A Multi-task Adaptive Monitoring System Combining Different Sampling Primitives

Imed Lassoued, Chadi Barakat

A Multi-task Adaptive Monitoring System Combining Different Sampling Primitives

Imed Lassoued, Chadi Barakat
Planete Project-Team, INRIA, France
{Imed.LASSOUED, Chadi.BARAKAT}@inria.fr

***Abstract*—Traffic measurement and analysis are crucial management activities for network operators. With the increase in traffic volume, operators resort to sampling primitives to reduce the measurement load. Unfortunately, existing systems use sampling primitives separately and configure them statically to achieve some performance objective. It becomes then important to design a new system that combines different existing sampling primitives together to support a large spectrum of monitoring tasks while providing the best possible accuracy by spatially correlating measurements and adapting the configuration to traffic variability. In this paper, and to prove the interest of the joint approach, we introduce an adaptive system that combines two sampling primitives, packet sampling and flow sampling, and that is able to satisfy multiple monitoring tasks. Our system consists of two main functions: (i) a global estimator that investigates measurements done by the different sampling primitives in order to deal with multiple monitoring tasks and to construct a more reliable global estimator while providing visibility over the entire network; (ii) an optimization method based on overhead prediction that allows to reconfigure monitors according to accuracy requirements and monitoring constraints. We present an exhaustive experimental methodology with different monitoring tasks in order to assess the performance of our system. Our experimentations are done on our MonLab platform that we developed for the purpose of this research.**

I. Introduction

The widespread use of the Internet infrastructure and the tremendous growth in its size have made the management and monitoring of ISP networks an indispensable function. Currently, the world of traffic measurements presents a large number of network management tasks including traffic engineering, network resource provisioning and management, accounting and anomaly detection.

In order to cope with the increasing trend in line speed that exceeds the monitoring capabilities of monitors, several proposals try to provide a widespread monitoring infrastructure that coordinates monitoring responsibilities and distributes the work between the different monitors in a way to satisfy to the best management applications requirements. The common objective of these efforts is to design a distributed monitoring system that samples traffic in a cost-effective manner by carefully placing monitors and configuring their sampling rates. However, these application-specific systems still present some drawbacks including the problem of tightly coupling target applications and sampling primitives (i.e. they focus on satisfying a specific application using one sampling primitive). For instance, the authors in [1] use the packet sampling primitive and reconfigure periodically the different sampling rates in order to calculate the traffic matrix, while the authors in [2] use the flow sampling primitive for flow counting. The main consequence of this trend is the deployment of monitoring systems using different sampling primitives for the achievement of specific monitoring tasks without thinking of a combination of these primitives for a broader usage.

We argue that it is possible to integrate the different existing monitoring tools and sampling primitives in order to support a larger spectrum of applications. In fact, the proliferation of monitoring solutions and sampling primitives motivates us to build a novel system able to achieve a myriad of concurrent monitoring tasks while offering the best possible accuracy at limited monitoring overhead.

Three main challenges arise in the development of such system:

- How to deal with multiple monitoring objectives and how to combine independent measurements collected across the network using different sampling primitives and different monitoring tools?
- How to coordinate responsibilities across the different monitors and how to share resources between the different sampling primitives in order to improve the global accuracy while respecting resource consumption constraints?
- How to adapt to variations in the monitored traffic and in network conditions?

In this paper, we introduce a novel system that is able to integrate various existing monitoring primitives in order to support multiple monitoring tasks. We explain and validate the system design for two sampling primitives, *packet sampling* and *flow sampling*, and for three monitoring tasks, *flow counting*, *flow size estimation* and *heavy-hitter detection*. Our system extends the local monitoring tools with a network-wide cognitive engine that consists of two main design modules: *(i)* a global estimator module that investigates the local measurements of the different deployed techniques in order to provide a global more accurate estimation; *(ii)* an optimization module that dynamically adjusts the different monitors and shares resources between the supported sampling primitives according to the requirements of the monitoring tasks while addressing the tradeoff between resource consumption and global measurement accuracy. This optimization is based on

This research work is funded by the European Commission through the ECODE project (INFSO-ICT-223936) of the European Seventh Framework Programme (FP7).

an overhead prediction method to track short-term and long-term changes in the traffic and on a global weighted utility function to deal with multiple monitoring tasks.

In order to evaluate the performance of our system, we introduce out an exhaustive evaluation methodology for network-wide monitoring applications to cope with the lack of a universal experimental platform for such applications. Indeed, for the purpose of this research, we have implemented MonLab [10], a real experimental platform for traffic sampling and monitoring using real traffic traces and real Netflow-like monitoring tools. Using our platform, we provide a global experimental study of the performance of the proposed system.

The rest of the paper is organized as follows. In Section II, we summarize the related work. Then, we present the system architecture in Section III. The experimental platform is presented in Section IV. Evaluation results are presented in Section V. Finally, conclusions and future work are drawn in Section VI.

II. Related work

The importance of passive traffic measurements for the understanding and diagnosis of core IP networks has led to a considerable evolution in the number and quality of monitoring tools and techniques. Recently, numerous monitoring primitives have been proposed in order to achieve a large number of network management tasks. The spectrum is broad covering among others flow sampling [3], sample and hold [4] and packet sampling [1]. Currently, NetFlow [5] is the most widely deployed measurement solution by ISPs. However, this solution still presents some shortcomings, namely the problem of configuring sampling rates according to network conditions and the requirements of monitoring applications. Another problem comes from the low values at which the sampling rate is set in practice (between 0.01 and 0.001) and this in order to cope with the increasing trend in line speed.

In order to provide a balance between scalability (respecting the resource consumption constraints) and accuracy, many works have investigated the existing sampling primitives and have used them to build network-wide monitoring systems that coordinate responsibilities between the different monitors. For example, the authors of [6] argue that performance limits can be addressed by reducing the sampling rates in the different monitors while accuracy can be improved by combining the parallel measurements of the same flow made in different monitors. The authors in [2] focus on improving the use of network resources to achieve network-wide monitoring tasks. They present a system approach that uses a hash-based flow selection to eliminate duplicate measurements in the network and a framework for distributing responsibilities across routers to satisfy network-wide monitoring objectives. The authors in [1] use packet sampling primitives and reconfigure periodically the different sampling rates in order to increase accuracy of traffic matrix estimations.

All the above solutions propose systems that use single sampling primitives to achieve specific management applications. No one of these systems is optimized to achieve a general class of monitoring tasks and to combine different sampling primitives. In order to solve these limitations, some proposals have presented simple combination of existing sampling primitives in order to achieve a larger class of tasks. For instance, the authors in [7] combine a small number of simple and generic router primitives that collect flow-level data to estimate traffic metrics, while the authors in [8] use a combination of flow sampling and sample-and-hold to provide traffic summaries and detect resource hogs. The system we propose in this paper is not only able to combine different sampling primitives, but more importantly can adapt their contribution in a way to maximize the global measurement accuracy at limited overhead. Different monitoring applications will automatically lead to different tuning of the sampling primitives.

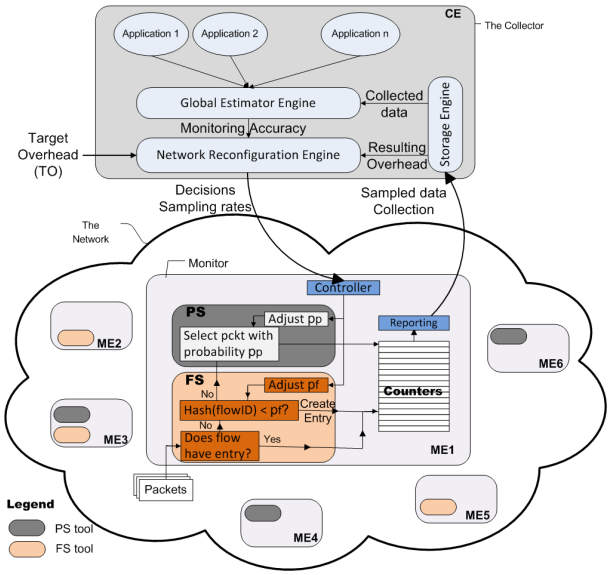


Fig. 1. System architecture.

III. System architecture

Figure 1 depicts the basic functional components of the proposed monitoring system together with the interactions among them. The system relies on existing NetFlow-like local measurement tools (Monitoring Engine (ME)) deployed in network routers. We chose to use two complementary sampling primitives: *(i)* Flow Sampling (FS) which is well suited for security and anomaly detection applications that require analyzing the flow communication structure, and *(ii)* Packet Sampling (PS) which is well suited for traffic engineering and accounting applications based on the traffic volume structure, e.g., heavy-hitter detection and traffic engineering that require an understanding of the number of packets/bytes per-port or per-prefix [7].[1] Our system extends these local

[1]While packet sampling consists in capturing a subset of packets independently of each other, flow sampling consists in capturing flows independently of each other. Once a flow is captured by flow sampling, all its packets are captured and analyzed. The decision to capture a flow or not is done at the beginning of the flow.

existing monitoring tools (MEs) with a centralized network-wide cognitive engine (CE) that drives its own deployment by automatically and periodically reconfiguring the different monitors in a way to improve the overall accuracy (according to monitoring application requirements) and reduces the resulting overhead (respecting some resource consumption constraints, typically the volume of measurements). Next we give a detailed description of the two main components of the cognitive engine of our system, which are *(i)* the global estimator engine that combines measurements and estimates network traffic, and *(ii)* the reconfiguration engine that updates the sampling rates in routers.

A. Global Estimator Engine

This component is motivated by the need to extend local existing monitoring tools (MEs) with a network-wide inference engine that combines their measurements to support a large spectrum of applications and provide more accurate results. Given a set of measurement tasks $\mathcal{T}$ to realize, this inference engine investigates the local measurements made by the different routers to obtain a global and more reliable view. Consider the set of available sampling primitives $\mathcal{S}$. For each monitoring task $T_i \in \mathcal{T}$, the global estimator takes as input the estimations of T_i, $(\hat{T}_i^s)_{s\in\mathcal{S}}$ made using the different sampling primitives in routers. The inference engine then tries to combine the different estimators and derive a better estimation of the T_i. This combination is motivated by the need to minimize the variance of the global estimation error for T_i. To this end, we construct the global estimator of the task T_i as a weighted sum of the estimators obtained from the different sampling primitives. This weighted summation of estimators is known to be the best linear combination in terms of minimization of mean square error [6],

$$\hat{T}_i = \sum_{s\in\mathcal{S}} \lambda_s \hat{T}_i^s \quad \text{with} \quad \lambda_s = \frac{\frac{1}{Var(\hat{T}_i^s)}}{\sum_{l\in\mathcal{S}} \frac{1}{Var(\hat{T}_i^l)}}. \tag{1}$$

Note that the weights are inversely proportional to the estimator error, which in their turn are inversely proportional to the configured sampling rate. Thus, primitives providing estimates with smaller error have a larger impact on the global estimator than those providing estimates with larger error.

In order to explain the operation of our system and how it combines their measurements, we consider three monitoring applications: flow counting, flow size estimation and heavy hitter detection. The analysis and validation are done for the two-well known sampling primitives: packet sampling and flow sampling.

1) Flow counting:: We explain in this section how the central estimator can combine measurements collected from the PS and FS tools to provide a global estimation of the number of flows N crossing the entire network, or part of it, during some observation time d that we define later. A flow is an aggregate of packets sharing some common features. Here we use the basic definition called the 5-tuple definition where packets of a flow share the same source and destination IP addresses and port numbers plus the protocol number. A flow can be flow-sampled using the FS tool or packet-sampled using the PS tool.

Consider the set of monitored paths $\mathcal{A}$. Each path $a \in \mathcal{A}$ consists of a set of monitors. Consider N_a to be the total number of flows crossing the path a. Let $\hat{N}_a^f$ and $\hat{N}_a^p$ be two estimators of N_a using respectively FS and PS and let $Var(\hat{N}_a^f)$ and $Var(\hat{N}_a^p)$ be their corresponding variances. Hence, according to (1), we can derive a better estimation of N_a using a linear combination of these two estimators:

$$\hat{N}_a = \alpha \hat{N}_a^p + \beta \hat{N}_a^f, \tag{2}$$

where,

$$\alpha = \frac{\frac{1}{Var(\hat{N}_a^p)}}{\frac{1}{Var(\hat{N}_a^p)} + \frac{1}{Var(\hat{N}_a^f)}} \quad \text{and} \quad \beta = \frac{\frac{1}{Var(\hat{N}_a^f)}}{\frac{1}{Var(\hat{N}_a^f)} + \frac{1}{Var(\hat{N}_a^p)}}. \tag{3}$$

Note that if a sampling primitive is not executed over a certain path, we set its corresponding weight to 0 and we verify that the weights sum to 1.

Next, we explain how to calculate $\hat{N}_a^f$, $\hat{N}_a^p$ and their corresponding variances $Var(\hat{N}_a^f)$ and $Var(\hat{N}_a^p)$. Consider for this the packet sampling rate vector $P^p = (p_k^p)_{k\in\mathcal{R}}$ and the flow sampling rate vector $P^f = (p_k^f)_{k\in\mathcal{R}}$. $\mathcal{R}$ is the set of monitors in the network, each monitor k is tuned with a packet sampling rate p_k^p and a flow sampling rate p_k^f.

First, using the FS primitive, the probability that a flow is flow-sampled along a path a is equal to: $\pi_a^f = 1 - \prod_{k\in a}(1-p_k^f)$. Consider n_a^f to be the number of flow-sampled flows crossing the path a. We can derive a first estimation of the number of flows along the path a that maximizes the likelihood:

$$\hat{N}_a^f = \frac{n_a^f}{\pi_a^f}. \tag{4}$$

Under independent sampling of flows with probability π_a^f, the number of flow-sampled flows n_a^f follows a binomial distribution whose variance is well known and equal to $N_a.\pi_a^f.(1-\pi_a^f)$. It follows that this path-level estimator of the number of flows has a variance equal to:

$$Var(\hat{N}_a^f) = \frac{N_a^{(t)}.(1-\pi_a^f)}{\pi_a^f}. \tag{5}$$

Now we give the expression of the estimator of N_a using the PS primitive (denoted above $\hat{N}_a^p$). Let n_a^p be the number of packet-sampled flows crossing path a. Let S be the size of a given flow, then the probability that this flow is packet-sampled along path a is equal to $1 - \prod_{k\in a}(1-p_k^p)^S$ (i.e. at least one packet sampled), which can be approximated by $S.\pi_a^p$ for small p_k^p, where $\pi_a^p = \sum_{k\in a} p_k^p$. The average number of packet-sampled flows n_a^p can then be approximated by $\pi_a^p.\Sigma_a.N_a$, where Σ_a is the mean size of 5-tuple flows crossing path a. Hence, we can give a second estimation of N_a:

$$\hat{N}_a^p = \frac{n_a^p}{\pi_a^p.\Sigma_a}. \tag{6}$$

This estimator has a variance equal to

$$Var(\hat{N}_a^p) = \frac{Var(n_a^p)}{(\pi_a^p)^2.(\Sigma_a)^2} = \frac{(1-\pi_a^p.\Sigma_a).N_a}{\pi_a^p.\Sigma_a}. \quad (7)$$

We still need to provide an estimation for the average flow size. We use the number of sampled flows to this end.

Let $O_a = n_a^f + n_a^p$ be the total number of sampled flows along the path a. Given the global estimator for the number of flows $\hat{N}_a$, we can approximate this overhead by: $O_a = \pi_a^f.\hat{N}_a + (1-\pi_a^f)\pi_a^p.\Sigma_a.\hat{N}_a$. This gives the following estimator for the mean size of flows crossing path a:

$$\hat{\Sigma}_a = \frac{O_a - \pi_a^f.\hat{N}_a}{(1-\pi_a^f)\pi_a^p.\hat{N}_a}. \quad (8)$$

Eq. (2) and (8) constitute a system of two equations with two unknowns that we can solve to find the values of $\hat{N}_a$ and $\hat{\Sigma}_a$.

2) Flow size estimation:: Consider C traffic aggregate flows whose volumes in packets are labeled F_1, F_2, ..., F_C. Denote by $\hat{F}_1$, $\hat{F}_2$, ..., $\hat{F}_C$ the corresponding estimators. Each aggregate flow F_i is formed of a set of 5-tuple flows whose volumes are denoted by S_{ji}. Again, denote by $\hat{S}_{ji}$ the best estimator for the size of each of these 5-tuple flows. We have then $\hat{F}_i = \sum_j \hat{S}_{ji}$. As target application and without loosing generality, we define a flow F_i as the set of 5-tuple flows that share the same AS source and AS destination. All AS-to-AS flows are jointly considered, which is often called in the literature the traffic matrix.

Similarly to (1), we can derive a global estimator for the size of flows $\hat{F}_i$ using estimations made by packet-sampled flows, $\hat{F}_i^p$, and flow-sampled flows, $\hat{F}_i^f$:

$$\hat{F}_i = \chi \hat{F}_i^p + \psi \hat{F}_i^f, \quad (9)$$

where,

$$\chi = \frac{\frac{1}{Var(\hat{F}_i^p)}}{\frac{1}{Var(\hat{F}_i^p)} + \frac{1}{Var(\hat{F}_i^f)}} \text{ and } \psi = \frac{\frac{1}{Var(\hat{F}_i^f)}}{\frac{1}{Var(\hat{F}_i^f)} + \frac{1}{Var(\hat{F}_i^p)}}. \quad (10)$$

Next we calculate $\hat{F}_i^p$, $\hat{F}_i^f$ and their corresponding variances, $Var(\hat{F}_i^p)$ and $Var(\hat{F}_i^f)$.

Take a 5-tuple flow S_{ji} crossing path a and belonging to aggregate flow F_i, and let $(s_{kji}^p)_{k\in a}$ be the number of packet-sampled packets from this 5-tuple flow in monitor k. We can easily derive an estimation for the size of this 5-tuple flow:

$$\hat{S}_{ji}^p = \sum_{k\in a} \lambda_k \hat{S}_{kji}^p, \quad \text{with} \quad \lambda_k = \frac{\frac{1}{Var(\hat{S}_{kji}^p)}}{\sum_{l\in a} \frac{1}{Var(\hat{S}_{lji}^p)}}, \quad (11)$$

where $\hat{S}_{kji}^p = \frac{s_{kji}^p}{p_k^p}$ is the best local estimator for the size of the 5-tuple flow in the monitor $k \in a$. This local estimator has a variance equal to $Var(\hat{S}_{kji}^p) = S_{ji}^p.(1-p_k^p)/p_k^p$. It follows that the estimator $\hat{S}_{ji}^p$ is equal to:

$$\hat{S}_{ji}^p = \frac{1}{\varphi_a}.\sum_{k\in a} \frac{s_{kji}}{1-p_k}, \quad \text{where} \quad \varphi_a = \sum_{l\in a} \frac{p_l^p}{1-p_l^p}. \quad (12)$$

The variance of this estimator is simply equal to:

$$Var(\hat{S}_{ji}^p) = S_{ji}/\varphi_a. \quad (13)$$

Hence, under independent sampling of packets in the different monitors, we can derive the expression of $\hat{F}_i^p = \sum_j \hat{S}_{ji}^p$ and its corresponding variance $Var(\hat{F}_i^p) = \sum_j Var(\hat{S}_{ji}^p)$.

Now we move to deriving the estimator of flow size using flow sampling. Consider s_{ji}^f to be the number of flow-sampled packets of the 5-tuple flow S_{ji}. Hence:

$$\hat{F}_i^f = \frac{\sum_j s_{ji}}{\pi_a^f}. \quad (14)$$

This estimator has a variance equal to:

$$Var(\hat{F}_i^f) = \frac{1-\pi_a^f}{\pi_a^f}.\sum_j (Sji)^2 = \frac{1-\pi_a^f}{(\pi_a^f)^2}.\sum_j (sji)^2. \quad (15)$$

3) Heavy hitter detection:: The goal here is to identify the top flows with the most traffic volume. These flows are used by operators to understand application patterns and resource hogs, as well as for traffic engineering and accounting. In this paper, we track heavy hitters at the AS level, i.e. we define a flow as the total volume of traffic generated by each stub AS and we make sure to count both outgoing and incoming traffic in each AS flow.

For the calculation of the outgoing and incoming traffic for each AS, we use the method used for flow size estimation while considering only large ASes in the optimization procedure. We estimate all AS flows but we only optimize sampling rates over those contributing to more than some percentage of the total network traffic. We present results for a 5% threshold under the Large ASes task. Some ASes are smaller than this threshold but their traffic might still be reported to the central collector, yet they are not included in the optimization procedure and they not returned to the monitoring application. For this application, our system will try to minimize the volume of undesirable measurement traffic coming from small flows.

B. Network Reconfiguration Engine

Given a list of measurement tasks $\mathcal{T}$ and an overhead constraint measured in terms of reported NetFlow records (Target Overhead $\mathcal{TO}$), our system adaptively adjusts its configuration to answer the requirements of the multiple tasks while tracking short-term and long-term variations in the traffic. A configuration is a selection of sampling rates of the different primitives on the different interfaces of network routers (or monitors). This configuration is periodically updated as a function of the overhead and in a way to optimize the accuracy of the considered measurement tasks. Next, we present the architectural ideas behind our system.

1) Overhead prediction: The optimization procedure requires the prediction of overhead after any reconfiguration, in order to find the optimal configuration that keeps the real overhead within a target value while providing the best possible measurement accuracy for the considered tasks. Defined

as the total volume of NetFlow records collected at the central cognitive engine, the overhead can be expressed as follows:

$$\mathcal{O} = \sum_{M \in \mathcal{R}} \sum_{a \in \Gamma_M} (\pi_a^f . N_a + (1 - \pi_a^f)\pi_a^p . N_a . \Sigma_a), \quad (16)$$

where $\Gamma_M \subset \mathcal{A}$ is the subset of paths containing the monitor M. In order to track signification variations in the traffic while removing undesirable noise, we use the Exponentially Weighted Moving Average (EWMA) filter which is a memoryless moving average filter whose weights are exponentially decreasing from more recent historical samples to older ones. Hence, an EWMA filter gives more importance to recent observations while still not discarding older observations entirely.

Let $\overline{N}_a$ and $\overline{\Sigma}_a$ be respectively the smoothed version of the number of flows across a path a and their mean size expressed in number of packets. Every period d, which is the period at which sampling rates are reconfigured, we update these moving averages as follows:

$$\overline{N}_a \leftarrow \delta N_a + (1 - \delta)\overline{N}_a, \quad (17)$$

$$\overline{\Sigma}_a \leftarrow \delta \Sigma_a + (1 - \delta)\overline{\Sigma}_a, \quad (18)$$

where N_a and Σ_a are the last estimations provided by Eq. (2) and (8) over the last d period. $\delta = \frac{2}{(n+1)}$ is the smoothing factor where n is the window length over which we smooth the traffic. This factor allows us to choose the time scale τ at which we track variations in the traffic. For instance, if we want to track changes on hourly scale (i.e. $\tau = 3600s$), we calculate the window length $n = \frac{\tau}{d}$, d being the period of configuration updates.

Using (17) and (18) and a slightly simplified version of (16), we can now give the analytical expression of the overhead prediction. This overhead prediction is no other than the smoothed version of the number of collected NetFlow records per d period.

$$\mathcal{O} = \sum_{M \in \mathcal{R}} \sum_{a \in \Gamma_M} (\pi_a^f . \overline{N}_a + \pi_a^p . \overline{N}_a . \overline{\Sigma}_a). \quad (19)$$

The overhead prediction method works as follows. For each path $a \in \mathcal{A}$, first we look for initial values for the number of flows $\overline{N}_a$ and the mean flow size $\overline{\Sigma}_a$. To do so, we can use values of the same period of the last week or the last day. Then, we start using the collected traffic to update estimators and predict the overhead. Algorithm 1 explains how the EWMA filter is implemented for prediction according to Eq. (19). Note that the smoothing factor δ plays a crucial role in the overhead prediction. In fact, using short time scale can disrupt the system with unnecessary details specific to a particular observation period while the use of a large time scale can lead to the loss of important changes in the traffic. One has to find the suitable time scale that addresses the tradeoff between these two extremes.

2) Optimization method: The optimization method is motivated by the need to coordinate responsibilities across the different monitors to improve the accuracy. This method is fed by the list of tasks T_i, their associated weights γ_i, and

Data: Number of flow-sampled flows and packets $(n_a^f)_{a \in \mathcal{A}}$ and $(s_{aji}^f)_{a \in \mathcal{A}}$.
Number of packet-sampled flows and packets $(n_a^p)_{a \in \mathcal{A}}$ and $(s_{kji}^p)_{k \in \mathcal{R}}$.
The predictions: $\overline{N}_a$ and $\overline{\Sigma}_a$, and the sampling rate vectors P^f and P^p.
Time scale τ and computation period d
Result: The expression of the *overhead prediction* $\mathcal{O}$
begin
  $n \leftarrow \frac{\tau}{d}$; $\delta \leftarrow 2/(n+1)$;
  foreach $a \in \mathcal{A}$ **do**
    calculate $\hat{N}_a^f = \frac{n_a^f}{\pi_a^f}$; calculate $\hat{N}_a^p = \frac{n_a^p}{\pi_a^p . \overline{\Sigma}_a}$.;
    calculate $\hat{N}_a = \alpha \hat{N}_a^p + \beta \hat{N}_a^f$;
    \\Estimate the number of 5-tuple flows.
    $\overline{N}_a \leftarrow \delta \hat{N}_a + (1 - \delta)\overline{N}_a$;
    calculate $\hat{\Sigma}_a = \frac{O_a - \pi_a^f . \hat{N}_a}{\pi_a^p . \hat{N}_a}$;
    \\Estimate the mean size of 5-tuple flows.
    $\overline{\Sigma}_a \leftarrow \delta \hat{\Sigma}_a + (1 - \delta)\overline{\Sigma}_a$;
    \\Derive the expression of the overhead prediction of the next period.
    $O_a \leftarrow \pi_a^f . \overline{N}_a + \pi_a^p . \overline{N}_a . \overline{\Sigma}_a$;
  end
  \\Derive the global overhead prediction expression.
  $\mathcal{O} = \sum_{M \in \mathcal{R}} \sum_{\mathcal{A} \in \Gamma_M} O_a$;
  return $\{\mathcal{O}\}$
end

Algorithm 1: Overhead prediction method

the normalized error of the global estimation of each task $\hat{T}_i$, $MRE(\hat{T}_i)$. Our objective is to find the optimal sampling rate vector that minimizes this utility function:

$$U = \sum_i \gamma_i MRE(\hat{T}_i), \quad (20)$$

under the following constraints:

$$\mathcal{O} \leq \mathcal{T}\mathcal{O}, \quad (21)$$

$$p_k \leq SR_{max}, \quad (22)$$

$$p_k \geq SR_{min}. \quad (23)$$

SR_{min} and SR_{max} are respectively the minimum and maximum sampling rate values. To solve this constrained optimization problem we define the corresponding Lagrangian:

$$L = U + \delta(O - TO) + \sum_k a_k(p_k - SR_{max}) + \sum_k b_k(SR_{min} - p_k).$$

(δ, a_k, b_k) is the set of Lagrange multipliers that enforce the satisfaction of the constraints. We solve this Lagrangian by an iterative procedure using the Newton method (refer to [9], Chapter 9.5). The idea of the method is as follows. We start with an initial guess of the optimal sampling rate vector. Then, at each iteration, we use the Newton method to go into a better direction while using a sophisticated line search algorithm to

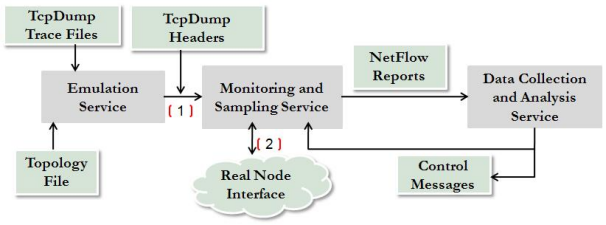


Fig. 2. Experimental platform

find the best step value. We continue until we either reach the global minimum or we exceed the maximum number of iterations.

IV. Evaluation methodology

In order to evaluate the performance of our system, we developed an experimental platform Monlab [10][2]. This platform has the following main features: *(i)* it is fed by real traffic captured on a transit link then spread and played over an emulated network topology, *(ii)* it includes real NetFlow-like tool for traffic monitoring on all router interfaces of the emulated topology, and *(iii)* it implements the central unit as it should be in reality.

A. Experimental platform

As shown in Figure 2, our experimental platform, is composed of three services: *(i)* the traffic emulation service, *(ii)* the traffic monitoring and sampling service, and *(iii)* the data collection and analysis service. Routers can be either virtual nodes connected by virtual links, or real routers connected by real links. The first service is responsible of generating the emulated traffic across the network routers. The second service implements packet sampling and flow monitoring a la NetFlow on each router interface. The later functionality is provided by SoftFlowd [11], an open source free software capable of NetFlow measurements in high speed networks. The third service mainly consists of the Flowd tool in the SoftFlowd package. It is a centralized service that collects NetFlow records, correlates them to better estimate network traffic, and then runs our adaptive algorithm to decide on which sampling rates to update.

SoftFlowd requires network traffic in the TcpDump format. Unfortunately, obtaining real traffic data from an entire backbone network is a hard issue. To cope with, we proceed in the following way. We first seek unsampled packet traces collected on high speed transit links. We consider for this study the ones coming from the Japanese MAWI project [12]. We parse the traces for the IP prefixes, and we dispatch them over the Autonomous Systems (ASes) connected to the edge routers of the emulated topology. The dispatching is done randomly according to some predefined weights that determine the importance of each stub AS. Our system allows to define the length of the prefix as a function of the granularity of the dispatching we want to achieve. For this work, we consider the /16 prefix as the unit for IP address assignment to ASes.

Once addresses are allocated, the packets in the TcpDump trace are split accordingly between the different ASes connected to the emulated topology. Shortest routes are calculated. Then packets in the TcpDump trace are associated to the different monitors over their respective paths across the network with the correct timestamps derived from the trace. SoftFlowd samples then packets, form flows and send them back to the central collector. This sampling and monitoring is done in parallel on all network router interfaces.

B. Scenario description

Our platform requires the definition of a network topology over which it dispatches and replays a real traffic. We chose to experiment over network topologies similar to the Geant topology [13]. The weights of ASes needed for traffic dispatching are set according to the size of stub ASes in Geant and we make sure these weights sum to 1. An AS of weight w will see itself attributed $100.w\%$ of the prefixes available in the trace and will see its traffic (incoming or outgoing) being around $100.w\%$ of the total trace traffic, both at the flow and packet levels (random prefix allocation). Once topology and weights are set, TcpDump traces coming from [12] are replayed.

The three services of the platform run on one machine each. In order to initialize the different estimators, we run a first computation without sampling. As target applications, we consider the three applications we introduced in Section III-A1: flow counting, flow size estimation and heavy hitter detection. The objective is to minimize the weighted normalized estimation error (Eq. 20).

V. Validation results

In this section, we show first the practical benefits of deploying our system by comparing it to application-specific sampling systems. Second, we evaluate the efficiency of our adaptive solution. Last, we present a global sensitivity analysis to study the importance of the different parameters.

A. Comparison with application-specific sampling

For this experiment, we set the timer d for updating sampling rates to 5 minutes, the time scale β to 3600s, the minimum possible sampling rate SR_{min} to 0.0005 and the maximum possible one SR_{max} to 1. The $\mathcal{TO}$ is set to 200 NetFlow-records/s.

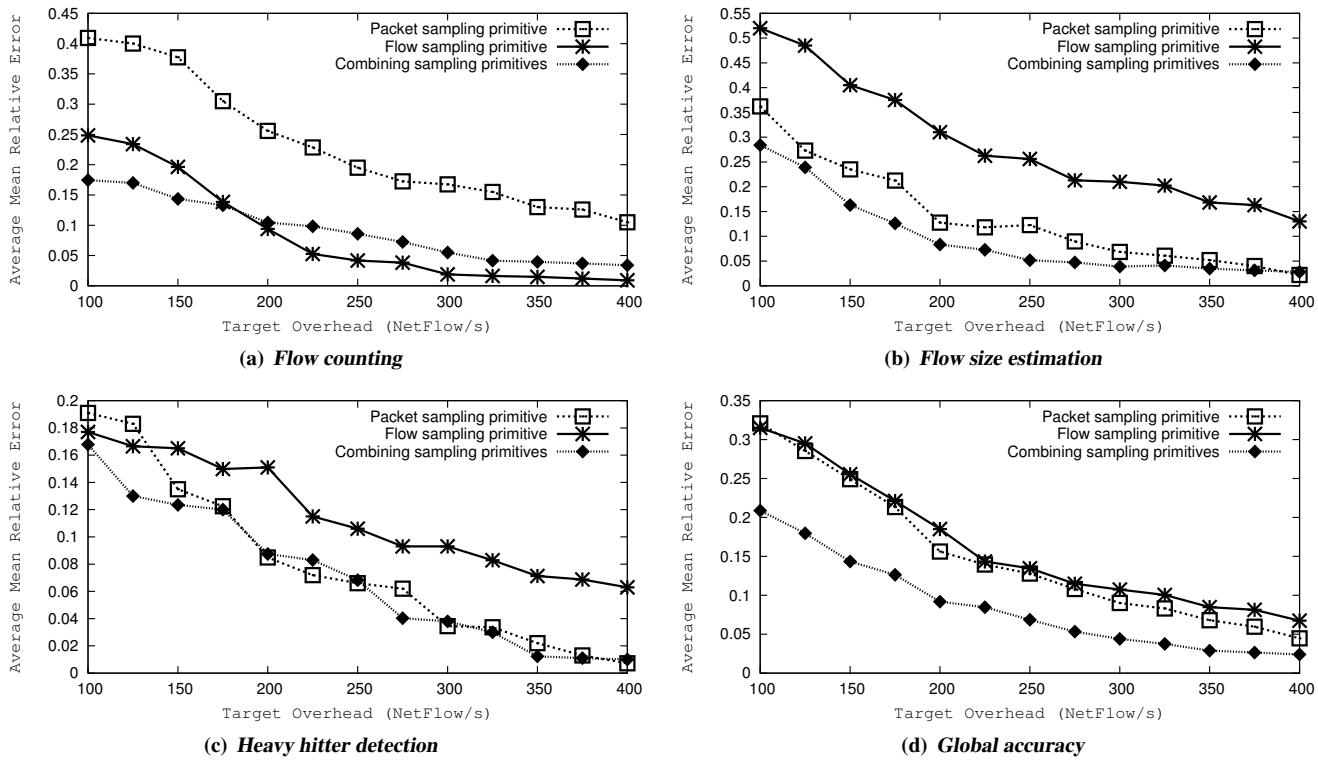
We plot in Figure 3 the average mean relative error for three specific monitoring applications:

- Flow counting application (FC).
- Flow size estimation (FS).
- Heavy hitter detection (HH).

For these applications, we use our proposed joint method and we compare its performance with the two application-specific primitives when used separately, i.e., packet sampling and flow sampling primitive.

We plot in Figure 3(d) the global accuracy defined as the global weighted accuracy as described in Eq. (20). This figure

[2]URL: http://planete.inria.fr/MonLab/



(a) ***Flow counting*** **(b)** ***Flow size estimation***

(c) ***Heavy hitter detection*** **(d)** ***Global accuracy***

Fig. 3. Average mean relative error vs. Target overhead ($\mathcal{TO}$) for three applications: Our approach vs. two application-specific approaches.

shows that our solution provides the best global accuracy for the different monitoring applications. It combines measurements of the different monitoring primitives to improve the accuracy of the global estimator. As we can see in this figure, in order to have similar global accuracy to our joint method using application-specific primitive, one has to use a $\mathcal{TO}$ value larger than 150% of the value used by the method we propose. In fact, each primitive focuses on achieving a specific application. Therefore, while the use of a single primitive allows improving results for its specific monitoring application, it provides inaccurate results for the other monitoring applications. These results are illustrated in figures 3(a),3(b) and 3(c). We observe that each primitive provides accurate measurements for its specific application and less accurate results for the other applications. For instance, packet sampling primitive improves accuracy of FS and HH applications while it provides a large estimation error for FC application. On the other hand, flow sampling primitive improves results of FC application and provides less accurate results for FS and HH applications. We notice that our solution based on combining results of the different monitoring primitives improves accuracy of the different applications especially when the value of $\mathcal{TO}$ is small. In this case, sampling rate values are low and each single primitive provides inaccurate results. We can thus combine their measurements in order to improve the global accuracy. Hence, using our solution based on combining different sampling primitive measurements, allows not only optimizing resource consumption but also improving the global accuracy as well as the accuracy of each specific monitoring application. Furthermore, these figures also show the large impact of the monitoring constraint ($\mathcal{TO}$) on measurement accuracies. We observe in particular the clear reduction of estimation errors when increasing the value of $\mathcal{TO}$.

B. System efficiency and adaptability

Now we want to study the impact of the weights used in the global utility function on the behavior of our system. The parameters of experiments are set as in the previous section. We run three scenarios while changing each time the assigned weight value to each monitoring application. Periodically, we measure the mean relative error. Then, we consider the average over all these values measured during the experiment. Table I and II present a summary of experimental results for a selection of scenarios. We notice that assigned weights have an impact on the behavior of our system. We observe I that increasing the weight value assigned to a given monitoring application allows reducing the average mean relative error (AMRE) of its measurement.

Consider the following experiments: in a first scenario (SC1), the operator assigns equal weights to the different tasks (0.33). In order to validate the operation of our solution we run again the same experiment in SC2 while increasing the weight assigned to the FC application from 0.33 to 0.5. We see in table I that the error of FC application is reduced from 0.1045 to 0.0738. In fact, by increasing the weight assigned to FC application the system finds automatically the new best configuration that minimizes the global error while respecting

TABLE I
SUMMARY OF ASSIGNED WEIGHTS AND EXPERIMENTAL RESULTS FOR A SELECTION OF SCENARIOS

Scenario	Flow counting		Flow size estimation		Heavy hitter detection		Global accuracy
	assigned weight	AMRE	assigned weight	AMRE	assigned weight	AMRE	
SC1	0.333	0.1045	0.333	0.0834	0.333	0.06745	0.08511
SC2	0.5	0.0738	0.25	0.164	0.25	0.0865	0.09952
SC3	0.5	0.0889	0.5	0.114	-	-	0.1014

TABLE II
AVERAGE SAMPLING RATE VALUES AND REPORTED NETFLOW RECORDS FOR A SELECTION OF SCENARIOS

Scenario	Flow Sampling		Packet Sampling	
	Average sampling rate value	Percentage of reported NetFlow records	Average sampling rate value	Percentage of reported NetFlow records
SC1	0.016	39.65%	0.314	60.35%
SC2	0.0287	62.84%	0.197	37.16%
SC3	0.03613	58.23%	0.023	41.77%

the monitoring constraint ($\mathcal{TO}$). We see in table II that the new average sampling rate value for flow sampling primitive increases from 0.016 to 0.0287 introducing an increase of the percentage of flow sampled flows (reported Netflow record for flow sampling primitive) from $39,65\%$ to $62,84\%$. Hence, for a *set of applications with their corresponding weights*, our solution finds the best monitors configuration that optimizes the global accuracy while respecting monitoring constraints. This result is illustrated in SC3. We assign the same weight (0.5) to FC application and we use only two applications instead of three. We observe in table II that the system finds a new optimal configuration for this new set of applications and their assigned weights.

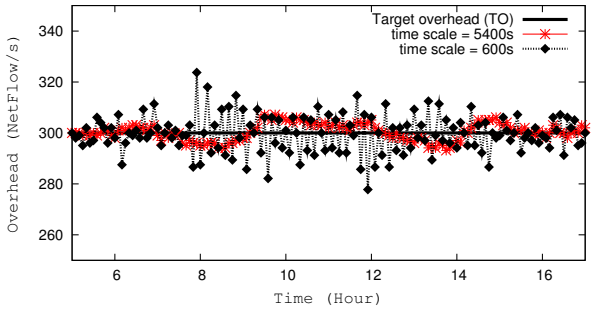


Fig. 4. Resulting overhead vs. time using two time scales to track traffic variations

C. Overhead prediction process validation

In order to evaluate the performance of the overhead prediction method, we plot in figure 4 the evolution of the measured overhead (exported NetFlow records) over time. We observe that for the two time scale values the system maintains the overhead around the $\mathcal{TO}$. In fact, the system tries to profit from the available resources in order to provide the best possible accuracy. However, the use of a small time scale ($\tau = 600s$) leads to an oscillating behavior of the overhead since the system tracks more details and changes in the traffic while tracking changes on a large scale ($\tau = 5400s$) leads to a stable behavior of the overhead. This is due to the avoidance of some details and variations in the traffic specific to one observation period.

VI. CONCLUSIONS

In this paper, we have presented an adaptive system that combines different existing sampling primitives in order to support a large spectrum of monitoring tasks while providing the best possible accuracy. Our system coordinates responsibilities between the different monitors and shares resources between the different sampling primitives. Experimental Results proved the ability of our system to keep the resulting overhead around a target value. Compared to application-specific solutions, our system has shown its advantages in providing more accurate results especially for low values of $\mathcal{TO}$. Our system is practical and provides a flexible optimization method based on overhead prediction that reconfigures monitors according to monitoring applications requirements and network conditions.

Our ongoing work is centred on the distribution of the control.

REFERENCES

[1] G. Cantieni, G. Iannaccone, C. Barakat, C. Diot, and P. Thiran, "Reformulating the monitor placement problem: Optimal networkwide sampling," in *Proc. of CoNeXT*, 2006.

[2] V. Sekar, M. Reiter, W. Willinger, H. Zhang, R. Kompella, and D. Andersen, "cSamp: A system for network-wide flow monitoring," in *Proc. 5th USENIX NSDI*, 2008.

[3] N. Hohn and D. Veitch, "Inverting sampled traffic," in *Proc. of IMC*, 2003.

[4] C. Estan and G. Varghese, "New directions in traffic measurement and accounting," in *Proc. of ACM SIGCOMM*, 2002.

[5] Cisco, "Netflow services and applications," *White Paper*, 2000.

[6] N. Duffield, C. Lund, and M. Thorup, "Optimal combination of sampled network measurements," in *IMC 2005*, 2005.

[7] H. Z. Vyas Sekar, Michael K Reiter, "Revisiting the case for a minimalist approach for network flow monitoring," in *Proc. of IMC*, 2010.

[8] K. Keys, D. Moore, and C. Estan, "A robust system for accurate real-time summaries of internet traffic," in *Proc. of SIGMETRICS*, 2005.

[9] S. Boyd and L. Vandenberghe, "Convex optimization," in *Cambridge University Press*, 2004.

[10] A. Krifa, I. Lassoued, and C. Barakat, "Emulation platform for network wide traffic sampling and monitoring," *TRAC*, 2010.

[11] , "Softflowd flow-based network traffic analyser," http://www.mindrot.org/projects/softflowd/.

[12] "Mawi working group traffic archive," http://tracer.csl.sony.co.jp/mawi/.

[13] , "Geant: The european research and academic backbone." http://www.geant.net/.