TRANSACTIONS ON EMERGING TELECOMMUNICATIONS TECHNOLOGIES
Trans. Emerging Tel. Tech. 2012; **00**:1–17
DOI: 10.1002/ett

RESEARCH ARTICLE

Lightweight Enhanced Monitoring for High-Speed Networks

R. Vilardi[1], L. A. Grieco[1*], C. Barakat[2] and G. Boggia[1]

[1]DEE - Politecnico di Bari, Via Orabona, 4, 70125 Bari, Italy.
Mail:{r.vilardi,a.grieco,g.boggia}@poliba.it
[2]INRIA, Sophia Antipolis, 2004, route des Lucioles, 06902 Sophia Antipolis Cedex, France.
Mail:Chadi.Barakat@inria.fr

ABSTRACT

In this paper, LEMON, a lightweight enhanced monitoring algorithm based on packet sampling, is proposed. It targets a pre-assigned accuracy on bitrate estimates, for each monitored flow at a router interface. To this end, LEMON takes into account some basic properties of the flows, which can be easily inferred from a sampled stream, and it exploits them to dynamically adapt the monitoring time-window on a per-flow basis. Its effectiveness has been tested using real packet traces. Experimental results show that LEMON is able to finely tune, in real-time, the monitoring window associated to each flow and, compared to a classic fixed-scale monitoring approach, it is able to better satisfy the accuracy requirements of bitrate estimates. Moreover, its communication overhead can be kept low enough by choosing an appropriate aggregation policy in the message exporting process. Finally, LEMON produces a low processing overhead, which can be easily sustained by currently deployed routers. Copyright © 2012 John Wiley & Sons, Ltd.

*Correspondence
Dipartimento di Elettrotecnica ed Elettronica, Politecnico di Bari, Via Edoardo Orabona, 4, 70125 Bari, Italy. Mail: a.grieco@poliba.it

1. INTRODUCTION

In network measurement systems, sampling techniques can greatly reduce packet processing overhead [1, 2, 3]. Using them, the properties of the traffic to analyze can be inferred only from a subset of packets, each one captured from the original stream with a packet sampling probability p [3]. In particular, when packet sampling is turned on at a monitored network interface, the packets in transit are firstly sampled with probability p. Then, each sampled packet is classified and processed, as belonged to a subset of packets called *flow*: the corresponding packet or byte counters are increased, and other metrics of interest are updated [4]. In this context, the definition of a *flow* is encoded using a *flow key*, which encompasses one or more packet header fields, depending on the metering process. Finally, as done in CISCO NetFlow [5] and IP Flow Information eXport (IPFIX) systems [6, 7], the counters associated to the monitored flows are exported towards a collecting point, which is in charge of executing the required statistical analyses.

Nowadays, in flow-based monitoring systems (e.g., NetFlow and IPFIX), this exporting process is triggered by timers, which are statically established and commonly set in the order of some minutes or which can expire when no packets are observed for a long period with respect to a given flow. In this manner, traffic characteristics are estimated with a coarse and fixed time resolution [8].

For sake of clarity, we remark that an exporting timer lasting T seconds leads to a bitrate estimate binned over an averaging window with the same size T. For this reason, from now on, the terms *bin size*, *monitoring window*, and *exporting timer* will be used interchangeably.

Approaches based on a large or fixed T pose serious limitations on the possibility to conceive management tools that are fast enough to detect anomalies in real time. The majority of the applications, in fact, recognize an anomalous event (e.g., caused by attacks or network failures) long after this event occurs: thus, they are not able to recognize the event while it is still in progress. This is due to the static and coarse nature of the measurements provided by the monitoring systems [9].

To this aim, it is also worth to mention that the estimation of a flow bitrate (averaged over a time bin with duration T) using a sampled stream leads to a loose of accuracy, which is tightly coupled to the properties of the traffic itself, the sampling probability p, and the monitoring window T [10]. This problem critically emerges for monitoring applications that require a fine grained time resolution (i.e., small T) with a very low processing overhead (i.e., small p), so that the parameters p and T have to be carefully tuned for each single flow to avoid to compromise the effectiveness of the overall measurement

Prepared using ettauth.cls [Version: 2012/06/19 v2.10]

system [11]. In other words, the adoption of a fixed monitoring window T is not feasible in all conditions and could generate unacceptable estimation errors of the traffic bitrate.

In addition, we highlight that high frequency components of traffic, which are irreversibly lost using a large T, can bring information very useful for a deep investigation of flows' properties. This aspect becomes very relevant for traffic engineering systems in networks [12, 13] and OpenFlow boxes [14, 15], where decisions on rerouting are based on variations in the traffic bitrate.

The same remark applies to anomaly detection systems [16, 17], which could infer shifts from traffic if out-profile behaviors are caused by malicious activities, or by failures in network equipments or protocols. The early and accurate detection of these anomalies allows quick and precise countermeasures.

Traffic anomalies are sudden events that usually cause a deviation from what is considered as normal behavior in the traffic flows. These events, due to network equipment failures, flash crowd occurrences, security attacks (e.g., denial of service attacks), external routing changes, are named "volume-based" anomalies [18]. There are also the so-called "non volume-based" anomalies, e.g., the port scanning attacks, in which server ports are scanned in order to exploit the vulnerability of a particular service.

In general, the unfavorable effect of the sampling on the effectiveness of the detection is more visible as the value of the corresponding sampling probability decreases, considering both "volume-based" and "non-volume-based" detection techniques. Packet sampling introduces two main effects that bias the distortion of the data and affect the goodness of the detections. At high sampling probability (with a low percentage of lost packets), the flow thinning effect (wherein a multi-packet flow is reduced to a single packet flow by random packet sampling [17]) is the main factor that increases the number of false positives. At low sampling probability, instead, the global reduction in the number of flows (or the global reduction in the distribution of the flows) causes a decrease of the major events observed and detected. Moreover (for a reasonable value of the sampling probability), if "volume-based" metrics are considered as anomaly index (i.e., packet counts and byte counts), the packet sampling does not affect the number of the measured anomalies. Otherwise for "non-volume-based" anomalies, packet sampling heavily impacts on the flow counts, affecting the number of detected anomalies [16]; for example, in port scanning the detection metric is strictly related to the number of ports queried by the server probe, and thus to the number of detected flows. "Entropy-based" techniques, that analyze changes in the distribution of traffic communication pattern, are generally more resilient to packet sampling than "volume-based" metrics [16].

One possible solution to these problems would consist in adopting dedicated measurement hardware, able to catch as many packets as necessary [8, 19]. But this requires additional equipments and inflates the costs of the monitoring systems. Conversely, approaches that leverage on the concept of entropy could nicely fit the requirements of anomaly detection systems [20]. But, unfortunately, they could hardly be applied to traffic engineering.

Another alternative is to use other kind of packet/byte-counting methods like, for example, the Simple Network Management Protocol (SNMP). As well known, the SNMP protocol is based on the query/response paradigm and exchanges management information between two software entities: the SNMP applications (running on network management devices) and the SNMP agents (running on external network devices). Differently from the flow-based monitoring systems, SNMP is a highly complicated protocol to implement: the design of SNMP agents and also their administration are difficult and usually entail individual configuration and habitual maintenance [21]. Moreover, SNMP is not a particularly efficient protocol. Bandwidth is wasted with needless information included into each exchanged message (i.e., the SNMP version and multiple length and data descriptors). Also, the way the variables are identified leads to wastefulness of large data that consume substantial parts of each SNMP message [22].

In our humble opinion, it is still possible to take advantage from flow-based monitoring and packet sampling techniques and, at the same time, to grant for an accurate traffic estimation. In line to this principle, we present herein a novel algorithm for traffic monitoring at the flow level based on packet sampling, which will be referred to as Lightweight Enhanced MOnitoring for high-speed Networks (LEMON).

The key features of LEMON are as follows.

- It works on a per network interface basis, that is, it is not a network wide system.
- It adopts a fixed value for the sampling probability p and it adapts the value T (i.e., the exporting timer for each monitored flow) in order to ensure a given target estimation accuracy of the bitrate to each monitored flow at a given interface.
- It represents the bitrate as a signal over the time and it models its estimation accuracy by using a Signal-to-Noise Ratio (SNR) concept.
- It links the desired accuracy (i.e, the target SNR) to the parameters p and T by adopting closed form expressions, recently devised in [10].
- It requires only minimal modifications to the NetFlow implementation; in particular, a dynamic adaptation of exporting timers on a per-flow basis and a slightly larger memory needs are required.

In a previous work [23], a first implementation of the LEMON algorithm was proposed. Two analytic expressions have been used to model the SNR, derived with a frequency based approach in [10]. In the present paper, a further step ahead has been done by extending in several directions the preliminary system in [23]: (i) one more model for the SNR, derived in [10] by using a

Trans. Emerging Tel. Tech. 2012; **00**:1–17 © 2012 John Wiley & Sons, Ltd.
DOI: 10.1002/ett
Prepared using ettauth.cls

classic stochastic approach, is adopted; (ii) a wider set of traces is considered in order to provide a better evidence of the LEMON effectiveness; (iii) three different strategies for message exporting are evaluated in terms of their communication overhead; (iv) a comparison with a fixed resolution approach is accomplished to motivate the need for LEMON in realistic settings; (v) an analysis of the complexity of LEMON is carried out (i.e., memory and CPU requirements are evaluated) to highlight its relevance in real-time monitoring systems.

Within this paper, the effectiveness of LEMON is demonstrated using two kinds of real traffic traces, one captured on Asia transpacific links (thanks to the MAWI project) [24] and the other one captured on an access router attached to a DSLAM with about a thousand of customers behind.

Results have shown that: (i) LEMON is able to tune, in real-time and and in a fine manner, the monitoring window associated to each flow; (ii) compared to a classic fixed-scale monitoring approach, LEMON is able to better satisfy the accuracy requirements imposed on the SNR of the bitrate estimates; (iii) the timing of the exporting process and, hence, the rate of messages sent to the collector (i.e., the "communication overhead") can be kept low enough by choosing an appropriate aggregation policy; (iv) LEMON produces only a low processing overhead, which can be easily sustained by currently deployed routers, such as the CISCO 12000 device. All these results show that LEMON can be a promising tool for traffic monitoring in high-speed networks.

The rest of the paper is organized as follows. In the next section, the related work is provided. In Sec. 3, the integration of LEMON within a real monitoring device, compliant to IPFIX exporting specifications, is discussed. Then, Sec. 4 widely describes all details of LEMON, which is then experimentally tested in Sec. 5 using real network traces. Finally, the last section draws conclusions and forecasts future research.

2. RELATED WORK

In high-speed networks, anomaly detection, network tomography, traffic engineering, traffic characterization, and intrusion detection techniques require accurate traffic measurements in order to pursue their targets [25, 4, 26, 27, 28]. At the same time, the huge volume of traffic carried by high-speed backbones and the increasing heterogeneity of new applications, if not properly handled, would cause an unsustainable growth of the overhead required by such measurements. As a matter of fact, from one side, it is necessary: (i) to process, classify, and characterize a large amount of data with strict temporal resolution and accuracy; (ii) to allow an effective analysis of traffic composition; and (iii) to monitor the network infrastructure trend or users' behavior. On the other side, the overall measurement process should not waste further precious communication and processing resources.

As testified by the huge number of studies on the topic [29, 28], the aforementioned challenges are at the center of an ebullient panorama of research activities, which are currently attracting a wide community of scientists all over the world. This is confirmed by the number of research projects which are involved in traffic measurement and analysis, for example see the projects: CAIDA* (Cooperative Association for Internet Data Analysis), MAWI† (Measurement and Analysis on the WIDE Internet), IPMA‡ (Internet Performance Measurement & Analysis), TMA§ (Traffic Monitoring and Analysis), and mPlane¶ (Intelligent Measurement Plane for Future Network and Application Management).

At the present, anomaly detection, security applications, traffic accounting, performance analysis, and network planning are, among others, top research trends [30, 4, 25, 31]. All these sophisticated applications need to access basic information about the monitored traffic (e.g., packet/flow counters, flow bitrates, flow/packet timestamps, flags and/or protocol ports) which can be extrapolated by considering active [32, 33], in-line [34, 35] or passive [36, 8, 19] measurements.

An active measurement system feeds the observed network with test packets, which are collected at one or more sinks and processed in order to infer the metrics of interest. The weakness of this class of techniques is related to the load generated by test packets, which, if not properly handled, could interfere with the traffic to monitor and thus could compromise the accuracy of the considered metering applications [32, 37]. Moreover, different probing streams generated from several measurement devices at the same time could interfere to each other by severely impairing the effectiveness of the measurement system [38].

The in-line approach, conceived for IPV6 environments only, considers measurement devices as native parts of the network and charges them with the task of monitoring the traffic and encoding the results directly into ad hoc packet headers [34, 35]. As example, in [35] it is proposed to transport measured metrics by using the native and unused fields of the IPv6 protocol at the network layer. In this way, it is possible to perform end-to-end unidirectional measurements, like packet delay and packet loss. Anyway, it is worth to note that in-line techniques require additional and/or unused packet headers that could not be available in all network device settings.

Techniques based on passive measurements appear very appealing since they do not require the injection of probing traffic into the monitored network nor the usage of dedicated packet header fields. They observe the traffic as

* See http://www.caida.org/.
† See http://mawi.wide.ad.jp/mawi/.
‡ See http://www.merit.edu/networkresearch/projecthistory/ipma/.
§ See http://http://www.tma-portal.eu/.
¶ See http://www.ict-mplane.eu/.

it flows through monitored interfaces that are periodically queried (or interfaces autonomously export the outcome of the measurement process) to let administrators access information [36, 39].

Originally, for what concerns this class of approaches, only simple packet/byte-counting methods have been used, such as in the Simple Network Management Protocol (SNMP).

Moreover, these very basic solutions, if alone, are no longer useful to fit the requirements of nowadays monitoring systems due to a poor accuracy [40]. Furthermore, in high speed links with rates of several gigabit per second, it is necessary to grant fast packet-processing at router interfaces and low communication overhead. Thus, internet service providers started to prefer the usage of lightweight flow-level tools (e.g., NetFlow and sFlow [41]) that can be easily embedded into routers, or other dedicated devices, and that can provide further information on the traffic (e.g., start/end or delta timestamps, packet/octet counters, and TCP flags counters) than simple counting methods [39].

Packet sampling techniques, used in conjunction with Cisco NetFlow, have also gained a lot of attention in recent years due to their high scalability and ease of implementation. In this direction, some interesting proposals have been conceived to tune the packet sampling probability p, based on the composition of the traffic mix [8, 42] or on the flow size [43].

A more challenging network-wide environment is considered in [44], where a novel cognitive monitoring approach is proposed by considering the cooperation of different network probes. Given a measurement task and a constraint on the volume of collected information, the system in [44] is able to set operating conditions of routers (i.e., the packet sampling rate at the different interfaces) in a centralized and adaptive way, by balancing measurement accuracy and system overhead. Unfortunately, this approach does not consider the real-time adaptation of the monitoring time resolution T given that it mainly focuses on flow volumes.

In [9], the attention is moved to end-host bandwidth measurements. A measurement tool is developed that succinctly summarizes bandwidth information and answers general queries at arbitrary resolutions without maintaining state for all time scales. With such a scheme, collected data can be handled using a relational database, thus allowing administrators to query off-line bandwidth statistics across links and time.

To conclude, all mentioned techniques have been designed assuming a fixed time resolution and/or using offline information processing schemes, but this could not fit the requirements of advanced monitoring applications (see also the discussion on the subject in the previous section). For this reason, the contribution of the present paper aims to complement such valuable research efforts by proposing a novel algorithm, able to adapt (in real-time) IPFIX (Netflow) exporting timers in order to grant a desired accuracy on the estimated bitrate of each monitored flow at the network interface of interest.

3. INTEGRATING LEMON IN A MONITORING SYSTEM

In this section, the functions of the LEMON algorithm are explaining by illustrating how it can be easily embedded in nowadays monitoring devices. To this aim, the main features of consolidated flow-level approaches, e.g., NetFlow and IPFIX, will be firstly summarized. Then, the role of LEMON in improving their performance will be highlighted.

Here, the term *flow* defines a set of packets, observed at a particular interface during a certain time interval, that share common properties (defined as *flow key*). These properties can be referred to packet headers (e.g., source/destination IP address, source/destination transport port, transport protocol type, packet length), or to packet payloads. Usually, a traffic monitoring device at the flow level is in charge of storing and handling all the information collected from each flow by using a *flow record table*. Such a table is updated by creating new entries, refreshing existing ones, and deleting or exporting information about expired flows.

Starting from the system guidelines of Cisco NetFlow v.9, IETF defined a standard way of exporting information related to IP flows, i.e., the IPFIX (IP Flow Information eXport) protocol [6, 7, 45].

The IPFIX reference architecture [46] is composed of interacting *monitoring devices* and *collectors*, both communicating using the IPFIX protocol (see Fig. 1). Usually, each monitoring device hosts the three following functional components [46].

- An *Observation Point* of incoming packets through one monitoring interface.
- A *Metering Process* for the creation and the management of flow records. This process is in charge of different functions, for example: to capture packets and parse their headers; to manage timestamps; to handle packet sampling and filtering; to classify flows; and to arrange IP traffic information in the *flow record table*.
- An *Exporting Process* for the transmission of IPFIX messages towards a *Collecting Process* at the collectors. Each message contains information about flows, such as the properties that characterize the definition of the flow key and the value of monitored counters. In this way, collectors can make the information on monitored flows accessible to network operators for further analyses and applications.

The exporting process usually applies to those flows considered as expired in the *flow record table* [46]. Their selection can depend on many events that include: the

Trans. Emerging Tel. Tech. 2012; **00**:1–17 © 2012 John Wiley & Sons, Ltd.
DOI: 10.1002/ett
Prepared using ettauth.cls

expiration of idle or active timeouts, configured by the metering process. In particular, a flow record is timed out either if no packets belonging to it have been observed for a time interval specified by the *flowIdleTimeout* variable, or if the *flowActiveTimeout* expires when the flow is still active [45].

In general, an exporting process is triggered also by: (i) traffic overloads, which may generate too many new flow record entries, more than what the memory capacity of a device can support; (ii) detection of packets carrying particular information (e.g., TCP headers with set FIN or RST flags, which indicate that a TCP session is ending). A monitoring system, equipped with IPFIX/NetFlow functionalities, is then able to provide traffic measures only at the end of the life of the monitored flows, or at coarse time scales (the default value of *flowIdleTimeout* is 300 s, the default *flowActiveTimeout* is 1800 s). This is a serious limitation for an effective traffic monitoring, because in this way one cannot counteract traffic problems in real time, while they are still in progress, but has rather to wait for the expiration of a timeout or for the end of the flows experiencing problems. In addition, such a coarse time resolution impedes to recover high frequency components of the traffic, which can be very relevant to advanced monitoring approaches, as explained in Sec. 1.

Concerning this, LEMON improves the value and the utility of flow-based measurements, because it is able to forward information about a flow also during its active period, with a finer time scales with respect to classic monitoring systems. In this way, it also makes possible to partially recover high frequency components of the traffic. To this end, a new per-flow timeout is defined in LEMON, i.e., the *flowBinTimeout*, which is compliant to the IPFIX information model guideline [45]. Such a timeout trips each time the actual time bin window T expires for a given flow; this allows the system to export the data collected for the considered flow during the last time bin. In the next Sec. 4, along with all details of the LEMON algorithm, it will be deeply described how this new timeout can be adaptively set in real time in order to grant for a target accuracy of bitrate estimates.

4. LEMON ALGORITHM

Herein, details about LEMON system are provided starting from its theoretical foundation.

4.1. Theoretical Considerations

In [10], three closed-form analytical equations were derived, each one modeling the estimation accuracy of the measurement system in terms of SNR (Signal-to-Noise-Ratio), that is, the ratio between the signal associated to the original bitrate (of a considered *i*-th flow) and the noise due to sampling operations. In other words, they represent the accuracy of bitrate measurements of a packet sampled flow, as a function of: the sampling rate p, the bin size T_i,

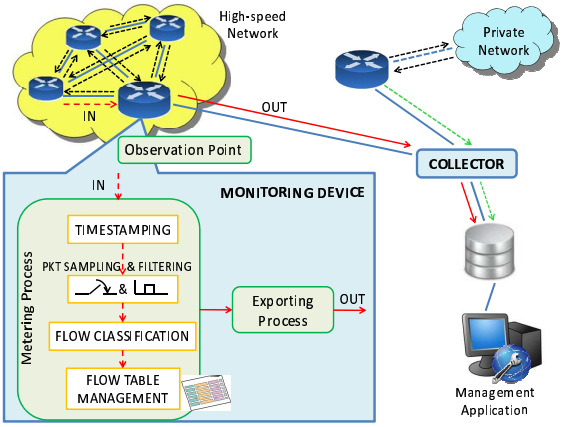


Figure 1. IPFIX reference architecture.

and some other metrics that can be estimated directly from sampled packets.

In LEMON, such models are used solving their representative equations with respect to the variable T_i (as described in the next subsection); this is made in order to dynamically adapt the observation window of each monitored flow and to ensure a *desired value* of SNR which, from now on, will be referred to as SNR_{th}.

The first SNR model assumes that all packets have the same constant packet size (CPS). With reference to the i-th flow, the expression for the SNR in the CPS model can be expressed as:

$$SNR_i = \frac{p}{1-p} \times \left[\frac{T_i \times C_i}{0.89} + 1 \right]. \qquad (1)$$

The second model, instead, explicitly takes into account a variable packet size (VPS). The expression for the SNR in this case is:

$$SNR_i = \frac{p}{1-p} \times \left[\frac{T_i \times C_i \times \overline{D}_i^2}{0.89 \times M_i} + 1 \right]. \qquad (2)$$

In the previous equations, with reference to the i-th flow and considering that the packet size can be seen as stochastic variable, $\overline{D}_i$ and M_i are the first and the second order moments of the packet size, respectively. Instead, C_i is the average packet transmission rate. All of these parameters are assumed to be stationary and have to be inferred from the sampled traffic of the i-th flow. As shown in [10], the 0.89 factor is obtained because a monitoring window with size T_i is modeled as a low-pass filter with a frequency band that is $0.89/T_i$ wide.

The third model also accounts for packets having different sizes, but it has been derived using a classic stochastic approach, so that it will referred to as "stochastic model" in the sequel; in this case, the SNR has the following expression:

$$SNR_i = \frac{p}{1-p} \times T_i \times C_i \times \frac{\overline{D}_i^2}{M_i}. \qquad (3)$$

Network administrators can take advantage of these models to tune, in real-time, the main parameters of the monitoring system, as T_i or p, in order to achieve the desired estimation accuracy and to trade off sampling overhead with frequency resolution [11]. In what follows, we will show how these models can be used to implement a real-time monitoring scheme, evaluating also their effectiveness.

4.2. Real-time adaptation of time resolution

As mentioned above, LEMON objective is to monitor the bitrate of packet sampled flows, which are identified by a flow key. Detected flows are handled using a table of flow records, so that every time a packet is captured and identified as belonging to a given flow, the relative counters are updated. Given a minimum performance level to ensure in terms of a SNR threshold, the time resolution modeled by the averaging time bin is not fixed, but it varies following the behavior of the monitored signal, as stated by the three analytical models presented above.

In order to handle a time-varying bin size, LEMON introduces a new per-flow timeout, namely *flowBinTimeout*, which is time-varying and adapts, in real-time, the frequency of the exporting process (and consequently the time-resolution of traffic measurements at the collectors). This per-flow timeout is properly defined according to the IPFIX information model guideline [45].

With reference to a given network interface of a router, LEMON samples incoming packets with a uniform probability[‖] p and it classifies them in several flows, according to the definition of a flow key. The *i*-th flow is monitored during observation windows named *bin time* T_i. The monitoring operations consist basically on counting the number of sampled packets and bytes of the i-th flow to infer an estimate of its bitrate.

A flow table is used to maintain flow records. Every time a packet is captured and identified as belonging to a flow, the relative flow counters are updated. Once the k-th time bin of the i-th flow, $T_i(k)$, expires, LEMON performs the estimation of the model parameters (i.e., $\overline{D}_i$, M_i, and C_i), updating the values calculated during the previous time bin with the contribution of counters at the last one. These operations are performed in order to adapt the time resolution to the variations of the bitrate associated to the considered flow.

Now, it is worth to explain how the CPS, VPS, and stochastic models can be used to dynamically set T_i given the packet sampling probability p and the target accuracy SNR_{th}. Such a performance bound can be typically set larger than 10, allowing a higher resolution in real-time monitoring of bitrate than the one obtained with the current traffic monitoring systems.

[‖] Obviously, $0 \leq p \leq 1$.

In particular, the k-th exporting period counter $T_i(k)$ for the i-th flow can be obtained as follows:

$$T_i(k) = \left[\frac{1-p}{p} \times SNR_{th} - 1\right] \times \frac{0.89}{C_i}, \quad (4)$$

$$T_i(k) = \left[\frac{1-p}{p} \times SNR_{th} - 1\right] \times \frac{0.89 \times M_i}{C_i \times \overline{D}_i{}^2}, \quad (5)$$

$$T_i(k) = \frac{1-p}{p} \times SNR_{th} \times \frac{M_i}{C_i \times \overline{D}_i{}^2}. \quad (6)$$

where Eq. (4) is obtained from the CPS model, Eq. (5) from the VPS model, and Eq. (6) from the stochastic one.

Parameters $\overline{D}_i$, M_i, and C_i are calculated considering the past history related to the i-th flow, using an exponential weighted moving average filter.

4.3. LEMON Algorithm

It is possible to summarize LEMON algorithm with the three main processing operations (see Fig. 2).

a) Initializing working parameters: in this phase the sampling probability rate p, the SNR threshold SNR_{th}, and an initial default value of the observation time window T_d, are set. Flow keys are also defined. Moreover, the reference model to adopt (whether CPS, VPS, or stochastic) is chosen. The flow table is cleared.

b) Management of identified monitored flows: the flow record table is used to maintain flow records. Thus, every time a packet is captured and identified as belonging to a flow, the relative flow counters are updated.

c) Data exporting and resolution update: finally, at the expiration of the i-th per-flow timeout, which is set equal to the duration of the flow time bin $T_i(k)$, LEMON inserts the recorded information about the expired flow into an IPFIX message. Then, the size of the next time bin $T_i(k+1)$ (hence, the time resolution and the expiration time of the timeout) is set accordingly to the reference model.

In particular, LEMON performs these operations, as follows:

1. exportation of a data record containing the flow key (or a correspondent flow ID associated to the flow), the $T_i(k)$ value as well as the number of bytes and packets captured during the last bin at the monitored interface;
2. updating of estimates for parameters $\overline{D}_i$, M_i, and C_i, taking into account sampled packets of the i-th flow;
3. setting of the next value of the *flowBinTimeout* $T_i(k+1)$, according to Eqs. (4), (5) or (6);
4. resetting of per-flow counters;

Trans. Emerging Tel. Tech. 2012; **00**:1–17 © 2012 John Wiley & Sons, Ltd.
DOI: 10.1002/ett
Prepared using ettauth.cls

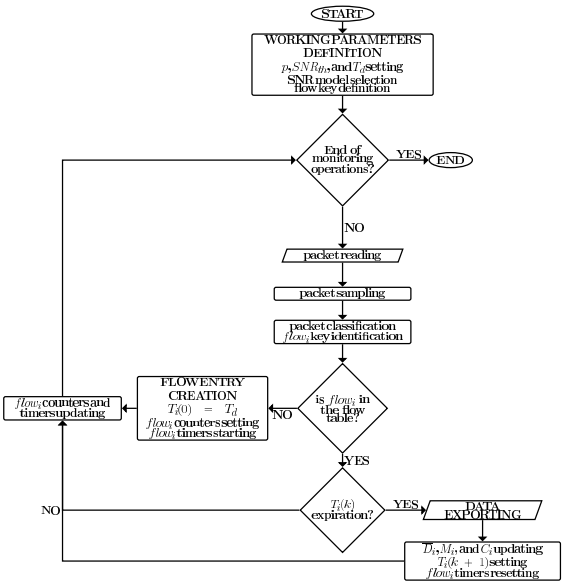


Figure 2. LEMON main processing operations.

5. restarting of the exporting timer relative to the i-th flow counter.

It is worth noticing that, as required by the standard, an IPFIX message could include information (data sets) about one or more flows. In Sec. 5.4, this issue will be explored by evaluating the monitoring overhead when several encapsulation strategies are used to build IPFIX messages.

5. EXPERIMENTAL RESULTS

The aim of this Section is to characterize the performance of LEMON from many points of view. In particular, we argue the effectiveness of LEMON by providing experimental evidences obtained during the operating phase. Mainly, the attention will be focused on the following aspects: (i) a comparison with respect to the state of the art approach, based on a constant value of time bin T; (ii) the analysis of the accuracy of the bitrate estimation that LEMON can grant for; (iii) a discussion about the implications deriving from a time-varying T on bitrate tracking operations; (iv) the evaluation of the communication overhead due to message exporting; (v) the estimation of the computational requirements of LEMON (both memory and CPU loads will be considered) and (vi) a comparison of obtained results with respect to the capabilities of current networking equipments.

All those investigations have been carried out using two kinds of real traffic traces; the first traces have been captured on Asia transpacific links (thanks to the MAWI project) [24], whereas the other one have been obtained on an access router attached to a DSLAM of an Internet provider with about a thousand of customers behind.

Table I. Main traffic parameters of aggregate traces

	Link Capacity [Mbps]	Link Usage [%]	$\overline{D}$ [Byte]	M [Byte^2]	# flows
Trace1 (MAWI) Jan.2009	150	87	748	1014959	153
Trace2 (MAWI) Jan.2009	150	13	341	400628	212
Trace3 (MAWI) Dec.2005	150	34	621	829281	151

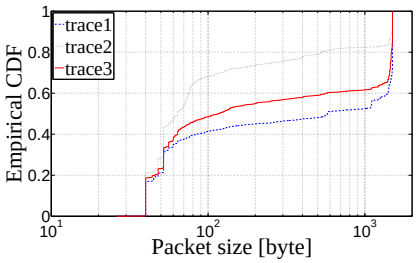


Figure 3. Empirical Cumulative Distribution Function of the packet size, for traces 1, 2, and 3.

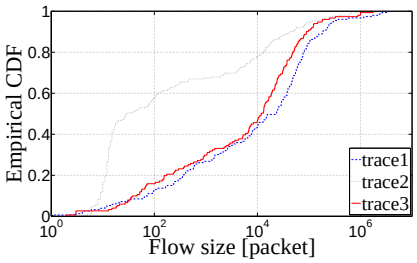


Figure 4. Empirical Cumulative Distribution Function of the flows size, for traces 1, 2, and 3.

Three distinct MAWI traces (each one is 15 minutes long) have been taken at two trans-pacific links during December 2005 and January 2009**. They include IP packets with TCP, UDP, and ICMP segments. The main characteristics of these traffic traces are summarized in Tab. I.

To provide a further insight into the characteristics of the three MAWI traces, Figs. 3 and 4 show the empirical Cumulative Distribution Function (ECDF) of packet and flow sizes, demonstrating that the three traces present very different behaviors, thus they consist of a useful benchmark for the LEMON algorithm.

** The traces are available at http://mawi.wide.ad.jp/mawi/samplepoint-F/2009/ and http://tracer.cls.sony.co.jp/mawi/samplepoint-B/2006/.

Analogous information about the trace collected at the DSLAM (that accounts for a period of three hours) cannot be provided due to a non-disclosure agreement with the Internet provider. Anyway, the experiments conducted on this trace are fully in agreement with those obtained with the MAWI traces, so that, the corresponding results will be analyzed only with reference to the computational complexity of LEMON in Sec. 5.5.

With reference to the experimental methodology, each trace is sampled with probabilities ranging in the interval $[10^{-4}, 8 \times 10^{-1}]$. For each sampling probability, ten distinct experiments are carried out applying different seeds for the random number generator that drives packet sampling. Furthermore, four distinct SNR_{th} values are considered as performance requirement: 10, 15, 20, and 50[††].

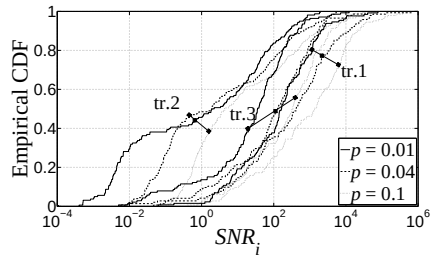
In LEMON, the number of flows that can be processed simultaneously depends on the link capacity and on the memory available at the line card. This aspect will be thoroughly analyzed in Sec. 5.5. Furthermore, LEMON works on a per-flow basis, so that the treatment applied to a flow is not dependent (to a large extent) on the number of monitored flows, as long as the average bitrate of the flow is large enough. Therefore, in what follows, unless otherwise specified and without less of generality, the term *flow* is referred to a collection of IP datagrams having the same most significant byte of the sending IP address.

Given the sampling probability p and the SNR_{th} constraints, we only track flows for which LEMON estimates an optimal time bin smaller than 15 minutes. Remaining flows, composed by a small amount of packets, can be handled using other measurement techniques based on a fixed coarse time resolution.

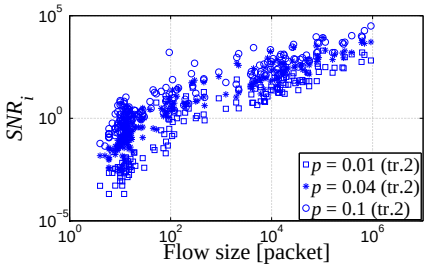
5.1. Limits of fixed resolution monitoring

To provide a ground for comparison between a fixed time scale monitoring (i.e., a flow-based NetFlow-like monitoring system) and LEMON, we firstly present results obtained with a fixed time bin equal to 240 s (see Fig. 5). Fig. 5(a) pictures the empirical CDF of the measured SNR for each analyzed flow, monitored with different sampling probabilities. It is worth to note that using fixed time bins, it is not possible to provide any guarantee on the SNR, especially in the presence of "small" flows, where small has to be intended in terms of number of packets. For further confirmation, Fig. 5(b) reports the mean measured SNR value referred to trace 2, which has the higher quota of small flows (as shown in Fig. 4). This problem, as we will see in what follows, can be greatly mitigated by LEMON, because it improves the resolution when possible, leaving very small flows to estimation methods based on fixed and large observation windows (even wider than 240 s).

[††] Due to the lack of space, only results obtained for SNR_{th}=10 and SNR_{th}=50 will be reported.



(a) Empirical CDF of the mean measured SNR.



(b) Mean measured SNR value (for trace 2).

Figure 5. Monitoring approach with a fixed time bin (240 s).

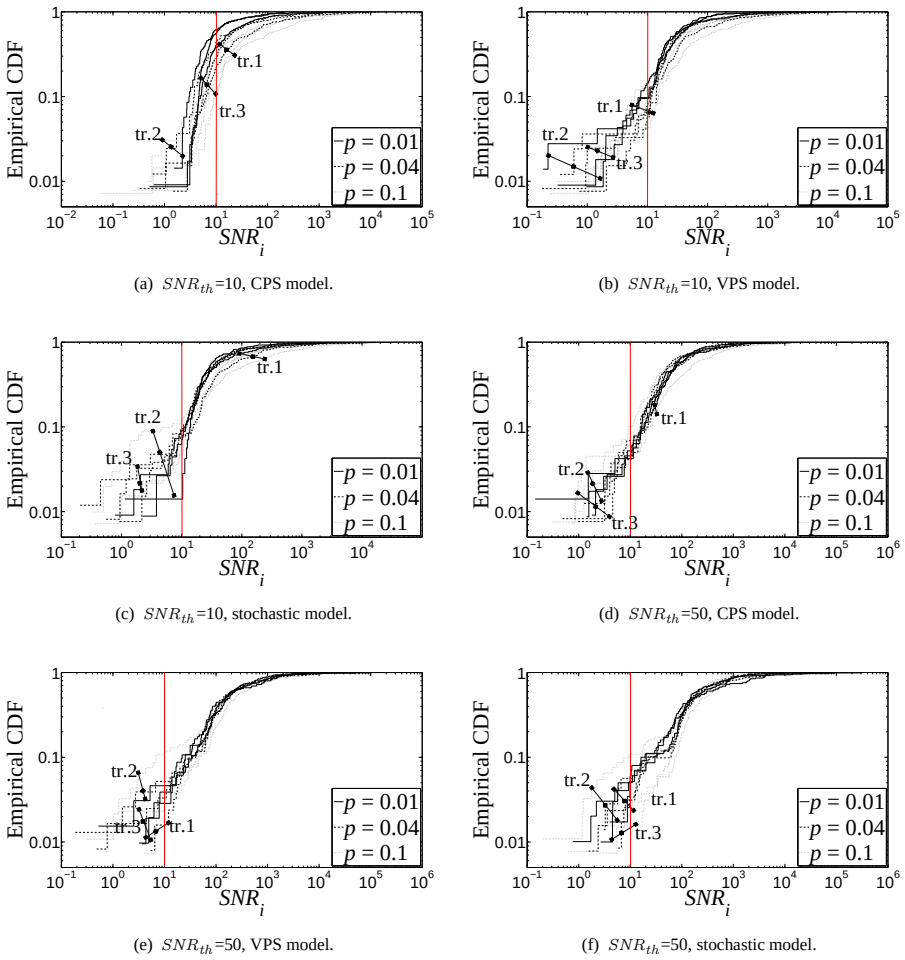
5.2. LEMON estimation accuracy

We look now at the estimation accuracy that LEMON can grant. This investigation is very relevant, given that models in [10] have been developed with the implicit assumption that the time bin T_i is constant. As a consequence, it is necessary to test if LEMON is able to respect the constraint on the SNR_{th} parameter when T_i varies with the time. In Fig. 6, the empirical CDFs of the mean measured SNR values are reported for all processed flows; they are compared with respect to the threshold SNR_{th}, considering several sampling probabilities. In particular, it is possible to note that, for all the three traces, the VPS and stochastic models provide higher performance than the one obtained with the CPS model.

This effect can be explained by noting that the CPS model does not account for the packet size variability, so that the accuracy it provides is smaller than with other two models we considered. Moreover, we can see that the trace 2 presents the highest percentages of flows for which LEMON provides a SNR below the target threshold. In fact, the trace 2 (see also Fig. 4) is characterized by the highest number of small flows (measured in terms of number of packets). For such flows, the number of sampled packets is not enough to reach a satisfying estimation of C_i, $\overline{D}_i$ and M_i parameters; this compromises the effectiveness of LEMON.

In general, observing plots in Fig. 6, it is possible to conclude that the latter two models are able to provide the expected measurement accuracy to about 90% of the processed flows, thus confirming the validity of the LEMON rationale. As further consideration, it is very

Trans. Emerging Tel. Tech. 2012; **00**:1–17
DOI: 10.1002/ett
Prepared using ettauth.cls



(a) SNR_{th}=10, CPS model.

(b) SNR_{th}=10, VPS model.

(c) SNR_{th}=10, stochastic model.

(d) SNR_{th}=50, CPS model.

(e) SNR_{th}=50, VPS model.

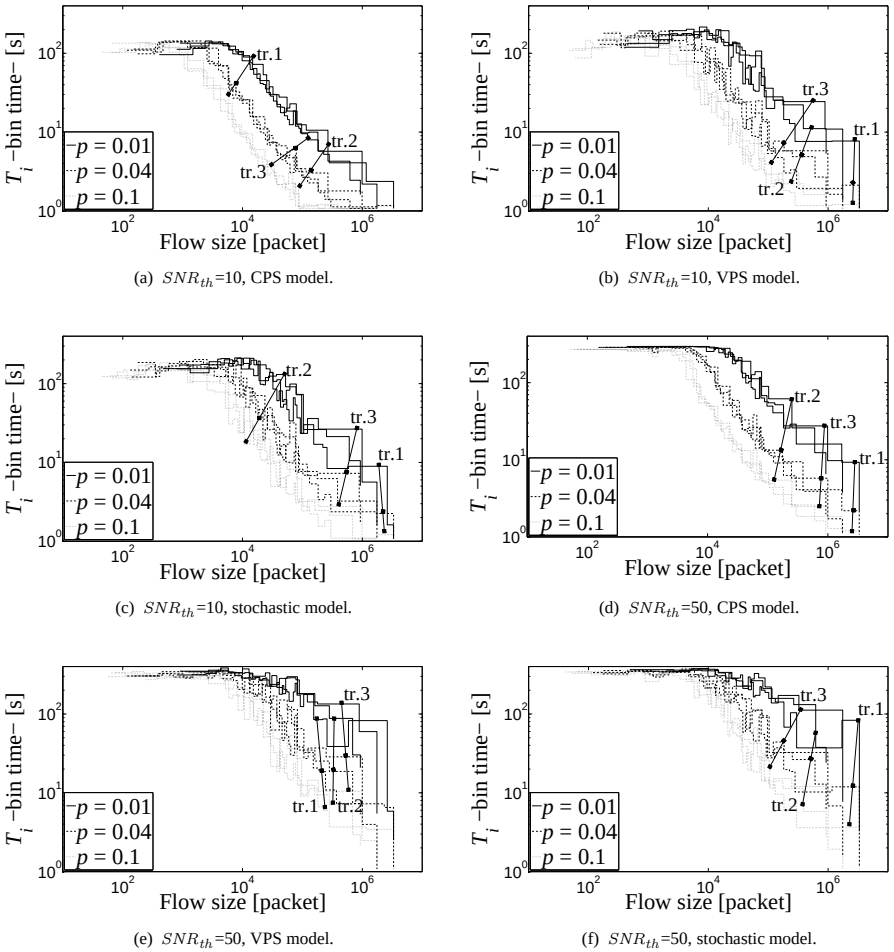
(f) SNR_{th}=50, stochastic model.

Figure 6. Empirical CDFs of the mean measured SNR. Vertical lines represent the SNR threshold constraints.

worth to notice how the SNR curves provided by LEMON are much less spread that those obtained using a fixed time bin, as in Fig. 5(b). This relevant outcome is highly beneficial because it demonstrates that LEMON is able to meet the expected constraints on the SNR, pursuing this objective in a very fair way with respect to the different flows it handles.

5.3. Time resolution and bitrate tracking

Herein, the level of time resolution reached by LEMON and its ability to track the bitrate of monitored flows are evaluated. First of all, it is interesting to dwell on the analysis of the mean time bin assigned to each flow as a function of the flow size, for all the experimental traces (see Fig. 7). For sake of clarity, all monitored flows are grouped considering their size and the average resolution (i.e., the mean time bin) of each group made by four flows is shown. Obviously, LEMON assigns smaller values of the time bin to larger flows. In fact, only in the presence of a reasonable number of packets composing a flow, it is possible to reach a finer time resolution, as predicted by Eqs. (4)-(6). Looking at the reported values, we can see that, opposite to current traffic monitoring tools, LEMON can estimate the bitrate at small time bins leading to a higher measurement resolution, i.e., one can obtain traffic updates for some flows every few seconds. This higher resolution allows operators to closely follow the traffic behavior and to take the appropriate management decisions on time. It is also worth to note that operators can always increase SNR_{th}(i.e., the target bitrate accuracy) and automatically perform measurements at a coarser time resolution. In fact, according to previous results, focusing in particular on flows larger than 10^5 packets, the level of time resolution, reached considering the lowest value of SNR threshold (i.e., SNR_{th}=10), is improved of about one order of magnitude with respect to the highest one (i.e., SNR_{th}=50). This means that LEMON is a highly flexible tool that allows an operator to trade estimation accuracy for time resolution, depending on the monitoring application.



(a) SNR_{th}=10, CPS model.

(b) SNR_{th}=10, VPS model.

(c) SNR_{th}=10, stochastic model.

(d) SNR_{th}=50, CPS model.

(e) SNR_{th}=50, VPS model.

(f) SNR_{th}=50, stochastic model.

Figure 7. Mean temporal resolution (mean bin size).

Graphs in Figs. 8-10 show the time evolution of the time bin of three selected flows from Traces 1, 2, and 3 respectively. In all these cases, we notice that the time bin settles very soon to a constant value, which is a nice property if one wants to grant that each flow is monitored on a regular basis. Obviously, it is worth to remark that the convergence of the time bin to a constant value can be achieved only for flows with stationary properties in terms of the parameters C_i, $\overline{D}_i$, and M_i, used in Eqs. 4-6.

To provide a further insight, Figs. 11-13 show the capacity of the proposed scheme to track the time bin over the time for each flow. In particular, we can observe how, fixing the SNR threshold parameter, the time bin is dynamically adapted over time, providing a finer time resolution with higher sampling probabilities. It is interesting to note that these plots fully confirm the foregoing analysis since the bitrate estimation accuracy is strictly related to the flow size and to the sampled packets captured at router interface.

5.4. Overhead of the exporting process

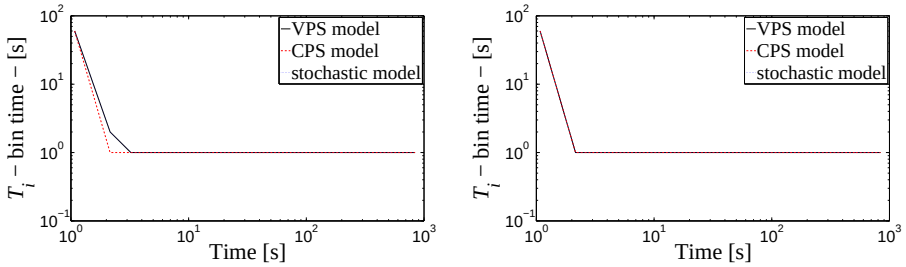
Unlike fixed-resolution monitoring systems, LEMON might export collected statistics with a higher frequency due to a time-varying bin. Hence, it is very important to analyze the behavior of its communication protocol in terms of the overhead that it creates.

We consider three different message exporting strategies, which will be referred to as $policy = 0$, $policy = 1$, and $policy = 2$. In particular, they are defined as follows:

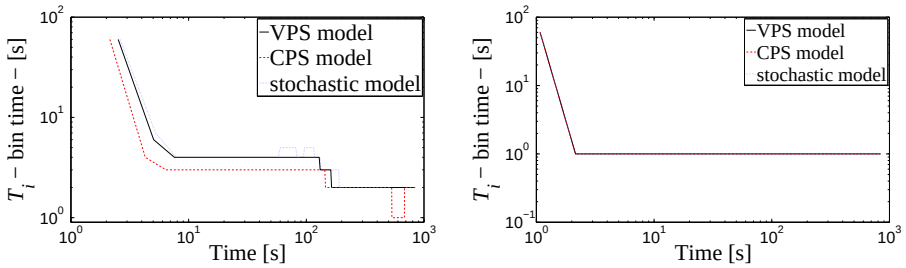
- in $policy = 0$, a single IPFIX message is sent every time the *flowBinTimeout* expires for a single flow.
- In $policy = 1$, an aggregate IPFIX message is sent at the expiration of 10 different *flowBinTimeout*.
- In $policy = 2$, an aggregate IPFIX message is sent at the end of a timeout lasting 5 s, for each expired *flowBinTimeout* (the value of this timeout can be set by the user, based on its needs).

In our analysis, we do not consider the headers of the transport and the network protocols, which are not due

Trans. Emerging Tel. Tech. 2012; **00**:1–17 © 2012 John Wiley & Sons, Ltd.
DOI: 10.1002/ett
Prepared using ettauth.cls

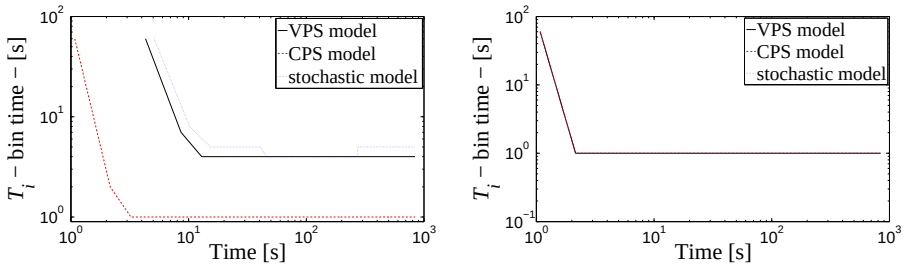


(a) SNR_{th}=10, p=0.01. (b) SNR_{th}=10, p=0.1.

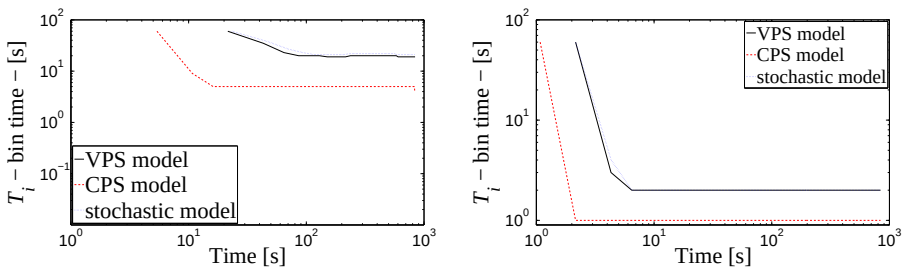


(c) SNR_{th}=50, p=0.01. (d) SNR_{th}=50, p=0.1.

Figure 8. Time evolution of the bin size flow #9, for trace 1.



(a) SNR_{th}=10, p=0.01. (b) SNR_{th}=10, p=0.1.

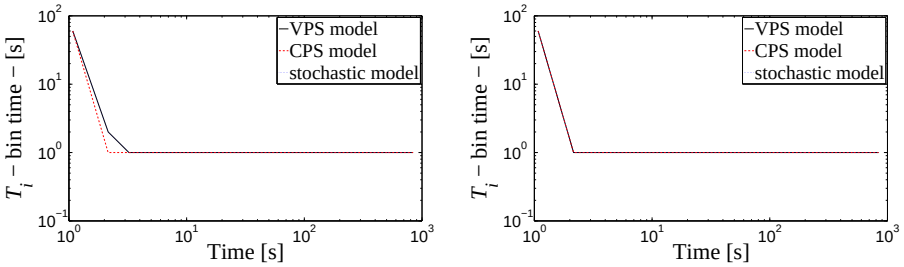


(c) SNR_{th}=50, p=0.01. (d) SNR_{th}=50, p=0.1.

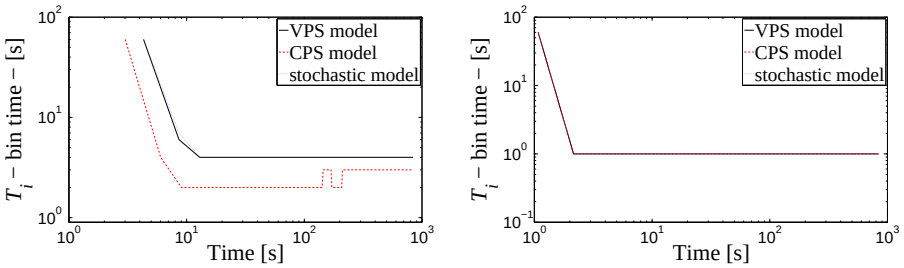
Figure 9. Time evolution of the bin size of flow #4, for trace 2.

to LEMON, but we consider the communication overhead due to both the flow records attributes (*information element* data records), and the control messages (*control information* records). Such control messages contain information related to the metering process, e.g., the sampling probability value p, the active or idle per-flow timeouts, and so on.

The communication overhead is evaluated in terms of bitrate related to the flow of IPFIX messages sent to the collector, at different operating conditions; it is expressed as a percentage of the capacity of the monitored link.

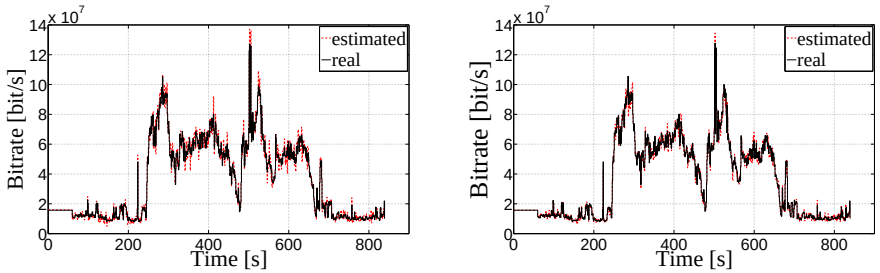


(a) SNR_{th}=10, p=0.01. (b) SNR_{th}=10, p=0.1.

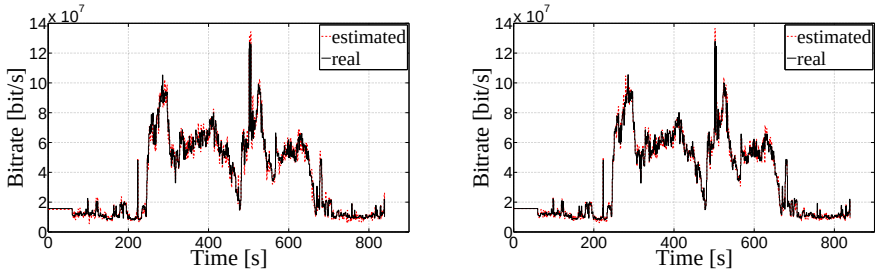


(c) SNR_{th}=50, p=0.01. (d) SNR_{th}=50, p=0.1.

Figure 10. Time evolution of the bin size flow #9, for trace 3.



(a) SNR_{th}=10, p=0.04. (b) SNR_{th}=10, p=0.08.

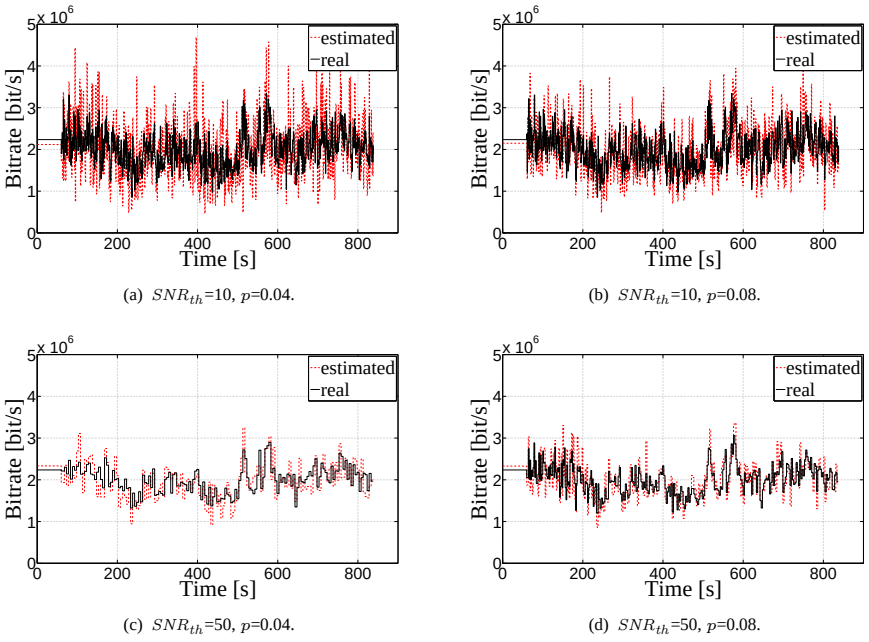


(c) SNR_{th}=50, p=0.04. (d) SNR_{th}=50, p=0.08.

Figure 11. Bitrate of flow #9, for trace 1 (VPS Model).

In Tab. II, we report results related only to the VPS model, but similar outcomes have been obtained also for the CPS and stochastic models. As expected, Tab. II shows that increasing the sampling probability p, the $policy = 0$ has a greater overhead with respect to the other ones, because with such a policy one message is transmitted every time the exporting timer expires. Instead, the other two approaches (i.e., $policy = 1$ and $policy = 2$) can greatly save communication resources, thus reaching a better tradeoff between channel overhead and frequency of exporting operations. In any case, the overall communication overhead is quite limited if we

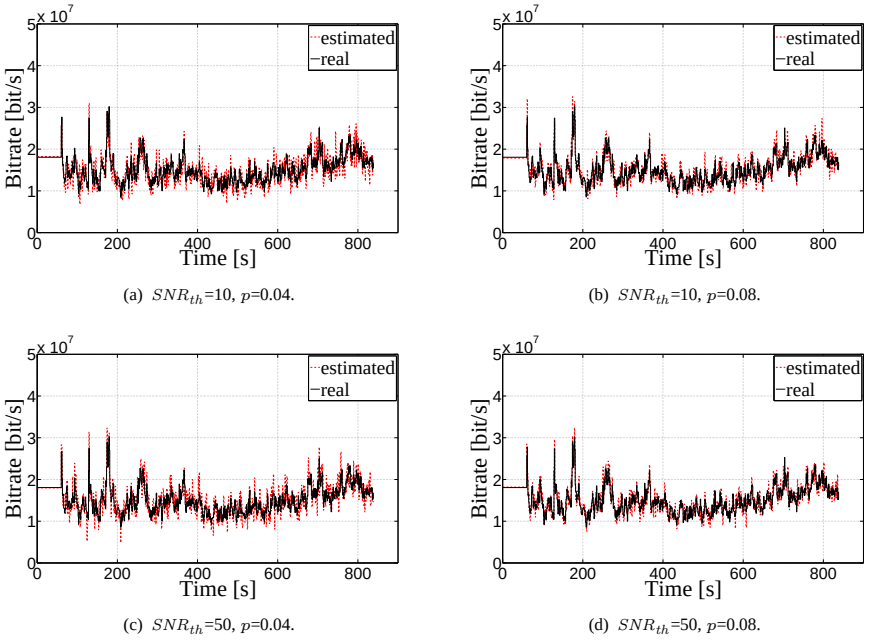
Trans. Emerging Tel. Tech. 2012; **00**:1–17
DOI: 10.1002/ett
Prepared using ettauth.cls



(a) SNR_{th}=10, p=0.04. (b) SNR_{th}=10, p=0.08.

(c) SNR_{th}=50, p=0.04. (d) SNR_{th}=50, p=0.08.

Figure 12. Bitrate of flow #4, for trace 2 (VPS Model).



(a) SNR_{th}=10, p=0.04. (b) SNR_{th}=10, p=0.08.

(c) SNR_{th}=50, p=0.04. (d) SNR_{th}=50, p=0.08.

Figure 13. Bitrate of flow #9, for trace 3 (VPS Model).

compare our results with existing technologies. In Cisco NetFlow, in fact, the total amount of exported data is estimated like about the 1.5% of the switched traffic into the routers [47], considering the version without sampling. This overhead is always larger than the communication overead LEMON reaches using $policy = 1$ and $policy = 2$.

	$trace1$			$trace2$			$trace3$		
	$p = 0.01$	$p = 0.1$	$p = 0.8$	$p = 0.01$	$p = 0.1$	$p = 0.8$	$p = 0.01$	$p = 0.1$	$p = 0.8$
SNR_{th}=10									
$policy = 0$	0.3%	1.3%	3.87%	0.08%	0.58%	2.35%	0.17%	0.8%	3.68%
$policy = 1$	0.1%	0.5%	1.47%	0.03%	0.22%	0.89%	0.06%	0.3%	1.4%
$policy = 2$	0.1%	0.46%	1.3%	0.03%	0.20%	0.8%	0.06%	0.28%	1.26%
SNR_{th}=50									
$policy = 0$	0.08%	0.48%	2.9%	0.02%	0.16%	1.5%	0.04%	0.28%	2.4%
$policy = 1$	0.03%	0.19%	1.1%	$\simeq 0\%$	0.06%	0.6%	0.02%	0.11%	0.9%
$policy = 2$	0.03%	0.17%	1%	$\simeq 0\%$	0.06%	0.51%	0.02%	0.1%	0.81%

Table II. Amount of exported messages with LEMON (percentage over the link capacity of test, i.e., $150Mbps$).

5.5. Computational requirements

Now, we evaluate the processing requirements of LEMON system. To this end, we use a packet trace, lasting 3 hours, collected at an access router attached to a DSLAM with about a thousand of customers behind it. In particular, we analyze the computational and memory consumption costs of LEMON under different operating conditions and when it runs on a laptop with an Intel Core 2 Duo P7450 (2.13 GHz, 3 MBytes L2 cache, 800 MHz DDR2), with 6 GBytes of RAM.

As explained before, LEMON is in charge of performing all the operations required by a monitoring device: the capture of the packets at the network interfaces, the classification of such captured packets into flows, the management of tracked flows in memory, and, finally, the exportation of measurements. These tasks are carried out in a dynamic way, varying in real-time the monitoring observation window of each flow. To provide a comprehensive view on the complexity of LEMON, we test it under different conditions, by varying both the packet sampling probability p and the flow key, which is encoded using the key_{mode} variable. The former has a direct influence on the number of packets to process, whereas the latter impacts the number of flows to handle and the granularity of measurements.

In the sequel, when $key_{\text{mode}} = 0$, the flow is defined as the collection of all the packets having the same most significant byte of their sending IP address. Instead, when $key_{\text{mode}} = 1$, the flow is defined as the collection of all the packets having the same sending IP address and the same transport protocol number. The purpose of this choice is to provide a comparison of the system requirements under two different working scenarios, the latter more challenging than the former.

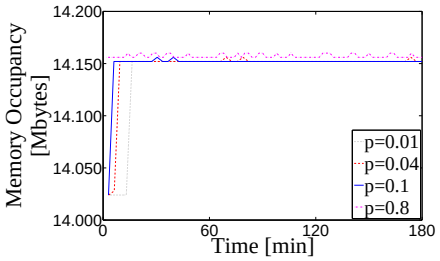
The memory consumption of LEMON along the time due to the monitoring operation is shown in Fig. 14, considering different operating conditions. As we can see in the graphs in Fig. 14.(a), LEMON exhibits a constant level of memory consumption for coarser flows (i.e., when $key_{\text{mode}} = 0$), almost independent on the packet sampling probability. This is a positive achievement; in fact, we can infer that, with aggregate flows captured for a large range of sampling rates, the memory consumption of LEMON is quite independent of the sampling rate. Thus, it is possible to run the tool under strenuous working conditions. Differently, the case of Fig. 14(b) shows a strong correlation between the memory consumption and the sampling probability. This is due to the definition of the flow key, which now allows a more granular analysis of flows, hence decreasing their sizes and making their detection by LEMON strongly dependent on the sampling rate. This fine-grained definition of a flow increases the size of the flow record table and, as a consequence, the memory usage. This effect is confirmed by Fig. 15, which reports that the number of detected flow records (each one having a size of 256 bytes) has exactly the same trend of the memory consumption. This means that the flow record table represents the main cause of the memory consumption in LEMON. At the same time, we remark that, in this test scenario, such a consumption never exceeds few tens of megabytes even when a high value of p (i.e., $p = 0.8$) is adopted. In order to compare this result with nowadays available technologies, we have to consider that, in Cisco NetFlow, each flow entry requires about 64 bytes of memory and that flow table can store up to 512 thousands of entries. Moreover, it should be noticed that the amount of memory on a line card sets an upper bound on the number of flows that can be handled. Starting from these premises, to evaluate the impact that LEMON may have in a real network, we consider (as an example) the engine 3 line card (256 Mbytes of memory and 16 network interfaces) embedded in Cisco 12000 routers [47]. Under these conditions, the Cisco 12000 router (when LEMON is on) is still able to handle 62500 flow entries in each table at each network interface, which is a value much higher than the number of flows observed in all considered traces of this paper. On the other hand, basic Netflow can process 4 times more flows due to smaller flow entries in the flow table (see also Tab. III).

Finally, we provide a preliminary analysis of LEMON requirements in terms of CPU usage by using an absolute index of the CPU performance, i.e. the CPU cycle number. In order to keep track of this parameter, we used the *ReaD Time Stamp Counter* (RDTSC) CPU instruction.

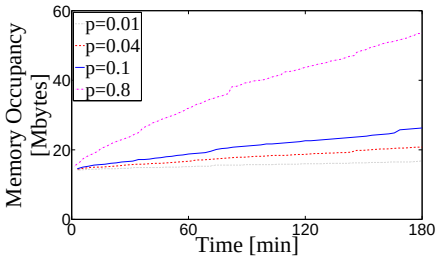
Trans. Emerging Tel. Tech. 2012; **00**:1–17 © 2012 John Wiley & Sons, Ltd.
DOI: 10.1002/ett
Prepared using ettauth.cls

	$CiscoNetFlow$	$LEMON$
Flow entry size	64 bytes	256 bytes
Memory consumption on the Cisco 12000 engine 3 line card (256Mbytes)	256M/16/64 = 256k entries	256M/16/256 = 62.5k entries

Table III. LEMON vs Cisco NetFlow: memory consumption comparison.

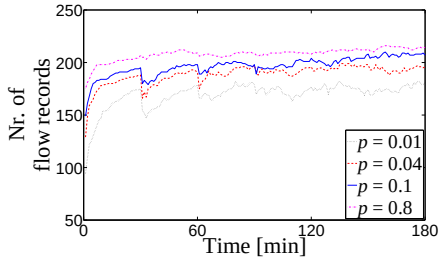


(a) Flow key definition: $key_{\text{mode}} = 0$.

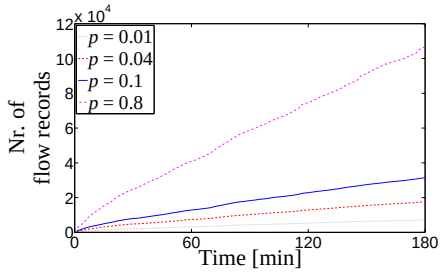


(b) Flow key definition: $key_{\text{mode}} = 1$.

Figure 14. Memory consumption of the whole LEMON process.



(a) Flow key definition: $key_{\text{mode}} = 0$.

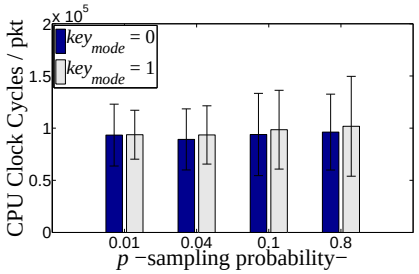


(b) Flow key definition: $key_{\text{mode}} = 1$.

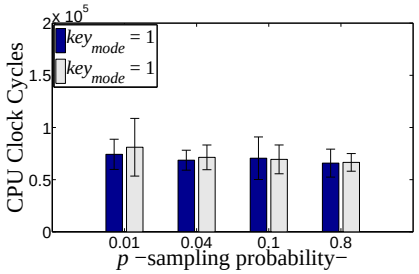
Figure 15. Number of flow records into the flow table during traffic monitoring operations.

This function, in fact, keeps an accurate count of cycles spent at the processor for each routine or section of code and it provides results that are general and absolute. During each experiment, we recorded the number of processing cycles required to execute the routines of LEMON. First, we analyze the execution effort of the whole LEMON routine for a single captured packet (i.e., packet classification, flow identification, management and updating of the corresponding flow record, timer expiration and exporting process when the last two happen). Then we evaluate the time bin expiration subroutine, that occurs for a particular flow when the expiration of its relative timer is detected; it includes the calculation of the optimal next time bin for a particular flow (as stated by the closed-form models) and the exporting of information collected at the previous time bin.

Fig. 16 shows the mean number of CPU clock cycles required to perform the two cited processes, with different operating conditions; figures report also the standard deviation interval (above and below the mean value) for each measure in order to give an estimation of the confidence of the obtained values. As we can see, the time bin expiration routine, which is a subroutine of the whole LEMON process, constitutes the highest quota of the total LEMON cost. All results exhibit mostly the same behavior, regardless of the sampling probability and the key definition, because metrics are measured on a per-packet basis. We should then use these per-packet metrics to estimate the whole processing costs in the time domain. For example, considering a realistic scenario with a Cisco 12000 router endowed with a main processor of 667 MHz [48], the per-packet processing time is 150 μs, estimated as the ratio between the number of required clock cycles for each packet and the CPU clock. This leads to the number of packets that are processed per second: $\frac{1}{150\times10^{-6}} = 6670$ packet/s, which, assuming a sampling probability of $p = 0.01$ with a mean packet dimension of 800 bytes, allows LEMON to monitor in real-time a network link of about $6670 \times 800 \times 8/0.01 = 4.27$ Gbps. Granted that the actual release of the project is still in a embryonic stage and it is not still optimized to be embedded in a real machine, in our humble opinion we think that these are preliminary and cheering results, justifying the high versatility of LEMON and its ability to be implemented in real network devices.



(a) Cost of LEMON routine per packet.



(b) Cost Time bin expiration subroutine.

Figure 16. Processing overhead of LEMON routines.

6. CONCLUSIONS

Starting from recent theoretical findings that catch the main properties of packet sampling and its implications on bitrate estimation, this work proposes a novel approach to flow-level link monitoring operations. The conceived LEMON algorithm has been designed to improve the accuracy of current NetFlow-like monitoring systems that propose flow-level measurements at fixed resolution. Its effectiveness has been evaluated using real packet traces, captured at backbone router interfaces. Results demonstrate that LEMON: (i) allows adaptive monitoring windows, hence the possibility to monitor the traffic with different granularity levels; (ii) is able to assure the estimation accuracy requirements, thanks to the obtained bitrate estimates; (iii) incurs a negligible communication overhead in IPFIX message exporting operations; (iv) has low impact on computational requirements, so that can it be easily integrated and supported by current deployed routers.

It is worth to note that the scope of the LEMON algorithm is not only limited to the real-time adaptation of the time resolution. Future achievements of our research, in fact, will cover different aspects as for example the joint tuning of the sampling probability with the time resolution, the integration within anomaly detection frameworks, the evaluation of different traffic granularities, the implementations in open source NetFlow-like monitoring tools, and the study of further scenarios and applications.

REFERENCES

1. Claffy KC, Polyzos GC, Braun KW. Application of sampling methodologies to network traffic characterization. *ACM SIGCOMM Computer Communication Review*, 1993; **23**(4).
2. Grieco LA, Barakat C. An analysis of packet sampling in the frequency domain. *Proc. of ACM SIGCOMM IMC*, 2009.
3. Cantieni GR, Iannaccone G, Barakat C, Diot C, Thiran P. Reformulating the monitor placement problem: Optimal network-wide sampling. *Proc. of ACM CoNeXT*, 2006.
4. Brauckoff D, Tellenbach B, Wagner A, May M, Lakhina A. Impact of packet sampling on anomaly detection metrics. *Proc. of ACM SIGCOMM IMC*, 2006.
5. B Claise E. *Cisco Systems NetFlow Services Export Version 9*. IETF, RFC 3954, Oct 2004.
6. Quittek J, Zseby T, Claise B, Zander S. *Requirements for IP Flow Information Export (IPFIX)*. IETF, RFC 3917, Oct 2004.
7. Claise B. *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information*. IETF, RFC 5101, Jan 2008.
8. Estan C, Keys K, Moore D, Varghese G. Building a better netflow. *Proc. of ACM SIGCOMM*, 2004.
9. Uyeda F, Foschini L, Baker F, Suri S, Varghese G. Efficiently measuring bandwidth at all time scales. *Proc. of USENIX NSDI*, 2011.
10. Grieco LA, Barakat C, Marzulli M. Spectral models for bitrate measurement from packet sampled traffic. *IEEE Transactions on Network and Service Management*, 2011; **8**(2).
11. Grieco LA, Barakat C. A frequency domain model to predict the estimation accuracy of packet sampling. *Proc. of IEEE INFOCOM*, 2010.
12. Sridharan A, Guérin R, Diot C. Achieving near-optimal traffic engineering solutions for current ospf/is-is networks. *IEEE/ACM Transactions on Networking*, 2005; **13**(2).
13. Awduche D. Mpls and traffic engineering in ip networks. *IEEE Communications Magazine*, 1999; **37**(12).
14. McKeown N, Anderson T, Balakrishnan H, Parulkar G, Peterson L, Rexford J, Shenker S, Turner J. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 2008; **38**(2).
15. Sherwood R, Chan M, Covington A, Gibb G, Flajslik M, Handigol N, Huang TY, Kazemian P, Kobayashi M, Naous J, *et al.*. Carving research slices out of your production networks with openflow. *ACM SIGCOMM Computer Communication Review*, 2010; **40**(1).
16. Brauckhoff D, Tellenbach B, Wagner A, May M, Lakhina A. Impact of packet sampling on anomaly

DOI: 10.1002/ett
Prepared using ettauth.cls

detection metrics. *Proc. of ACM SIGCOMM IMC*, 2006.
17. Mai J, Chuah CN, Sridharan A, Ye T, Zang H. Is sampled data sufficient for anomaly detection? *Proc. of ACM SIGCOMM IMC*, 2006.
18. Casas P, Fillatre L, Vaton S, Nikiforov I. Traffic management and traffic engineering for the future internet. chap. Volume Anomaly Detection in Data Networks: An Optimal Detection Algorithm vs. the PCA Approach, Springer-Verlag, 2009.
19. Schneider F, Wallerich J, Feldmann A. Packet capture in 10-gigabit ethernet environments using contemporary commodity hardware. *Proc. of PAM*, 2007.
20. Lakhina A, Crovella M, Diot C. Detecting distributed attacks using network-wide flow traffic. *Proc. of FloCon*, 2005.
21. Schonwalder J, Pras A, Martin-Flatin JP. On the future of internet management technologies. *IEEE Commun. Mag.* Oct 2003; **41**(10):90 – 97, doi:10.1109/MCOM.2003.1235600.
22. Case J, Fedor M, Schoffstall M, Davin J. *A Simple Network Management Protocol (SNMP)*. IETF, RFC 1157, May 1990.
23. Vilardi R, Grieco LA, Boggia G, Barakat C. Temporal resolution adaptation in real-time bitrate estimation from packed sampled traffic. *Proc. of IEEE IWCMC*, 2011.
24. Mawi Working Group Traffic Archive. `http://mawi.wide.ad.jp/mawi/`.
25. Silveira F, Diot C, Taft N, Govindan R. Astute: detecting a different class of traffic anomalies. *Proc. of ACM SIGCOMM*, 2010.
26. Kanuparthy P, Dovrolis C, Ammar M. Spectral probing, crosstalk and frequency multiplexing in internet paths. *Proc. of ACM SIGCOMM IMC*, 2008.
27. Katti S, Katabi D, Blake C, Kohler E, Strauss J. Multiq: Automated detection of multiple bottlenecks along a path. *Proc. of ACM SIGCOMM IMC*, 2004.
28. Ameigeiras P, Ramos-Munoz JJ, Navarro-Ortiz J, Lopez-Soler J. Analysis and modelling of youtube traffic. *Transactions on Emerging Telecommunications Technologies*, 2012; **23**(4).
29. Claffy KC, Polyzos G, Braun HW. Traffic characteristics of the t1 nsfnet backbone. *Proc. of IEEE INFOCOM*, 1993.
30. Barford P, Kline J, Plonka D, Ron A. A signal analysis of network traffic anomalies. *Proc. of ACM SIGCOMM IMW*, 2002.
31. Palkopoulou E, Merkle C, Schupke DA, Gruber CG, Kirstdter A. Traffic models for future backbone networks a service-oriented approach. *European Transactions on Telecommunications*, 2011; **22**(4).
32. Matthews W, Cottrell L. The pinger project: active internet performance monitoring for the henp community. *IEEE Communications Magazine*, 2000; **38**(5).
33. Alves M, Corsello L, Karremberg D, Ögüt C, Santcroos M, Sojka R, Uijterwaal H, Wilhelm R. New measurements with the ripe ncc test traffic measurements setup. *Proc. of PAM*, 2002.
34. Pezaros DP, Hutchison D, Sventek JS, Garca F, Gardner RD. In-line service measurements: an ipv6-based framework for traffic evaluation and network operations. *Proc. of IEEE NOMS*, 2004; 497–510.
35. Pezaros DP, Hoerdt M, Hutchison D. Low-overhead end-to-end performance measurement for next generation networks. *IEEE Transactions on Network and Service Management*, 2011; .
36. Fraleigh C, Diot C, Lyles B, Owezarski SMP, Papagiannaki D, Tobagi F. Design and deployment of a passive monitoring infrastructure. *Proc. of PAM*, 2001.
37. Anagnostakis K, Ioannidis S, Miltchev S, Greenwald M, Smith J, Ioannidis J. Efficient packet monitoring for network management. *Proc. of IEEE/IFIP NOMS*, 2002.
38. Croce D, Mellia M, Leonardi E. The quest for bandwidth estimation techniques for large-scale distributed systems. *ACM SIGMETRICS Performance Evaluation Review* 2010; **37**(3).
39. Deri L, Suin S. Effective traffic measurement using ntop. *IEEE Communications Magazine*, 2000; **38**(5).
40. Roughan M. A case study of the accuracy of snmp measurements. *Journal of Electrical and Computer Engineering* 2010; .
41. Phaal P, Panchen S, McKee N. *InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks*. IETF, RFC 3176, Sept 2001.
42. Kumar A, Xu J. Sketch guided sampling - using on-line estimates of flow size for adaptive data collection. *Proc. of IEEE INFOCOM*, 2006.
43. Duffield N, Lund C, Thorup M. Learn more, sample less: Control of volume and variance in network measurement. *IEEE Transactions on Information Theory*, 2005; **51(5)**.
44. Lassoued I, Krifa A, Barakat C, Avrachenkov K. Network-wide monitoring through self-configuring adaptive system. *Proc. of IEEE INFOCOM*, 2011.
45. Quittek J, Bryant S, Claise B, Aitken P, Meyer J. *Information Model for IP Flow Information Export*. IETF, RFC 5102, Jan 2008.
46. Sadasivan G, Brownlee N, Claise B, Quittek J. *Architecture for IP Flow Information Export*. IETF, RFC 5470, Mar 2009.
47. Netflow services solutions guide. `http://www.cisco.com/en/US/docs/ios/solutions_docs/netflow/nfwhite.html`.
48. Cisco 12000 series performance router processor-1. `http://www.cisco.com/en/US/prod/collateral/routers/ps167/product_data_sheet0900aecd800f4147.html`.