and use multi-core workstations instead of a fully distributed storage and processing system. Such multi-core workstation can also be used as a traffic capture probe with a high-speed driver [3]. Thus, the same workstation can be used for traffic capture, analysis and report generation purposes. Furthermore, we note that network operations' teams have strict budgetary restrictions and cannot afford a large processing infrastructure to produce traffic reports. Actually, the traffic report generation system should be fully optimized to make the most of commodity workstations.

Such automatic traffic report generation systems pose significant research challenges for their design and implementation, the most important of which is to deal with a huge volume of (enriched) flow records and produce a timely report in a cost-effective manner. Needless to say, reducing delivery time is a must for realistic network operations since, in case of an incident, a very fast response from the network operations center should follow.

In this paper, we provide an extensive performance evaluation of the different processing and visualization techniques that can be used to automatically produce the report from the flow records. The state of the art shows considerable activity in packet capturing, and also in delivering flow records from the packets. However, as will be shown in Section 1.1 there is scarce literature on the performance issues involved in the next step of transforming such flow records into usable information, namely, into a human readable report, especially when the volume is very large.

In this regard, we contribute to the state of the art by analyzing the design alternatives for automatic report generation from traffic flow records and provide an extensive trace-driven performance evaluation. To this end, we discuss the Key Performance Indicators (KPIs) involved in a traffic report and how to obtain them efficiently from the flow records. Traces have been obtained from real commercial environments, thus providing a realistic traffic load. Our findings serve as a guidance to other researchers and practitioners in the field in their efforts to implement automatic traffic report generation systems, capable to handle the vast amount of data seen in current networks and datacenters.

1.1 | State of the art

Several approaches for traffic report generation systems have appeared in the literature, but to the best of our knowledge, either they are based on dashboards or they lack a performance evaluation with a large volume of traffic.

Those that are based on dashboards provide a visual representation of network activity, which is usually supplied in real-time. As such, a dashboard is not comparable with a traffic report neither in traffic volume being treated nor in insights provided. Actually, dashboards are interactive, providing search capabilities, zoom in and out features, etc., which implies that the data volume handled by the dashboard remains within reasonably small values. As a result, the aggregation level of the offered traffic statistics is necessarily coarse. In the realm of dashboard-based tools, works such as ntop [4] already provided at the beginning of the century a network traffic probe that captured packets to monitor network usage and displayed generic real-time statistics with a web-based interface. To this end, as noted before, packets were summarized into traffic flows (for instance at the transport level), which, in turn, serve to produce records that feed the dashboard.

Such an approach of transforming packets into flow records and then feeding the dashboard, possibly in real-time, is very challenging at high-speed. In this light, alternate approaches based on Netflow records are appealing because the data volume is drastically reduced, albeit only network-level statistics can be provided. For instance, NfSen project developed NFDUMP tools [5] to capture and process these records, generating flow-based statistics that were also displayed in a web interface. In a similar vein, SiLK [6] also provided a set of tools to collect, store and analyze network flows based on NetFlow or IPFIX. This

type of records has been considered so useful that even a query language named NFQL [7,8] has been defined for them. However, we note that transport and application level statistics are missing in Netflow records. Consequently, they are targeted towards volumetric analysis, rather than quality of service assessment.

Given the limitations of dashboard-based approaches, especially those built on top of Netflow records, the research community focused on extending the Netflow capabilities by providing further insight into higher layers, such as Tstat [9]. This tool is able to generate records including TCP, UDP, and several upper layer protocols, such as TLS, RTP, DNS or HTTP. Moreover, it is a dashboard-based tool, namely it provides a web interface to visualize data. The authors of this tool have also studied how to aggregate statistical data from these records in [10]. However, although they assess the error in their aggregation, they do not evaluate the time it takes to generate such statistics, especially with a large number of records. Furthermore, statistics are provided as tables and graphs, and not as a report.

We also note that there are commercial traffic sniffers (Netscout, Riverbed, Viavi, etc.) that provide traffic reports. However, such systems are not open-source and the internals are not described in the scientific literature.

Another example of tool that generates enriched records is Tranalyzer [11]. Such tool can generate general statistics per protocol, and it can be used to feed standard tools such as RRD or Splunk, and even yield a PDF report, in the same way we present in our work. However, in their tests they have only documented the analysis of packet capture files as large as 44 GBytes, and reported the processing time for a small 1.3 GByte trace, whereas we are focusing on the analysis of very large captures with several TBytes worth of daily network traffic, as it usually happens in a real-world scenario.

As shown, the state of the art (summarized in Table 1) features dashboard-based and reporting systems that handle a moderate volume of traffic traces or flow records. However, in a real-world datacenter, the collected data volume is much larger. For instance, in a large datacenter of a Spanish logistics company we collected a grand total of 18 TB worth of traffic (packets) and 298 GB worth of flow records on Thursday, November 9th, 2017. Furthermore, as reported by other authors, such traffic

TABLE 1 Summary of the State of the Art

Solution	Last Version	Purpose and Features	Drawbacks
ntop/ntopng	2001-today	Probe to monitor network usage with generic real-time statistics, providing throughput and bytes, grouped by hosts or applications.	It does not provide distribution functions nor percentiles for these values, and important performance records are missing.
sFlow	2001-2004	Industry standard technology for monitoring switched networks.	Limited to layer 2 and 3 of the OSI model. Uses mandatory sampling to achieve scalability.
Netflow records	1996-today	Reduce data volume from raw data into flow records.	Only network-level statistics can be provided. Transport and application level statistics are missing.
NFDUMP from NfSen	2005	Capture and process Netflow records generating flow-based statistics.	Uses Netflow records, and therefore only network-level statistics can be obtained.
SiLK tools	2003-today	Collect, store and analyze network flows based on NetFlow or IPFIX.	Transport and application level statistics are missing in Netflow records. Aimed to volumetric analysis, rather than QoS.
NFQL	2013	A language to process flow records, aggregate them into groups, apply absolute or relative filters.	Uses Netflow records, consequently, they are targeted towards, volumetric analysis, rather than QoS.
Tstat	2002-2016	Dashboard-based tool that generates records including TCP, UDP, and several upper layer protocols, such as TLS, RTP, DNS or HTTP.	There is no evaluation of how long it takes to generate such statistics, especially with a large number of records. Statistics are provided as tables and graphs, and not as a report.
Netscout, Riverbed, Viavi etc.	-	Commercial traffic sniffers which provide traffic reports.	Such systems are not open-source and their internals are not described in the scientific literature.
Tranalyzer	2016	Generates enriched records and PDF reports, general statistics per protocol, feeds other tools.	Performance evaluation with PCAP files as large as 44 GBytes, and processing time assessed for a small 1.3 GByte trace.

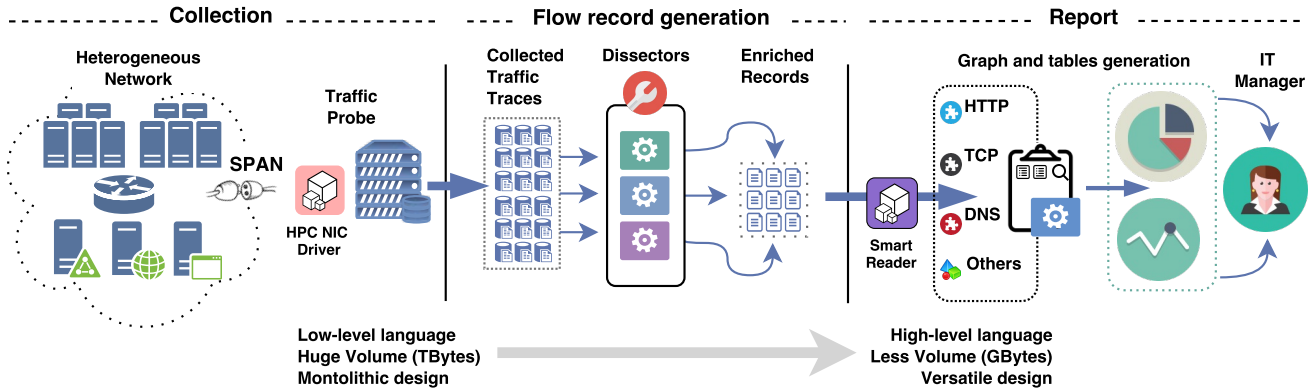


FIGURE 1 Typical deployment scenario for traffic collection and analysis, depicting the different stages involved in the process.

volumes are common in large datacenters [12]. Clearly, this is a different scale in terms of data volume, compared to the current state of the art. In this paper, we analyze how to cost-effectively handle such a huge amount of data and produce a usable report.

Actually, we provide insights into the design principles of automatic traffic report generation systems currently in production in large networks and datacenters, from our hands-on experience in the design and implementation of such systems [13]. The lessons learned help researchers and practitioners in the field to build efficient and useful automatic report generation systems.

1.2 | Deployment scenario

We consider a large IT infrastructure (datacenter, network, etc.) that performs IT service continuity management according to the ITIL methodology. To this end, a risk assessment (also known as risk analysis) is performed to identify the assets, threats, vulnerabilities and countermeasures for each IT service. As a part of such an assessment, a traffic analysis is conducted on a daily or weekly basis, by means of an automated traffic analysis report. Then, the appropriate corrective actions are rolled out, and the alarm-based real-time monitoring systems are updated accordingly.

Figure 1 shows the different stages of automatic report generation. In the data collection stage, a traffic probe receives a copy of traffic from certain VLANs or ports of interest through a SPAN or port mirror. Then, several dissectors act on the captured traffic in order to deliver enriched flow records, which correspond to the different protocols in the traffic. Which, for example, contain information about HTTP records such as the client, server, URL, response time, response code, etc. Finally, the report is delivered using such enriched records as the data input.

The above stages can be performed in a single appliance or in several appliances, either in series or in parallel. We advocate for the use of a single appliance whenever possible. If several appliances are used, huge volumes of data must be moved from one appliance to the other. For example, if an appliance is used for data collection and dissection and a second appliance for analysis and report generation, respectively, all the enriched records must be transferred from the first appliance to the second one. Concerning parallel deployment of appliances, namely load balancing, we note that evenly distributing the incoming data is a challenging task. For example, in [14] Shi *et al*. conclude that *"under certain Zipf-like flow-size distributions, hashing alone is not able to balance workload"*, with regard to Internet flows. Furthermore, cost increases if several appliances are used. Thus, we strive for the maximum efficiency in the data collection, dissection and analysis stages, to include all of them in the same host.

We note that the above three stages are not necessarily concurrent. A traffic trace can be collected during the day, and then, dissection and analysis can be performed at nighttime. For many use cases, automatic report generation systems are not aimed for real-time operation and alert triggering, but for long-term proactive analysis.

2 | METHODOLOGY

In what follows, we will explain the methods involved in each of the aforementioned stages, as shown in Figure 1 . In the next section, we describe how to perform traffic sniffing at multi-Gigabit-per-second rates and generate enriched records and aggregate statistics using high-speed dissectors. Then, the selection of Key Performance Indicators (KPIs) follows.

2.1 | Collection

Firstly, the corporate network traffic must be captured through either ad-hoc hardware or optimized drivers capable of receiving network traffic at 10 Gbit/s such as the HPCAP network driver[3] for Intel® Ethernet NICs. As for traffic dissection, the traffic trace should be summarized into enriched records that contain amenable information for the analysis. In this regard, there is a trade-off between accuracy and real-time delivery of the enriched flow records.

We note that the online delivery of flow records is extremely challenging, if not impossible, at high-speed rates, especially if the application layer is to be inspected. Consequently, in some cases, the departure point for report generation is not the set of enriched records produced in real-time, but the traffic trace itself, which is subsequently processed to obtain enriched records. For example, a precise count of TCP retransmissions can only be obtained by fully reassembling the TCP connection and, then, searching for a possible retransmission for every incoming packet that belongs to the TCP connection. This is a very processing intensive task, which is unfeasible to perform live at high speed and should be performed offline.

We note that traffic analysis is an interactive top-down process. First, high-level statistics are analyzed and, then, the data is drilled-down in order to perform further investigation. As it turns out, chances are that the traffic trace has to be read again, which is costly in terms of I/O and adds significant processing time to the report generation. This is the case when inspecting the traffic from a particular IP that showed anomalous behavior in flow records, and calls for further analysis at the packet level.

In any case, the first step is traffic dissection and delivery of flow records, either online or offline. Traffic dissectors provide enriched records per service (HTTP, DNS, SQL, SMB, etc.), per protocol (TCP, UDP, ICMP), etc. Furthermore, they also provide time series that show the temporal evolution of a given statistic, for example, traffic in bits per second, in and out a given VLAN or subnetwork. As it turns out, there are high performance traffic dissectors available in the state or the art[15,16] and we will not focus on them in this article.

2.2 | Flow records generation

Generally speaking, enriched flow records can be classified into stateless and stateful.

2.2.1 | Stateless flow records

The previously described flow records usually feature absolute counters that are not context-aware, counters for whose estimation it's not required additional knowledge from previous or future events. Typically, they report global statistics such as the number of packets or bytes transmitted per flow. Usually, such metrics can be delivered online, even at high rates.

2.2.2 | Stateful flow records

In addition, more elaborated metrics require awareness of previous states, such as those related to conversations, connections or sessions. Such metrics require higher amount of resources and are more time-consuming, hampering real-time delivery. In the worst-case scenario, they may require multiple read passes over the analyzed trace.

For example, some traffic probes[3] produce enriched records in real-time that provide the number of zero-window announcements from a given client or server in a TCP flow. However, we note that not only the zero-window announcement events matter, but the maximum time that the client or server was not opening that window. If a given client or server announces a zero-window just once in a TCP connection, then it does not matter that much. Conversely, if such an announcement is held for 10 seconds, chances are that the server is heavily saturated. However, the client or server blocked time (i.e. time it takes to the host to open the window back again) should be obtained offline, to reduce the probe processing requirements while capturing and storing traffic at high speed.

2.3 | Report

The report is made from the previous flow records, which are read either from disk or online from the flow generation stage. First, graphs and tables are produced and, then, the report itself is delivered. In order to deliver useful visualizations, it is always good practice to follow simple rules[17] about the appropriate way to represent data. Second, report typesetting can be performed by means of a typesetting language such as LaTeX, which is open source and allows creating a formatted document from a text file. Other output formats such as plain text can also be delivered in order for the automatic traffic generation system to feed other management tools.

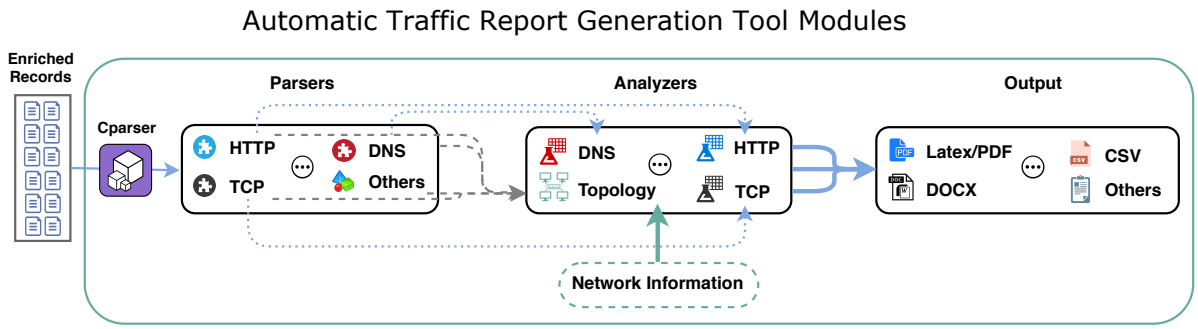


FIGURE 2 Diagram depicting different internal modules from the automatic traffic report generation tool.

Processing speed is a crucial requirement in this reporting stage. We note that reading TBytes of data from hard disk is very time consuming. Consequently, the less read passes over the trace the better performance. Thus, it becomes cornerstone that the enriched records obtained in the previous stage are sufficient to produce the graphs and tables, which imposes restrictions to the KPIs to be included in the report. Ideally, the traffic trace is read from hard disk just once, then is dissected into records as it is read, and, finally, the necessary graphs and tables for the report are produced. Thus, most of the KPIs should be obtained from enriched records, without getting back to the trace itself for further statistics. For example, obtaining a detailed time series for a given host, which has been identified in the enriched records analysis, implies reading the traffic trace a second time, which is very costly in terms of time.

Even in the case that enriched records are sufficient to produce the report, their fields of interest must be extracted and arranged in memory. As noted before, in a large datacenter, a one-day worth of traffic may be around several TBytes and the enriched records may occupy several tens of GBytes. In order to obtain the report graphs and tables, such enriched records are heavily accessed during the report generation process. Therefore, they should be cached in memory and not retrieved from disk if possible, to avoid penalizing performance significantly.

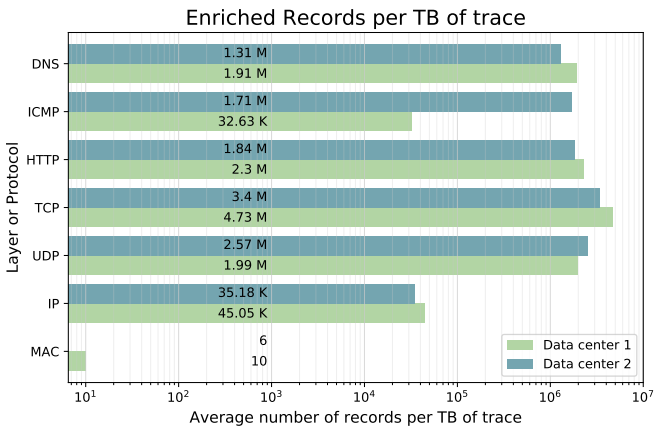
3 | ARCHITECTURE

To better understand the challenged involved in providing a real implementation of the architecture described above maps into a practical implementation it is convenient to estimate the amount of traffic and enriched records typically produced in a datacenter, as follows.

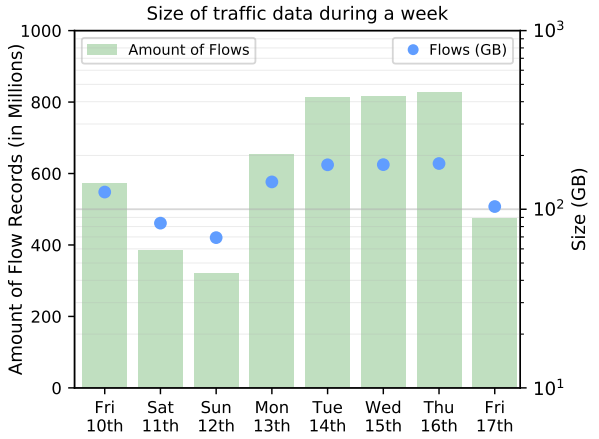
3.1 | Data volumes

Figure 3 a shows the volume of enriched records per TB of traffic trace for two different datacenters, for MAC/IP/TCP/UDP/HTTP flows with an average record length of 29 ± 44 fields and 195 ± 121 bytes per record. The enriched records' length ranges from less than 10 bytes for MAC records to more than 1000 bytes for TCP records. The former usually consists of a few counters reporting the number of packets or bytes transmitted from each endpoint, while the latter has dozens of metrics for both directions, such as number of retransmissions or duplicates. Consequently, a careful capacity planning of the host memory is in order. To this end, knowledge of the approximate volume of enriched records per TB of traffic trace becomes fundamental. Figure 3 b shows the amount of flow records (similar to [3, pg. 10] flows) collected per day in a large Spanish logistics company datacenter, showing that a number of 600-800 million of flow records per day can be expected, which make up 200 GBytes approximately in size.

The figure shows that for a large datacenter several tens of millions of records can be produced per day. Thus, a trade-off between high-level programmability and efficiency has to be sought.



(a) Enriched records per TB of trace from different companies. Labels indicate the value of each bar.



(b) Amount of flows records for each weekday and their corresponding size in GigaBytes. Data was collected in a large datacenter of a Spanish logistics company.

FIGURE 3 Statistics from different enterprise networks.

3.2 | Implementation considerations

In this section, we provide considerations for the practical implementation of automatic traffic reporting systems, based on the architecture explained above.

Clearly, the collection, dissection and analysis stages have very different needs from the programming perspective. As we climb from the wire to the charts (see Figures 1 and 2) we tend to aggregate the information, from the individual packets to the enriched records and finally the charts in the automatic report. During the collection process, kernel-level ad-hoc software provides maximum performance to avoid packet loss during the capture process. Afterwards, high-speed dissectors swiftly extract the information into text files, possibly offline.

These two first stages usually prioritize non-functional requirements such as performance, resilience, efficiency and stability and tend to evolve slowly with few changes during its lifecycle due to its monolithic and fit-for-purpose design. For example, protocol traffic dissectors are not modified often since protocols such as TCP do not change frequently.

However, the analysis process requires higher level of abstraction and complex functions more prone to blow with the wind depending on the analysts and IT manager needs. These are often functional requirements such as analyzing new metrics or drawing additional charts. Thus, such software layers usually require non-functional requirements such as extensibility, modularity and versatility. Besides, as seen in Section 1.1, most solutions are limited to network-level metrics from layers 2 and 3, and analyzing deeper layers requires faster processing due to the increase of information the suppose.

Hence, in order to tackle with the various analysis requirements mentioned above, we have considered *Python* due to its versatility and portability, making use of high-level statistical libraries such as *Pandas*[18] and *Numpy*[19], which allowed for a rapid development of our system while keeping good performance. Clearly, compiled languages such as C or C++ would definitely provide better performance but at the cost of excessive development time.

However, we can avoid that excessive development time and take advantage of the extra performance and leaner footprint of the C language by using the C-Python interface in performance-critical sections. This has proven extremely useful in our solution, as we found that Python had an unacceptable poor performance (both in terms of memory usage and speed) when reading all the records from the files, and that alternatives within Pandas or Numpy could not support all our data types.

3.2.1 | Cparser

To circumvent the above performance barrier, we developed a C module (`cparser`[20]) that provided faster reading of the records and lower memory usage compared with Python (see Fig. 4). On top of this module, we implemented another module that provides common data operations (filters, aggregations, maps, top values, etc.) that may be chained efficiently, aching the data

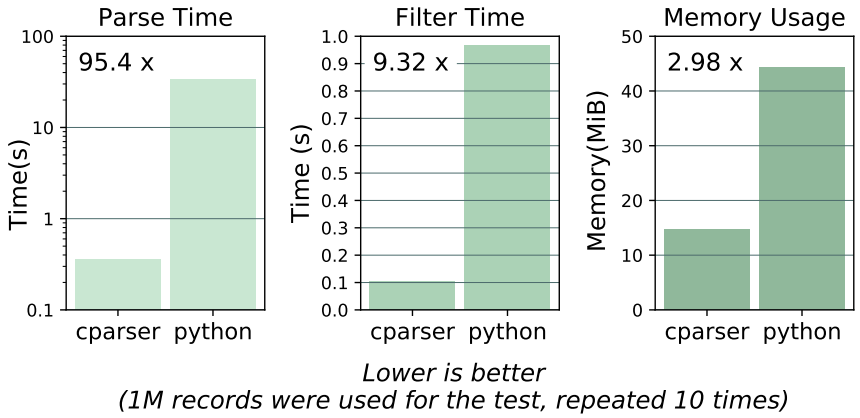


FIGURE 4 Performance comparison between cparser and standard Python I/O libraries.

and deferring read operations until needed. The cparser code is available to the community, as an additional contribution of the paper, and can be used for network data analysis or other data analysis purposes.

Figure 4 shows the performance comparison between cparser and the standard Python I/O libraries conducted with synthetic records with metrics of different kind (i.e. IPs, floats, integers, etc.).

The figure shows the read times for one million records, with 10 runs per experiment. As shown, the decrease in the read time is striking with our cparser solution.

The improved performance brought by cparser implies that the time spent during the record parsing is approximately 1% of the total execution time (8 minutes). Without cparser, our heaviest reports took a full day to generate. Instead, cparser reduces the generation time to 9 hours and 30 minutes.

3.2.2 | Graph generation

Since many different kinds of charts and visualization libraries are available, it is not a trivial task to choose the right way to display information, especially when we require chart automation. We have studied and extensively tested the features of the most popular visualization libraries such as *bokeh*[21], *ggplot*[22] (based on *The Grammar of Graphics*[23]), *gnuplot*[24], *tikz*[25], *matplotlib*[26] or *seaborn*[27] (based on matplotlib). We opted for *gnuplot* and *seaborn* for the main charts and *tikz* for the topology graphs.

The latter are well suited for our requirements of speed, integration, versatility, and customization. We were able to produce a large variety of custom charts such as survival distribution function plots, box plots, time series, pie charts and topology graphs that reshape themselves depending on the data. For example, they allow resampling any time series in a given time span.

We look forward to the progress of projects such as PyPy[28], Numba[29] or Grumpy[30] which aim to dilute the boundaries between high and low level programming languages[31], bringing together the strong abstraction of the former while keeping the high performance of the latter. We note that they still require further refinement and wider compatibility but the trend of high level languages with faster just-in-time compilers looks promising for the upcoming challenges of automatic traffic reporting and network management.

3.3 | Description of the tool modules

In the light of the above we proceed to briefly describe different modules of the automatic report generation tool depicted in Figure 2 . All these modules make use of the different methodologies abovementioned to avoid slow behaviour and improve speed and versatility such as single-pass analysis, deferment and cache of operations.

First, the aforementioned module cparser is the core module to improve the performance of I/O operations done from the enriched records. A series of parsers provide a data model of the different kind of records such as HTTP, TCP, etc. with their corresponding data types. These parser classes inherit functionality from a class called SmartReader which executes and caches the different transformation operations (i.e. read, filter, aggregate, etc.) through the cparser module.

Each parser is used in their corresponding analyzer (or multiple if needed, as in topology analysis) to perform their corresponding analysis (i.e. get the top server IPs by TCP retransmissions or the top URLs with server errors) following the described

advices. The omniscient module Network Information gathers common information from the records such as the network topology (i.e. subnets, VLANs, NAT information, etc.) used by multiple analyzers to determine the network wellness.

While the analysis is performed, the different analyzers output their results in different forms (i.e. tables, charts) through the output module which typesets a report in one or multiple formats, should they be requested.

4 | MAIN KEY PERFORMANCE INDICATORS

The previous sections served to explain the report generation methodology and the data volumes involved. As noted before, the selection of KPIs is conditioned by the limiting factors stated above of single-pass analysis and arrangement of data files in memory. Needless to say, the basic volumetry indicators (top clients and servers, conversations, ports, etc.) can be easily obtained from the enriched records using hash tables stored in memory. More elaborate KPIs may actually require several passes to the enriched records, which could penalize performance.

We will not provide a description of all the KPIs for the sake of brevity. Instead, we will focus on original KPIs that can be obtained with moderate processing resources, and that provide valuable information to an IT manager.

4.1 | Burst analysis and identification of root cause

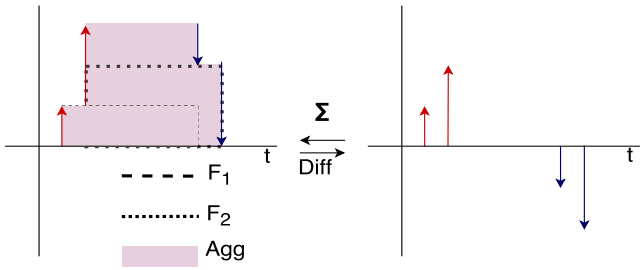
Burst are useful to the IT manager to identify potential saturations, for example, a saturated link shows a sustained traffic burst at its line rate. However, time resolution is cornerstone in order to identify short bursts and perform the appropriate proactive actions before they become significant.

In order to identify such bursts (see Fig. 5 a) or plateaus, we use the time series of bits per second of the whole trace, with one-second time resolution, which is obtained from the first pass to the trace. We note that such a time series cannot be obtained from the flow records. Then, a simple heuristic for burst detection based on throughput threshold and duration above threshold is adopted. Finally, we provide the candidate burst root cause, which is of interest to identify what happened.

To this end, we perform the matching of a set of time series from different metrics (HTTP response time, number of UDP flows, etc.) against each of the bursts from the bits per second time series of the trace, looking for correlations in the variation of the metrics. The metric with highest variability in the burst time interval is chosen as the candidate root cause (as in principal component analysis), and the endpoints identified in the root cause metrics are highlighted in the report. This variability is computed using a rolling centered window, assigning to each window the standard deviation of its values normalized by its mean.

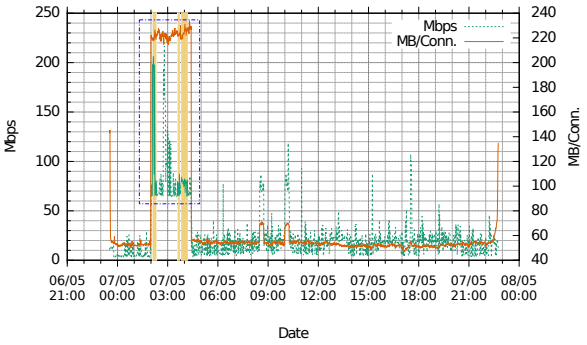
For instance, Figure 6 a depicts an example of traffic burst. Bursts are marked with a yellow area, and in this case, the candidate root cause of the traffic burst is an increase of traffic per connection (MB/Connection line). As shown, connections unexpectedly grow in size, which may be indicative of an anomaly.

(a) Example situations for the burst detection algorithm: detected bursts (green), extended bursts (dotted yellow) and rejected bursts because of low duration and/or average speed (dashed red). Orange lines represent the detection thresholds.

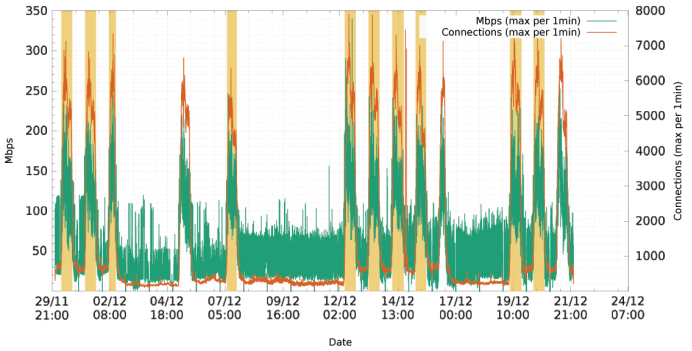


(b) Reconstruction of a time series. The figure depicts an example with two flows, in which the reconstruction is performed by using the cumulative sum of these impulses.

FIGURE 5 Representation of some methods used for the analysis of time series.



(a) In the span marked with a dotted blue rectangle, a traffic burst (dotted green) is caused by a sudden increase of the traffic per connection (orange line).



(b) Burst detection in a real report and corresponding cause time series. Highlighted regions correspond to bursts detected with a rate of 100 Mbps.

FIGURE 6 Real examples of traffic bursts.

Figure 6 b shows another example in which several bursts are caused by an increased number of connections. Based on these correlations, the software isolates the flows that were observed during the burst time and shows the clients with most connections in that period. This allows the manager to see not only why a burst happened, but also which nodes in the network were responsible for it.

4.2 | Time series reshaping from enriched records

Time series are essential to understand the dynamics of a given statistics, such as for example evolution of consumed bandwidth. Nevertheless, we note that an accurate time series cannot be obtained from enriched records[32] only. For example, if we wish to obtain the traffic in bits per second in and out from a given IP and port, the enriched records will only provide the average traffic per flow, for those flows that have that IP and port as an endpoint.

Therefore, we can recreate the time series we are looking for by adding up such averages as step functions in the intervals where the flow is alive (see Fig. 5 b). This method assumes that the bandwidth consumed during the flow lifetime is uniform. Then, we add up the step functions to yield an approximate time series for the specific metric. However, this is just an approximation of the real-time series of the input and output traffic for a given set of IP addresses and port numbers. In fact, the resulting traffic from the composition of such step functions is less *bursty* than the real one. Hence, we cannot display the traffic peaks accurately, which is cornerstone for some services. For example, detection of peaks in Citrix[33] traffic is a very important issue as such peaks produce instantaneous queuing delays that are noticeable by users, even in small time scales.

In this example, we note that the time series to be obtained cannot be programmed beforehand. First, a flow-based analysis is performed that provides the hosts with potential problems (for instance a high retransmission rate per flow). Then, a detailed traffic time series can be obtained for that particular host in order to look for possible traffic anomalies, such as micro-outages in the scale of seconds, that cannot be derived from the enriched records directly. However, the latter requires a second pass to the trace.

4.3 | Red-Amber-Green summaries

A Red-Amber-Green (RAG) diagram provides a high-level performance evaluation of a given system, service or link. To this end, summaries of specific metrics are conducted for different layers and protocols such as DNS, TCP, HTTP, etc. For example, the following TCP metrics are taken into account for a TCP service RAG analysis: zero-window announcements, retransmissions, duplicate ACKs, SYN attempts, downtimes, RTT, connection establishment time, and the number of connections and the relative contribution in bytes to the TCP traffic.

Several thresholds and heuristics are used to characterize the server health in different groups, for instance, a time interval between the SYN-ACK and the ACK in the TCP handshakes serves to estimate the RTT. If such RTT is higher than 1 second for 5 consecutive minutes or shows a significant departure from the mean (e.g. a sudden tenfold RTT increase), then an anomaly is signaled for the RTT metric.

TABLE 2 Key Performance Indicators used for the RAG tables and their corresponding conditions. When the conditions are fulfilled, a value is added up to the server score. Servers with highest scores are shown first in the RAG table.

Protocol	KPI	Trigger value	Score to be added
TCP servers	Duplicate ACK Src. To Dst.	5% per connection	2 per unit
	Duplicate ACK Dst. To Src.	5% per connection	2 per unit
	Retransmissions Src. To Dst.	5% per connection	2 per unit
	Retransmissions Dst. To Src.	5% per connection	2 per unit
	Zero Window Dst. To Src.	5% per connection	2 per unit
	Downtimes	SYNs from source to server without packets from the server and no data packets from the source during 90% of the time	25 if true
	Connection Establishment Time (s)	CET $\geq$ 0.1s during 5 consecutive minutes OR a 5 min spike 10 times higher than the mean value	50 if true
	Round-Trip Time (s)	RTT $\geq$ 1s during 5 consecutive minutes OR a 5 min spike 10 times higher than the mean value	50 if true
	Ignored / Denied SYNs	Used to measure the importance of the server	0.1 per unit
	Number of Connections	Used to measure the relevance of the server. -1 if there are no records with SYNACK but more than 1000 with SYN	0.01 per unit OR a total of 10 if -1
	Bytes transmitted	Used to measure the importance of the IP server	0.1 per unit
HTTP servers and ports	Server Errors (%)	5% of Server Errors relative to the server	$\frac{3 \cdot Server\ Errors\ (\%) \cdot Transactions}{100}$
	Client Errors (%)	20% of Client Errors relative to the server	$\frac{Client\ Errors\ (\%) \cdot Transactions}{100}$
	Median Response Time (s)	Greater than or equal to 0.1 seconds	50 if true
	Avg. Response Time (s)	Greater than or equal to 0.5 seconds	50 if true
	Acc. Response Time (%)	Used to measure the importance of the HTTP server and its corresponding port	2 per unit
	Transactions	Used to measure the importance of the HTTP server and its corresponding port	1 per unit
DNS servers	Errors (%)	5% of Errors relative to the server	2 per unit
	Median Response Time (ms)	Greater than or equal to 100 ms	50 if true
	Avg. Response Time (ms)	Greater than or equal to 500 ms	50 if true
	Acc. Time (%)	Used to measure the importance of the DNS server	1 per unit
	Transactions	Used to measure the importance of the DNS server	1 per unit

Table 2 summarizes most of the KPIs used for the RAG tables of the traffic report, together with a brief description of the trigger values that indicate an anomalous behavior of the KPI metric. Each of these trigger values, if fulfilled, adds up a score value to the corresponding server. The final RAG table features the servers sorted by the described score and colored according to the severity of their status.

5 | EVALUATION

For performance evaluation purposes, we collected two groups of real network traces from different multinational corporations.

These two groups have different traffic characteristics. The 6.9 TB (10,036 Million Packets) of traffic traces from **Dataset 1** were captured from two different interfaces at the edge of a company network. **Dataset 2** was captured in the core distribution layer of a different company network, providing 2.1 TB (3,693 Million Packets) of traffic traces.

Following the collection of the network traces, they were summarized into enriched records[1] by specialized dissectors. Such records contain different statistics depending on the dissected layer, and the amount of information is very variable as well. The MAC or IP conversations associated records naturally provide coarser information than the TCP counterpart. Application

[1]Examples available on request.

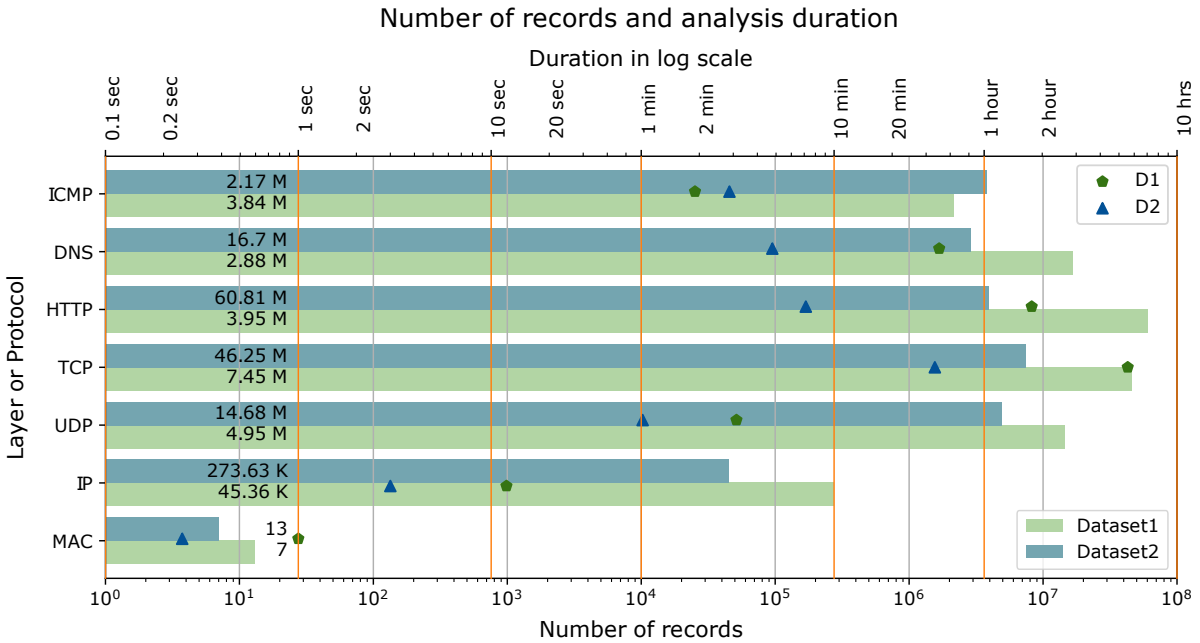


FIGURE 7 Amount of enriched records and their corresponding processing time for each layer/protocol in different datasets.

protocols usually have more text-based fields such as the HTTP URI or the requested host during a DNS query, with an impact on both the record length and memory consumption during the analysis.

The resulting enriched records are also depicted in Figure 7 , showing the amount of records for each layer or protocol. As expected, **Dataset 1** yields more records than **Dataset 2** due to its size.

5.1 | Results

As noted before, some layers and protocols produce higher amounts of records with different number of fields and length. Consequently, their corresponding stages are more time consuming and memory hungry. Thus, the first experiment to be considered is the sequential execution of each analysis stage for each dataset in order to find out which stages require more memory or processing time.

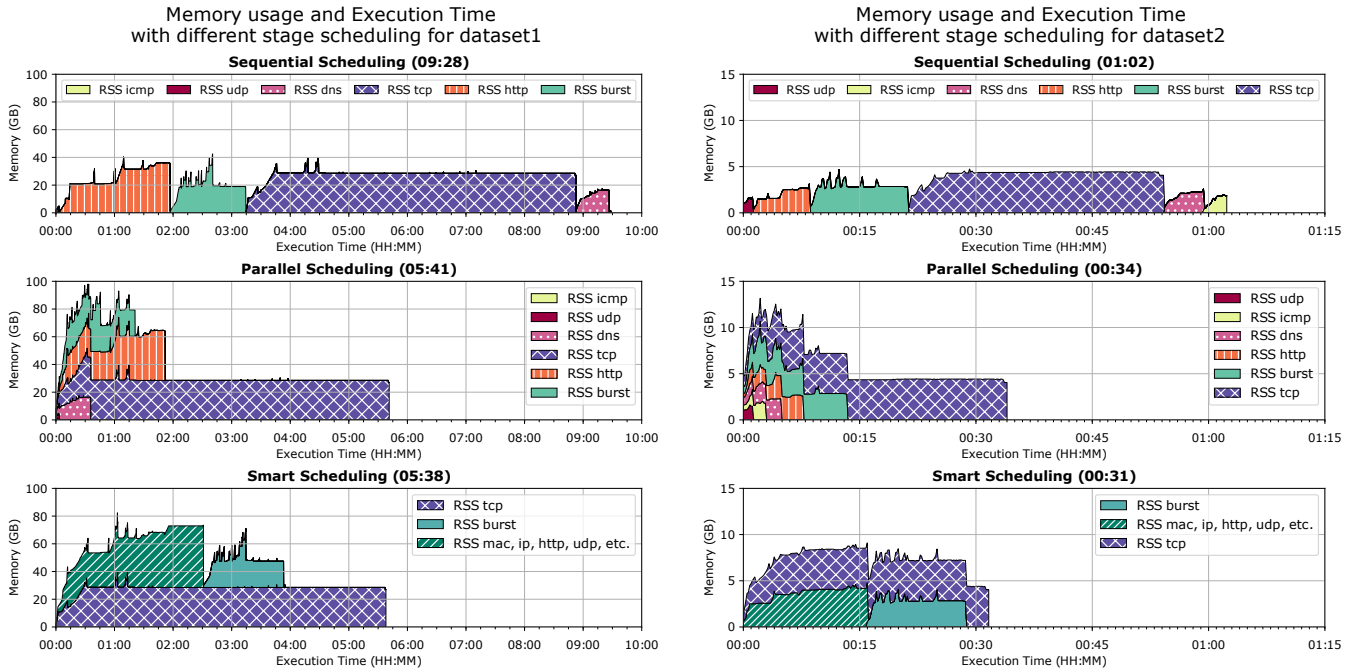
These results serve to drive conclusions about how some stages can be parallelized in order to optimize both time and memory usage for different scenarios. Therefore, the second experiment involves testing different stage scheduling for the optimization of memory usage and execution time, looking for a compromise between both. Decreasing the execution time is essential to provide a report that covers the daily behavior of the IT infrastructure, within a reasonable execution time, and then roll out the corrective actions, should they be needed.

We set our objective to the goal of processing the corresponding enriched records from 1 TByte worth of traces in less than a quarter of a day, because the traffic volume from large data centers may reach some TByte per day on average as seen in the **Dataset 2**.

The report features several sections, such as topology; MAC, IP, UDP conversations; TCP connections; HTTP, DNS transactions, ICMP messages, among others. The report has more than 75 tables and 48 figures, with about 10 tables and charts per section. Such prototype system was tested on an Intel ® Xeon E5-2640 v3 @ 2.6Ghz host with 128GB of RAM and two RAIDs 0 storages of 16TB each that provides read and writing speeds up to 10 Gbit/s, ideal for traffic sniffing at multi-Gigabit-per-second rates. This setup is similar to those that can be found in datacenter network probes.

A single RAID system was used to store the enriched records for the analysis. We note that our system, as many other network analysis systems, is favored by the vertical scalability. Thus, a faster processor would provide much better performance.

Sequential execution of the system confirmed (see Figure 7) that some sections, such as TCP, are prone to require more time owing to the complexity of the metrics. As Figure 8 shows, the sections have different memory usages as well. All of this might lead to some form of stage starvation in which shorter stages await for a very long time, instead of being processed



(a) Dataset 1 **(b)** Dataset 2

FIGURE 8 Memory usage and execution time for different stage schedulings. Short stages are not shown.

in parallel along with the most resource consuming tasks or prior to them. These results helped us to identify certain sections of the report generation that could be delivered earlier if they were executed with a better scheduling.

5.2 | Better scheduling

In this light, we conducted tests for each dataset with different stage schedulings. The modularity of the tool allows the network manager to schedule the tasks in a particular order based on his expert knowledge (which stages lasted longer in previous analysis or which have more data) to improve the performance. Thus, the tool has been designed with stage modularity and section modularity within each stage, that allows the network manager to disable some stages or sections within stages.

In the first place, we used a scheduling in which all stages run in parallel. Afterwards, we tried a hand-crafted smart scheduling which gathers the least resource hungry stages in one process and queues the longest (such as burst and TCP stages) in another process, being both processes executed in parallel. These approaches are compared to the sequential scheduling in Figure 8 . As shown, execution times are halved in both parallel and smart scheduling. However, memory usage is very intensive for the former, while the latter provides a good compromise in both time and memory usage. Furthermore, we note that the very same behavior can be observed for both datasets. Therefore, we note that smart ordering of execution of the different report sections is cornerstone to achieve a good performance.

6 | CONCLUSIONS

In this paper, we have shown the challenges of automatic traffic reporting systems with the ever increasing large data volumes, which provide the final step of transforming flow records into a human-readable report. Such flow records not only encompass layer 2-3 parameters, as traditional monitoring solutions do, but also include transport and application layer statistics, which make the data volume even larger. As such, the data processing scenario is very challenging, from the low-level I/O routines performance, to the appropriate selection of KPIs and data visualization modules.

In this paper, we have provided a number of recommendations and design tips for complete network data analysis that are most valuable to other researchers and practitioners in the field for the efficient design of automatic traffic reporting systems.

We have also contributed with the release of the cparser[20] module for faster I/O operations that can be used for network data analysis or other data analysis purposes.

More specifically, we have shown that the use of a high-level language, complemented by low-level I/O routines and adequate data analysis methodologies, provides ample functionality and enough performance to cope with large volumes of data in commodity hardware. As shown in the trace-driven performance evaluation with real-world traffic traces, our solution has exceeded our initial performance goals, being able to analyze significant volumes of data in a timely manner.

ACKNOWLEDGMENTS

This work has been partially supported by the Spanish Ministry of Economy and Competitiveness and the European Regional Development Fund under the projects TRÁFICA (MINECO/FEDER TEC2015-69417-C2-1-R) and Procesado Inteligente de Tráfico (MINECO/FEDER TEC2015-69417-C2-2-R). The authors also thank the Spanish Ministry of Education, Culture and Sports for a collaboration grant.

References

1. Institute Ponemon. *Cost of datacenter Outages, datacenter Performance Benchmark Series.* : Ponemon Institute; 2016. http://www.ponemon.org/blog/2016-cost-of-data-center-outages.

2. sFlow.org . *sFlow: sampled flow.* https://sflow.org; [last accessed: 26-June-2018].

3. Moreno Victor, Río Pedro M, Ramos Javier, et al. Multi-granular, multi-purpose and multi-Gb/s monitoring on off-the-shelf systems. *International Journal of Network Management.* 2014;24(4):221–234. http://doi.org/10.1002/nem.1861.

4. Deri Luca, Carbone Rocco, Suin Stefano. Monitoring networks using ntop. In: :199–212IEEE; 2001. https://doi.org/10.1109/INM.2001.918032.

5. Haag Peter. Watch your Flows with NfSen and NFDUMP. In: ; 2005. http://meetings.ripe.net/ripe-50/presentations/ripe50-plenary-tue-nfsen-nfdump.pdf.

6. NetSA CERT. SiLK: CERT NetSA security suite: Monitoring for large-scale network http://tools.netsa.cert.org/silk; .

7. Bajpai Vaibhav, Schauer Johannes, Schönwälder Jürgen. NFQL: A tool for querying network flow records. In: :643–649IEEE; 2013. http://ieeexplore.ieee.org/document/6573045/.

8. Bajpai Vaibhav, Schönwälder Jürgen. Network Flow Query Language - Design, Implementation, Performance, and Applications. *IEEE Transactions on Network and Service Management.* 2017;14(1):8–21. https://doi.org/10.1109/TNSM.2016.2633511.

9. Finamore Alessandro, Mellia Marco, Meo Michela, Munafo Maurizio M, Di Torino Politecnico, Rossi Dario. Experiences of internet traffic monitoring with tstat. *IEEE Network.* 2011;25(3). http://dx.doi.org/10.1109/MNET.2011.5772055.

10. Colabrese Silvia, Rossi Dario, Mellia Marco. Aggregation of Statistical Data from Passive Probes: Techniques and Best Practices. In: :38–50Springer; 2014; Berlin, Heidelberg. http://dx.doi.org/10.1007/978-3-642-54999-1_4.

11. Burschka Stefan, Dupasquier Benoît. Tranalyzer: Versatile high performance network traffic analyser. In: :1–8IEEE; 2016. https://doi.org/10.1109/SSCI.2016.7849909.

12. NetSA CERT. *Cisco Global Cloud Index: Forecast and Methodology 2016-2021.* http://www.cisco.com/c/dam/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.pdf; 2018.

13. HPCN Naudit. *FERMIN: Factual Executive Report of a Monitored IP Network.* http://www.naudit.es/fermin; 2018.

14. Shi Weiguang, MacGregor Mike H, Gburzynski Pawel. Load balancing for parallel forwarding. *IEEE/ACM Transactions on Networking (TON).* 2005;13(4):790–801. https://doi.org/10.1109/TNET.2005.852881.

15. Deri Luca, Martinelli Maurizio, Bujlow Tomasz, Cardigliano Alfredo. nDPI: Open-source high-speed deep packet inspection. In: :617–622IEEE; 2014. http://doi.org/10.1109/IWCMC.2014.6906427.

16. Vega Carlos, Roquero Paula, Aracil Javier. Multi-Gbps HTTP traffic analysis in commodity hardware based on local knowledge of TCP streams. *Computer Networks.* 2017;113:258–268. http://doi.org/10.1016/j.comnet.2017.01.001.

17. Rougier Nicolas P., Droettboom Michael, Bourne Philip E.. Ten Simple Rules for Better Figures. *PLOS Computational Biology.* 2014;10(9):1-7. http://doi.org/10.1371/journal.pcbi.1003833.

18. Augspurger Tom, Bartak Chris, Cloud Phillip, et al. *Pandas: Python Data Analysis Library.* http://pandas.pydata.org/; [last accessed: 26-June-2018].

19. SciPy. . *NumPy.* http://pandas.pydata.org/; [last accessed: 26-June-2018].

20. HPCN. Naudit. *Cparser.* https://hpcn-uam.github.io/ccore/; [last accessed: 26-June-2018].

21. Bird Sarah, Canavan Luke, Mari Maggie, et al. *Bokeh.* https://bokeh.pydata.org; [last accessed: 26-June-2018].

22. Yhat . *Ggplot.* http://ggplot.yhathq.com/; [last accessed: 26-June-2018].

23. Wilkinson Leland. The grammar of graphics. In: Berlin, Heidelberg: Springer 2012 (pp. 375–414).

24. Williams Thomas, Kelley Colin. *Gnuplot.* http://www.gnuplot.info/; [last accessed: 26-June-2018].

25. Tantau Till, FeuersÃď nger Christian. *Tikz.* http://pgf.sourceforge.net/; [last accessed: 26-June-2018].

26. Hunter John, Dale Darren, Firing Eric, Droettboom Michael, team . *Matplotlib.* https://matplotlib.org/; [last accessed: 26-June-2018].

27. Waskom Michael. *Seaborn.* https://seaborn.pydata.org/; [last accessed: 26-June-2018].

28. Rigo Armin, Fijalkowski Maciej, Bolz Carl Friedrich, et al. *PyPy.* https://pypy.org/; [last accessed: 26-June-2018].

29. David Potts Antoine Pitrou. *Numba.* http://numba.pydata.org/; [last accessed: 26-June-2018].

30. Dylan Trotter Mansour Rahimi, team . *Grumpy.* https://github.com/google/grumpy; [last accessed: 26-June-2018].

31. Marowka Ami. Python accelerators for high-performance computing. *The Journal of Supercomputing.* 2018;74(4):1449–1460.

32. Sommer Robin, Feldmann Anja. NetFlow: Information loss or win?. In: :173–174ACM; 2002. http://dx.doi.org/10.1145/637201.637226.

33. Schlosser Daniel, Staehle Barbara, Binzenhofer Andreas, Boder Björn. Improving the QoE of Citrix Thin Client Users. In: :1–6IEEE; 2010. http://doi.org/10.1109/ICC.2010.5501891.

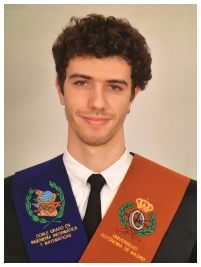
AUTHOR BIOGRAPHY

Carlos G. Vega Moreno received his M.Sc. and Ph.D. (*cum laude* and industrial distinctions) degrees in Computer Science from Universidad Autónoma de Madrid, Spain, in 2014 and 2017. In 2012, he joined the HPCN Research Group at the same university as a researcher, collaborating in different European research projects such as the Network of Excellence InterNet Science, Fed4Fire or dReDBox. His current research at Naudit includes log collection and network traffic analysis.

carlosgonzalo.vega@predoc.uam.es

Eduardo Miravalls Sierra received his double B.Sc. degree in Computer Engineering and Mathematics in 2016 and his M.Sc. in Computer Science in 2017 from Universidad Autónoma de Madrid, Spain. He joined the High-Performance Computing and Networking Research Group at the same university as an intern in 2015 where he collaborated on the European research project Fed4Fire and the Racing Drones national research project. Right now, he is working at GMV in GNSS projects.

eduardo.miravalls@uam.es



Guillermo Julián Moreno received his double B.Sc. degree in Computer Engineering and Mathematics from Universidad Autónoma de Madrid, Spain, in 2016, and his M.Sc. in Computational Science and Engineering at École Polytechnique Fédérale de Lausanne, Switzerland, in 2018. He has been collaborating in the High-Performance Computing and Networking research group at UAM since 2013, and now works in its spinoff Naudit HPCN in network capture drivers and network analysis.

guillermo.julian@naudit.es

Jorge E. López de Vergara MÃľndez is associate professor at Universidad Autónoma de Madrid (Spain), and founding partner of Naudit HPCN, a spin-off company devoted to high performance traffic monitoring and analysis. He received his M.Sc. and Ph.D. degrees in Telecommunication Engineering from Universidad Politécnica de Madrid (Spain) in 1998 and 2003, respectively. He researches on network and service management and monitoring, having co-authored more than 100 scientific papers about this topic.

jorge.lopez_vergara@uam.es

Eduardo Magaña Lizarrondo received his M.Sc. and Ph.D. degrees in Telecommunications Engineering from Public University of Navarra (UPNA) (Spain), in 1998 and 2001, respectively. He is an associate professor at UPNA. During 2002 he was a postdoctoral visiting research fellow at the Department of Electrical Engineering and Computer Science, University of California, Berkeley. His research interests are network monitoring, traffic analysis and performance evaluation of communication networks.

eduardo.magana@unavarra.es

Javier Aracil Rico received the M.Sc. and Ph.D. degrees (Honors) from Technical University of Madrid in 1993 and 1995, both in Telecommunications Engineering. In 1995, he was awarded with a Fulbright scholarship and was appointed as a Postdoctoral Researcher of the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley. In 1998, he was a research scholar at the Center for Advanced Telecommunications, Systems and Services of The University of Texas at Dallas. He has been an associate professor for University of Cantabria and Public University of Navarra and he is currently a full professor at Universidad Autónoma de Madrid, Madrid, Spain. His research interests are in optical networks and performance evaluation of communication networks. He has authored more than 100 papers in international conferences and journals.

javier.aracil@uam.es