Université de Nice - Sophia Antipolis – UFR Sciences
École Doctorale STIC

THÈSE

Présentée pour obtenir le titre de :

Docteur en Sciences de l'Université de Nice - Sophia Antipolis

Spécialité : INFORMATIQUE

par

Imed LASSOUED

Équipes d'accueil : INRIA Planete Team, Sophia Antipolis

ADAPTIVE MONITORING AND MANAGEMENT OF INTERNET TRAFFIC

Thèse dirigée par : Dr. Chadi BARAKAT

Soutenance le jj mois 2011, devant le jury composé de :

Président :	First Name	LAST NAME	Affiliation
Directeur :	Chadi	BARAKAT	INRIA
Rapporteurs :	Guy	LEDUC	University of Liège
	Laurent	MATHY	Lancaster University
Examinateurs :	Luigi Alfredo	GRIECO	Politecnico di Bari
	Emmanuel	LETY	Product Manager at UDcast

Thèse

Solutions adaptatives pour la métrologie et la gestion du trafic de l'Internet

Adaptive Monitoring and Management of Internet Traffic

Imed LASSOUED

Mois 2011

ADAPTIVE MONITORING AND MANAGEMENT OF INTERNET TRAFFIC

by
Imed LASSOUED
Directeurs de thèse: Dr. Chadi BARAKAT

Équipes d'accueil : PLANETE – INRIA Sophia Antipolis

ABSTRACT

Traffic measurement allows network operators to achieve several purposes such as traffic engineering, network resource provisioning and management, accounting and anomaly detection. However, existing solutions suffer from different problems namely the problem of scalability to high speeds, the problem of detecting changes in network conditions, and the problem of missing meaningful information in the traffic. The main consequence of this trend is an inherent disagreement between existing monitoring solutions and the increasing needs of management applications. Hence, increasing monitoring capabilities presents one of the most challenging issues and an enormous undertaking in a large network. This challenge becomes increasingly difficult to meet with the remarkable growth of the Internet infrastructure, the increasing heterogeneity of users' behavior and the emergence of a wide variety of network applications. In this context, we present the design of an adaptive centralized architecture that provides visibility over the entire network through a network-wide cognitive monitoring system.

We consider the following important requirements in the design of our network-wide monitoring system. The first underscores the fact that the vendors do not want to implement sophisticated sampling schemes that give good results under certain circumstances. They want to implement simple and robust solutions that are well described by some form of a standard (i.e. sFlow, NetFlow). Thus, we decide to design a new solution that deals with existing monitoring techniques and tries to coordinate responsibilities between the different monitors in order to improve the overall accuracy. The second requirement stipulates that the monitoring system should provide general information of the entire network. To do so, we adopt a centralized approach that provides visibility over the entire network. Our system investigates the different local measurements and correlates their results in order to address the tradeoff between accuracy and monitoring constraints. And the last requirement indicates that the monitoring system should address the scalability problem and respect monitoring constraints. To this end, our system relies on a network configuration module that provides a responsive solution able to detect changes in network conditions and adapt the different sampling rates to network state. At the same time it avoids unnecessary details and oscillations in the traffic in order to keep the resulting overhead within the desired bounds. The network reconfiguration module deals with local monitoring tools and adjusts automatically and periodically sampling rates in order to coordinate responsibilities and distribute the work between the different monitors.

ACKNOWLEDGMENTS

I wish to express my gratitude to my supervisor, Dr. Chadi Barakat, for his continuous encouragement, guidance and support during my thesis work. This work and its underlying details were greatly influenced and enriched by his assistance and guidance. I extend my thanks to him for having so much patience with me and for always giving me his honest opinion. It has been a real pleasure to work with him.

I will always remain grateful to Dr. Konstantin Avrachenkov for his support, advices and constructive comments, as well as for taking the time to discuss my research goals and providing me with helpful feedback.

Then, I am much obliged to //Guy Leduc, Laurent Mathy, Luigi Alfredo Grieco, and Emmanuel Lety for accepting to be members of the thesis jury and for their pertinent feedback. It is a great honor for me to have them evaluate my work.

This thesis would not have been possible without the aid and the expertise of Walid DABBOUS, director of the Planete Project-team at INRIA. I would also like to thank all the academic and support staff of the department; special attention goes to Thierry Turletti, Arnaud Legout, Thierry Parmentela and, Mathieu Lacage; then, I would like to acknowledge the fruitful collaboration I have had with the research engineer Amir Krifa.

My appreciation goes as well to the Ecode project researchers Dimitri Papadimitriou, Kave Salamatian and philippe owezarski for their constructive comments and suggestions during the prototype design and implementation and for the technical assistance I have received.

There are also many colleagues to be considered individually, but some names stand out; particular thanks go to Amine, Roberto, Laurie, Anais, Giovanni, Shafqat, Karim, Stevens and Mohamed with whom I had many discussions about everything. Finally, my life would not have been the same without my family that has remained close to me even if we were thousands of miles away. I am deeply thankful to them.

Imed LASSOUED

Sophia Antipolis, France

Dedicated to my parents, Habib and Faiza

CONTENTS

FIGURES

TABLES

1

INTRODUCTION: OVERVIEW AND CONTRIBUTION

Traffic measurement and monitoring are important tasks in order to understand the performance of a network infrastructure and to efficiently manage network resources. However, the remarkable growth of the Internet infrastructure and the increasing heterogeneity of applications and users' behavior make more complex the manageability and monitoring of **I**nternet **S**ervice **P**rovider (ISP) networks and raises the cost of any new deployment. The main consequence of this trend is an inherent disagreement between existing monitoring solutions and the increasing needs of management applications. Hence, in this introductory chapter, we will briefly review the current context of monitoring solutions and we will discuss the technical challenges facing increasing monitoring capabilities that must still to be overcome and that motivate this work. Then, we will present the main topics to which we have contributed during this thesis.

Contents

1.1 Context and Problem Statement

The Internet infrastructure has seen an immense growth in importance and size. One of the main reasons for the success of the Internet is its service model that emphasizes flexibility: any computer can send any type of data to any other computer at any time [52]. Internet users consider the Internet as a medium that provide the connectivity for the end-user applications and they are shielded from having to know the Internet infrastructure or details of how it works. In fact, the Internet protocol stack follows the principle of information hiding. This protocol stack defines the rules for how user-level data is transformed into network packets for transport across the network. The users do not need to know how many network packets are generated when using Internet or how these packets are routed through the network. They care only about the robustness, flexibility, reliability and simplicity of Internet usage. Hence, in order to ensure reliable operation and smooth growth of computer networks, network operators need mechanisms to constantly monitor and study data traffic. This often relies on network traffic measurement as a methodology to collect and analyze data regarding the operation and performance of network protocols.

Network monitoring is an enormous engagement in a large network. It consists of monitoring a network using a set of monitors deployed in network routers, with the purpose of monitoring the performance of the deployed network infrastructure and understanding the behavior of the network and its users. Monitoring is a main process in the engineering, and operation of a network. This can serve several purposes such as traffic engineering [55], network resource provisioning and management, accounting [40] and anomaly detection [19, 33, 98]. Therefore, improving network monitoring systems and capabilities has a direct impact on network performance, user experience and operators revenues.

The authors in [95] have defined four main reasons why network traffic measurement is a useful methodology:

- *Network Troubleshooting*. Computer networks are not infallible. Often, a single malfunctioning piece of equipment can disrupt the operation of an entire network, or at least degrade performance significantly. Examples of such scenarios include ”broadcast storms”, illegal packet sizes, incorrect addresses, and security attacks. In such scenarios, detailed measurements from the operational network can often provide a network administrator with the information required to pinpoint and solve the problem.

- *Protocol Debugging*. Developers often want to test out ”new, improved” versions of network applications and protocols. Network traffic measurement provides a means to ensure the correct operation of the new protocol or application, its conformance to required standards, and (if necessary) its backward-compatibility with previous versions, prior to unleashing it on a production network.

- *Workload Characterization*. Network traffic measurements can be used as input to the workload characterization process, which analyzes empirical data (often using statistical techniques) to extract salient and representative properties describing a network application or protocol. Knowledge of the workload characteristics can then lead to the design of better protocols and networks for supporting the application.

- *Performance Evaluation*. Finally, network traffic measurements can be used to determine how well a given protocol or application is performing in the Internet. Detailed analysis of network measurements can help identify performance bottlenecks. Once these performance problems are addressed, new versions of the protocols can provide better (i.e., faster) performance for the end users of Internet applications.

Currently, there is an increasing interest in efficient passive measurement solutions that scale with the network traffic while providing the maximum possible information.

Until recently, network operators have used coarse-grained data provided by the different monitors (deployed in network routers). These data consist of aggregate bytes and packets counts in each direction at a given interface, aggregated over time windows of a few minutes. However, these data are no longer sufficient to engineer and manage networks. This is due to two reasons: the Internet infrastructure growth in terms of geographical size and importance and the tremendous success of Internet-based applications: *(i)* Internet infrastructure growth: The Internet has known an impressive growth in terms of size, connectivity, importance and capacity [21]. Its success is in great part due to its five base design principles (modularization by layering, connectionless packet forwarding, end-to-end principle, uniform inter-networking principle and simplicity principle). One cost we pay for this success and growth of Internet is the difficulty of network monitoring and the inability to efficiently respond to new monitoring application requirements. Furthermore, with an increasing reliance on the Internet infrastructure for economic and social activities, the impact of network-wide attack in the form of worms

or viruses is also increasing [76, 88, 77]. These malicious phenomena make the measurement and monitoring of Internet traffic even more important. Hence, the Internet has progressively become an infrastructure that is architecturally more complex to operate. This complexity progressively impacts the Internet robustness and reliability and in turn impacts its scalability. *(ii)* success of Internet-based applications: The evolution of the Internet has been accompanied by the development, growth, and use of a wide variety of network applications [95, 71]. These applications range from text-based utilities such as file transfer, remote login, electronic mail, and network news from the early days of the Internet, to the World-Wide Web, the video-conference applications, multimedia streaming, and electronic commerce on today's Internet. In fact, the application area has seen rapid growth in popularity of new emerged applications like social networks (e.g., Facebook), peer-to-peer (p2p) systems (e.g., Skype, Bit-Torrent), and cloud-based applications like Google Apps [5] and Salesforce [10]. For example, P2P traffic accounted for up to 60% of traffic on most networks [62]. This shows that P2P has become a mainstream. The emergence of these new applications has changed the characteristics and communications patterns of existing applications. Hence, network operators need to continuously monitor and manage traffic characteristics in their networks. Network monitoring allows operators achieving several purposes like traffic engineering and network resource provisioning and management. Moreover it allows detecting anomalies and any types of malicious behavior within applications (e.g., p2p system vulnerabilities [6, 24, 78, 92], attacks on Ajax-based web services and in general, prepare their networks better against any major application trends.

Hence, due to the immense success and growth of Internet infrastructure and the emergence of new applications, and their rapidly changing characteristics, Network operators need more finely differentiated information on the use of their network and need to obtain fine-grained flow measurements. There exist different ways to perform and improve traffic measurements. The various solutions present a trade-off between the accuracy of the measurement and the amount of computing resources they require. Currently, router-level measurement solutions such as NetFlow [30, 32, 79]) represent the most widely deployed and popular approach used for traffic monitoring. The widely available nature of NetFlow across many modern routers makes it an ideal candidate for ubiquitous low-cost network monitoring. However, this solution presents some shortcomings including the problem of setting sampling rates at low values to cope with the increasing trend in line speed. This is known to introduce a serious bias against small flows and fine-grained measurements [48, 43]. In fact, because of technological and resource constraints, modern routers cannot each record all packets or flows that pass through them. Instead, they rely on a variety of sampling techniques to selectively record as many packets as their CPU and memory resources allow. Each router independently selects a packet with a sampling probability (typically between 0.001 and 0.01) and aggregates the selected packets into flow records. While sampling makes passive measurement technologically feasible (i.e., op-

erates within the router constraints), the overall fidelity of flow-level measurements is reduced. Another problem comes from the static deployment of sampling rates since a static placement of monitors cannot be optimal given the short-term and long-term variations in traffic due to re-routing events, anomalies and the normal network evolution. On the other hand, some network behavior may not be accurately detected with low sampling rates. This particularly refers to anomaly detection. However, sampling with high sampling rate produces vast amounts of data that need to be sent to the collector and processed afterwards especially in periods of high traffic. Hence, we need to conceive a new monitoring system that responds to these different requirements. Packet Sampling has been available for many years. It has been successfully used in many applications, like accounting and billing with acceptable accuracy. However, in order to preserve the accuracy and provide accurate estimations, the sampling mechanism should adapt to network traffic state and monitoring application requirements. Sampling mechanism should be dynamic in order not to lose any important information during the busy periods of traffic bursts as well as to get enough information during idle states. Unfortunately we do not have many possibilities to change sampling techniques applied in network equipment. However we can change sampling rate, which means that we can control the accuracy by controlling the number of samples captured in the defined time period.

In this context, we need to design a novel architecture for network traffic monitoring. This architecture should extend local existing monitoring tools with a central unit that adaptively configures their sampling rates. We need to multiply the monitoring points inside the network, couple their observations and find the best network-wide configuration as a function of the monitoring needs. Moreover, one has to adapt the configuration on the fly if he wants the monitoring to adapt to changing network conditions. That should lead to better performances than in the case of static uncorrelated configuration.

Some recent proposals take this previous direction and try to provide a widespread monitoring infrastructure that coordinates monitoring responsibilities and distribute the work between the different monitors e.g. [48, 86, 28, 89]. The common objective of these efforts is to design a distributing monitoring system that samples traffic in a cost-effective manner by carefully placing monitors and configuring their sampling rates. For example, the authors in [86] present a framework for distributing work across routers to achieve network-wide monitoring goals while respecting router resource constraints. The responsibility to monitor the different traffic flows is distributed among the different routers in an exclusive way. Confronted with the generalization of monitoring in operational networks, the authors in [28] propose placement and packet sampling solutions that coordinate responsibilities across different network routers to lower the cost of deploying monitoring infrastructure, while maximizing the benefits of their infrastructure and improving measurement accuracies. The common objective of these efforts is to design a distributing monitoring system that sample traffic in a cost-effective manner by

carefully placing monitors and configuring their sampling rates.

Recent works have demonstrated the benefits of a network-wide approach for traffic engineering [99] and network diagnosis [19, 96], and have suggested that a centralized approach can significantly reduce management complexity and operating costs [57, 17]. Other works on network-wide monitoring have focused on the placement of monitors at appropriate locations to cover all routing paths using as few monitors as possible [89].

In this context, we present in this thesis the design of an adaptive centralized architecture that provides visibility over the entire network through a network-wide cognitive monitoring system. The proposed architecture should drive the sampling rates on the interfaces of network routers to achieve the maximum possible accuracy, while adapting itself to any change in network traffic conditions. It adopts an adaptive centralized approach to coordinate responsibilities across monitors by adjusting their sampling rates. Furthermore, it extends the existing NetFlow monitoring tools with a cognitive engine that correlates collected measurements from all routers to infer a better global view of the network traffic. Moreover, it automatically reconfigures the sampling rates in the different monitors according to the monitoring application requirements and resource consumption constraints.

1.2 Thesis Contributions

Network monitoring is an immense management activity for network operators in order to determine the composition of the network traffic, to understand the behavior of users, and to monitor the performance of the deployed network infrastructure. Monitoring is a central activity in the design, engineering, and operation of a network. It consists of monitoring a network using a geographically distributed set of monitoring stations, with the purpose of using the monitoring data to serve several purposes such as traffic engineering, network resource provisioning and management, accounting and anomaly detection. The overall should lead to better service to end users and better revenues for the network operator. Increasing monitoring capabilities, along with the associated increased understanding of the network and user behavior, present the most challenging issue and an enormous undertaking in a large network.

This challenge becomes increasingly difficult to meet with the evolution of the Internet in terms of importance and size. The latter has been accompanied by an increasing heterogeneity of users' behavior and the development, growth, and use of a wide variety of network applications.

The objective of this dissertation is to design and validate a new network-wide monitoring system that provides better visibility over the entire network and adapts its configuration according to network condition changes in order to improve measurement accuracies and respect monitoring constraints (e.g. volume of collected measurements).

The main contributions of this thesis are summarized as follows:

- We have carried out a critical survey of the existing monitoring tools. We have presented some works and discussed their advantages and shortcomings. In fact, until recently, the usage data provided by network elements (e.g. routers) has been coarse-grained, typically comprising aggregate byte and packet counts in each direction at a given interface, aggregated over time windows of a few minutes. It has been successfully used in many applications, like accounting with acceptable accuracy. However, these data are no longer sufficient to engineer and manage networks that know a remarkable growth. In order to preserve the accuracy and provide accurate estimations, monitoring solutions should cope with the increasing trend in line speed and the tremendous success of new emerged applications. In this thesis we provide a critical survey of the different proposed solutions and we discuss their performances.

- One of the major issues we faced during this PhD was the experimentation of the proposed monitoring approaches. In fact, although many experimental solutions exist today, they tend to be highly specialized, or to have a limited availability and openness. We present our emulation platform [68, 8] for network wide traffic monitoring as an answer to these limitations. Additionally, having a real-time control on real measurement points or getting collected traces from these points together with the related network topology are most of time not satisfied by ISP(s) for privacy and security reasons. Thus and as an intermediate solution, researchers make use of either network emulators coupled with synthetic traffic generators [2] or simply limit their research to parsing and studying link-level traffic traces collected on specific links. We introduce a new approach for the emulation of Internet traffic and for its monitoring across the different routers. We put at the disposal of users a real traffic emulation service coupled to a set of libraries and tools capable of Cisco NetFlow data export and collection, which are meant to run advanced applications for network wide monitoring and optimization. Furthermore, we provide an exhaustive analytical sensitivity analysis in order to characterize, qualitatively or quantitatively, what impact a particular input parameter has on a system output. In other words, using Sensitivity Analysis, one can determine how changes in one or several parameters will impact the target output variable.

- We introduce a network-wide cognitive monitoring system that profits from advances in machine learning techniques and flow-level monitoring tools [69, 61, 70]. The system starts from the NetFlow monitoring capabilities deployed in routers, tunes them in an adaptive and optimal way, and combines their results to answer the best the monitoring application needs. The proposed system adopts an adaptive centralized architecture that provides visibility over the entire network through a network-wide cognitive monitoring

system. Our system should adjust its configuration according to network conditions, measurements accuracy and monitoring constraints. Given a set of measurement tasks and a monitoring constraint, our network-wide system is able to: *(i)* Use existing local monitoring tools (i.e., NetFlow) deployed in routers inside the network to collect and obtain data. *(ii)* Support a central cognitive component that investigates the local measurements collected from the different routers, correlates them in order to construct a global view of the traffic and the network state. This component also measures the amount of overhead (defined as the total number of flow reports that are exported in the entire network, modelling processing and storage resources) caused by the actual configuration of monitors. *(iii)* Drive its own deployment by automatically and periodically reconfiguring the different monitors in a way that improves the overall accuracy (according to monitoring application requirements) and reduces the resulting overhead (respecting some resource consumption constraints). The automation of the control of sampling rates is achieved by learning experiences from the accuracy of the collected data and the resulting overhead. *(iv)* Investigate the multi-task and multi-objective techniques in order to support a general class of monitoring applications.

- We provide a *reactive* network reconfiguration method [61]. Our method proceeds in optimizing the configuration in small steps based on dynamics inspired from the one used by TCP for the adjustment of its congestion window. We use the **G**radient **P**rojection **M**ethod (GPM) to identify the monitors that we should reconfigure until the optimal configuration is reached.

- We introduce a *proactive* network reconfiguration method that adjusts its configuration according to network conditions and measurement accuracy [69]. This method relies on *(i)* an overhead prediction based on an **E**xponential **W**eighted **M**oving **A**verage (EWMA) filter to track sustainable changes while removing unnecessary variations in the traffic, *(ii)* a global weighted utility function to deal with multiple monitoring tasks at the same time, and *(iii)* an optimization algorithm that configures monitors to address the tradeoff between resource consumption and accuracy of different tasks.

- We provide a critical discussion and an experimental comparison of the *reactive* and the *proactive* network reconfiguration methods. We study the performance of these optimization methods in terms of measurement accuracy, resulting overhead and responsiveness to changes in the network traffic.

- We extend the presentation of our system using two sampling primitives, packet sampling and flow sampling [70]. We prove the ability of our system 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.

1.3 Thesis Outline

We have commenced our research work by investigating the existing architectures and mechanisms. Thus, the following chapter (i.e., Chapter 2) is a survey of the projects and approaches related to our work, with a detailed review of the most important techniques. Moreover, we compare these approaches from different points of view.

Given the lack of a universal experimental platform for monitoring applications, we present, in Chapter 3, our own platform together with an exhaustive experimental methodology for the evaluation of network-wide monitoring tools and systems. We have implemented a real experimental platform for traffic sampling and monitoring using real traffic traces and real monitoring tools [8]. We have also carried out a global study of the performance of monitoring systems and the impact of the different parameters on their behavior.

Initiated in Chapter 2, the original work of the present thesis has been completed throughout Chapters 4, 5 and 6. Chapter 4 defines a novel monitoring system architecture that adopts a centralized approach and drives its own configuration in order to address the tradeoff between monitoring constraints and measurement task requirements. We detail each of the constituent modules and their interfaces, and we present the various kinds of decision algorithms which interact and work together for providing better monitoring solution.

Furthermore, two different network reconfiguration methods are presented in Chapter 5. In fact, monitor configuration and reconfiguration is a crucial issue within the next-generation of monitoring systems. We present the *reactive* and the *proactive* network configuration methods, we validate their operation and we study their performance. Moreover, we provide an experimental comparison of these two methods as well as a discussion about their performance and the use case of each of them.

Then, in chapter 6, we extend the presentation of our system using two sampling primitives, packet sampling and flow sampling, in order to satisfy multiple monitoring tasks. We show that combining different sampling primitives improves the global accuracy of the different tasks.

Finally, the thesis concludes with Chapter 7 in which we review the main findings of our work and suggest some topics for further research.

2

STATE OF THE ART

Traffic measurement and analysis are important management activities for network operators in order to determine the composition of the network traffic, to understand the behavior of users, and to monitor the performance of the deployed network infrastructure. This can serve several purposes such as traffic engineering, network resource provisioning and management, accounting and anomaly detection. The overall should lead to better service to end users and better revenues for the network operator. The monitoring of network traffic is often done by passive measurements at the interfaces of some or all routers. Packets are captured and aggregated, then reported to a central collector for storage and analysis. Currently, there is an increasing interest in efficient passive measurement solutions that scale with the network traffic while providing the maximum possible information. Existing solutions often balance between number of monitored routers, volume of captured packets and aggregation level to meet application requirements while limiting the overhead on both routers and network links. Most of them treat routers independently of each other and function by static configuration, while few correlate the information collected from different routers to further increase the accuracy of measurements. A large part of these works have focused on coarse-grained measurements where the content of packets is disregarded. For instance, Simple Network Management Protocol (SNMP) [73] collects coarse information on network traffic as the packet rate and the burstiness and requires a low cost while packet-level solutions like [35] look at the content of all packets, and hence suffer from scalability problems given the large of volume of information they have to process and store. Currently, NetFlow [30, 32, 79] constitutes the most popular tool for network monitoring at the flow level. While the deployment of NetFlow lowers the cost of processing and storage resources, it clearly reduces the accuracy of measurements and entails a loss of information. This is unfortunately unavoidable given the actual disagreement between

the increasing speed of links and the router resource constraints. The main consequence of this trend is a fundamental disagreement between the existing monitoring capabilities and the increasing requirements of the network management applications in terms of accurate measurements. The solution to this discrepancy has certainly to pass by designing a novel architecture for network traffic monitoring. This architecture should multiply the monitoring points inside the network, couple their observations and find the best network-wide configuration (the set of sampling rates in network monitors) as a function of the monitoring needs. Moreover, one has to adapt the configuration on the fly if he wants the monitoring to adapt to changing network conditions. That should lead to better performances than in the case of static uncorrelated configuration. Some recent proposals take this previous direction and try to provide a widespread monitoring infrastructure that coordinates monitoring responsibilities and distribute the work between the different monitors e.g. [48, 86, 28]. 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.

To fulfil these basic requirements we have proposed a new network-wide cognitive monitoring system that benefits from advances in machine learning techniques and flow-level monitoring tools. The system starts from the NetFlow monitoring capabilities deployed in routers, tunes them in an adaptive and optimal way, and combines their results to answer to the best the monitoring application needs.

Nonetheless, we will first proceed, In this chapter, with a brief definition for the most important monitoring tools and techniques. We will present the history of network monitoring and the main monitoring tools. We will discuss the advantages and the shortcomings of the different tools. Moreover, we will present the chronology of the different tools as well as the monitoring requirements that led to their appearance. We will start by presenting the SNMP and the main monitoring applications achieved using SNMP (i.e. traffic matrix, anomaly detection).

Contents

2.1 SNMP: Overview and utilization

As mentioned above, SNMP is one of the most used monitoring tools. It is a simple solution deployed with a very low cost in terms of router processing resources. However, it gives only a rough idea about the network traffic and fails to provide details about its composition. In this section we will present SNMP and we will present some related works. Moreover, We will discuss the shortcomings of this tool.

2.1.1 SNMP counters

SNMP counters [73] are a very basic and widely available traffic measurement mechanism available in all routers today. Among other things, they count the number of packets and bytes transmitted on each of the links of a router. There are many traffic measurement applications that use them. The most common one is **M**ulti **R**outer **T**raffic **G**rapher (MRTG) [91]: it collects the values of the counters, and produces web pages with time series plots of the traffic. Another important use of these counters is in billing: some ISPs charge their clients based on the total traffic they send and receive on their link to the ISP. SNMP is the network management protocol proposed by **I**nternet **E**ngineering **T**ask **F**orce (IETF) for managing devices on IP networks. Devices that typically support SNMP include routers, switches, servers, workstations, printers, modem racks. SNMP is used to monitor and manage network-attached devices. These devices called managed systems continuously execute a software component called agent. The agent collects and repots information via SNMP to a central device called manager. SNMP consists of a set of standards for network management, including an application layer protocol, a database schema, and a set of data objects. Essentially, SNMP agents expose management data on the managed systems as variables. The protocol also permits active management tasks, such as modifying and applying a new configuration through remote modification of these variables. The variables accessible via SNMP are organized in hierarchies. These hierarchies, and other metadata (such as type and description of the variable), are described by **M**anagement **I**nformation **B**ases (MIBs). The SNMP management environment consists of several key components: the monitoring station called manager, the data objects of the network, the variables MIB and a protocol. The various components of SNMP are:

- The active elements of the network equipment or software which is a network node that implements an SNMP interface in order to exchange information with the manger. These nodes range from a workstation to a hub, router, bridge. Each network element has an entity called agent that responds to requests from the monitoring station. Agents are network management modules that reside on network elements. They will seek management information such as the number of packets received or transmitted. These agents have local knowledge of management information and translate that to or from an SNMP specific form.

- The monitoring station (also called manager) executes management applications that monitor and control network elements. Physically, the station is a workstation. One or more managers can exist in a managed network.

- The MIB (Management Information Base) is a collection of objects residing in a virtual database. These collections of objects are defined in specific MIB modules. MIBs describe

the structure of the management data of a device subsystem, they use a hierarchical namespace containing object identifiers (OID). Each OID identifies a variable that can be read or set via SNMP. The MIB is a tree structure where each node is defined by the OID.

- The protocol, which allows the monitoring station to get information about network elements and receive alerts from these same elements.

SNMP is based on an asymmetric operation. It consists of a set of queries, answers and a limited number of alerts. The manager sends requests to the agent, which returns answers. When an abnormal event arises on the network element, the agent sends an alert to the manager. SNMP uses the UDP protocol. The port 161 is used by the agent to receive requests from the management station. The port 162 is reserved for the management station to receive alerts agents. While the information offered by the SNMP counters is useful, it omits a lot of details. They do not say what type of traffic is using the links. More generally, for network planning, ISPs need to know the traffic matrix: the traffic between pairs of clients or pairs of traffic aggregation locations for the network (POPs). There are many approaches for reconstructing the traffic matrix based on SNMP counters and network topology [74] and they are collectively known as network tomography. One disadvantage of these methods is that they can only give approximate traffic matrices.

In the next section we will present some works that try to identify and classify anomalies using the SNMP counters. Furthermore, we will show some drawbacks of using SNMP counters such as the approximate and inaccurate results we get and the difficulty of achieving this task.

2.1.2 Identification and classification of anomalies in network-wide traffic

The principle objective of this work is to conceive a traffic analysis method that detects and identifies a large and diverse set of anomalies. However, mapping an anomalous subspace traces to one or more responsible flows is challenging since they worked on aggregated measurements data.

Lakhina et al. [19, 18, 20, 97, 96] have proposed a detector based on principal component analysis and some heuristics in order to detect and classify network attacks. They have used the principal component analysis in order to detect anomalous time bins. Once done, they provide a direct mapping between anomalous subspaces (anomalous time bins) and the original flows (i.e., anomalous flows). They have applied heuristics that associate an anomaly with the flows with the largest contribution to the anomalous subspace. In previous works [19, 18, 20], they have detected network-wide anomalies when analyzing the flows aggregation, and could detect a wide variety of types of anomalies when analyzing entropy timeseries of IP header features. They have also recently used sketches [97] and distributed monitors [96] to provide more efficient traffic anomaly detection.

For instance, in [19] they have proposed a general method to diagnose anomalies. This method is based on a separation of the high-dimensional space occupied by a set of network traffic measurements into disjoint subspaces corresponding to normal and anomalous network conditions. They have demonstrated that this separation can be performed effectively by principal component analysis. They have studied volume anomalies and have shown that the method can: *(i)* accurately detect when a volume anomaly is occurring, *(ii)* correctly identify the underlying origin-destination (OD) flow which is the source of the anomaly, and *(iii)* accurately estimate the amount of traffic involved in the anomalous OD flow. They have addressed the anomaly diagnosis problem in three steps: first, they use a general method to detect anomalies in network traffic, and then they employ distinct methods to identify and quantify them.

- The detection problem consists of designating those points in time at which the network is experiencing an anomaly. An effective algorithm for solving the detection problem should have a high detection probability and a low false alarm probability.
- The identification problem consists of selecting the true anomaly type from a set of possible candidate anomalies. The method they proposed is extensible to a wide variety of anomalies.
- The quantification problem consists of estimating the number of additional or missing bytes in the underlying traffic flows. Quantification is important because it gives a measure of the importance of the anomaly.

For a successful diagnosis of a volume anomaly, they have argued that it is important to detect the time of the anomaly, identify the underlying responsible flow, and quantify the size of the anomaly within that flow. The problem here is, if a timeserie is classified as anomalous, it is important to determine precisely which set of flows of the traffic matrix were primarily responsible for the detection. Knowing that an anomaly occurred at a particular time is typically not sufficient. We need to know where it happened to identify and classify this anomaly. For example, the network operator may need to know which ingress routers were the entry points for the anomalous traffic. It is important to realize that this flow identification step is a necessary step in the context of network traffic anomaly detection.

They have demonstrated in [20], that the distributions of packet features (IP addresses and ports) observed in flow traces reveals both the presence and the structure of a wide range of anomalies. Using entropy as a summarization tool, they have shown that the analysis of feature distributions leads to significant advances on two fronts: (1) it enables highly sensitive detection of a wide range of anomalies, augmenting detections by volume-based methods, and (2) it enables automatic classification of anomalies via unsupervised learning. In fact, using feature distributions, anomalies naturally fall into distinct and meaningful clusters. These clusters can be used to automatically classify anomalies and to uncover new anomaly types.

Table 2.1: Qualitative effects on feature distributions by various anomalies.

Anomaly label	Definition	Traffic feature distributions affected
Alpha flows	Unusually large volume point to point flow	Source address, destination address (possibly ports)
DOS	Denial of service attack (distributed or single-source)	Destination address, source address
Flash crowd	Unusual burst of traffic to single destination, ”typical” distribution of sources	Destination address, destination port
Port scan	Probes to many destination ports on a small set of destination addresses	Destination address, destination port
Network scan	Probes to many destination addresses on a small set of destination ports	Destination address, destination port
Outage events	Traffic shifts due to equipment failures or maintenance	Mainly source and destination address
Point to multipoint	Traffic from single source to many destinations, e.g., content distribution	Source address, destination address
Worms	Scanning by worms for vulnerable hosts (special case of network scan)	Destination address and port

As shown in Table 2.1, most traffic anomalies share a common characteristic: they induce a change in distributional aspects of packet header fields (i.e., source and destination addresses and ports, for brevity in what follows, these are called traffic features). For example, a DOS attack, regardless of its volume, will cause the distribution of traffic by destination addresses to be concentrated on the victim address. Similarly, a scan for a vulnerable port (network scan) will have a dispersed distribution for destination addresses, and a skewed distribution for destination ports that is concentrated on the vulnerable port being scanned. Even anomalies such as worms might be detectable as a change in the distributional aspect of traffic features. Table 2.1 lists a set of anomalies commonly encountered in network traffic. Each of these anomalies affects the distribution of certain traffic features.

Therefore, they have argued that entropy is an effective metric for anomaly identification and classification since it captures the distributional changes in traffic features, and observing the time series of entropy on multiple features exposes unusual traffic behavior. For instance, they have demonstrated that entropy timeseries of the four main IP header features (source IP, destination IP, source port, and destination port) is a rich data type to analyze for traffic

anomaly detection. That is, for every measurement there are four measurements. Entropy is studied because it provides a computationally efficient metric for estimating the dispersion or concentration in a distribution, and a wide variety of anomalies will impact the distribution of one of the discussed IP features. For example, the probability of seeing port 80 is defined to be number of packets using port 80 divided by the total number of packets in the given time interval. A sudden flash crowd to a webserver, for example, will cause a specific destination IP (the webserver) and destination port (port 80) to become much more prevalent than in previous time-steps, which will cause a decrease in the destination IP and destination port entropy timeseries, respectively, and hence allow us to detect it.

Lakhina et al. were able to achieve such promising early results because of their great familiarity with both the technique and the data. They have proposed an approach called the subspace method to diagnose network-wide traffic anomalies. The method can detect, identify and quantify traffic anomalies. The subspace method uses principal component analysis to separate network traffic into a normal component that is dominated by predictable traffic, and an anomalous component which is noisier and contains the significant traffic spikes. Moreover, they have detected network-wide anomalies when analyzing the OD flow aggregation, and could detect a wide variety of types of anomalies when analyzing entropy timeseries of IP header features. They have also recently used sketches [97] and distributed monitors [96] to provide more efficient traffic anomaly detection.

In the previous section, we have seen one of the most useful monitoring tools (i.e. SNMP). SNMP is a simple solution that collects coarse information on network traffic. We have presented some works that used SNMP counters in order to achieve important monitoring applications (e.g. traffic matrix estimation, anomaly detection). However, SNMP counters are not sufficient to respond to the increasing requirements of monitoring applications and suffer from approximate and inaccurate measurements. Hence, many research works have been done in order to provide a better monitoring systems and techniques.

In the next section we will present the most popular monitoring tool which is Netflow. We will give an exhaustive overview and we will present some extended works of Netflow. Moreover we will discuss the advantages and shortcomings of this widely deployed monitoring solution.

2.2 NetFlow overview

The interest in passive monitoring for the understanding and diagnosis of core IP networks has grown at an astonishing rate. Recently, numerous works have focused on the design of new monitoring techniques and on the analysis of the collected results. The spectrum is large covering simple solutions like SNMP [73] which is a simple solution deployed with a very low

cost in terms of router processing resources. But, it gives only a rough idea about the network traffic and fails to provide details about its composition. To packet-level solutions, at the other extreme, that examine finer details of traffic and allow extremely accurate measurements [35]. However, these solutions have scalability problems since they capture every packet on a link. To address this latter drawback, recent works have demonstrated the benefits of traffic sampling and aggregation for reducing the amount of collected data and improving the overall accuracy, e.g. [100, 42, 41]. NetFlow has evolved into a solution that satisfies this by reporting flow records that summarize a sample of the traffic [30, 32, 79, 49]. Packets are sampled at the packet level of flow level, then aggregated into records, before being sent back to a central collector for later analysis. Currently, NetFlow is the most widely deployed measurement solution. However, this solution still presents some shortcomings, namely the problem of reconfiguring sampling rates according to network changes and the requirements of monitoring applications. Another problem comes from the low values to which the sampling rate is set in practice (between 0.01 and 0.001) to be able to cope with the increasing trend in line speed. This in its turn is known to introduce serious bias against small flows and fine-grained measurements [48].

Many efforts have explored ways to improve NetFlow. For instance, Estan et al. present in [49] an adaptive NetFlow that dynamically adapts the sampling rate in each router to changing traffic conditions for the purpose of limiting the local resource consumption. Duffield et al. present in [47] means for dynamic control of sample volumes to minimize the overall amount of measurement traffic while reducing the variance of the estimated sum of the measurements.

Next, we will first proceed with a brief presentation of Netflow.

2.2.1 NetFlow background

NetFlow [30, 32, 79], first implemented in Cisco routers, is the most widely used flow measurement solution today. Routers maintain flow records collecting various bits of information. Flows are identified by fields present in the header of every packet: source and destination IP address, protocol, source and destination port, and type of service bits. The flow record keeps information such as the number of packets in the flow, the (total) number of bytes in those packets, the timestamp of the first and last packet, and protocol flag information such as whether any of those packets had the SYN flag set. NetFlow uses four rules to decide when to remove a flow record from router memory and report it to the collection station: *(i)* when TCP flags (FIN or RST) indicate flow termination, *(ii)* 15 seconds configurable ”inactive timeout” after seeing the last packet with a matching flow ID, *iii* 30 minutes (configurable ”active timeout”) after the record was created to avoid staleness and *iv* when the memory is full. On every new packet, NetFlow looks up the corresponding entry (creating a new entry if necessary) and updates that entry’s counters and timestamps. NetFlow [30, 32, 79] is widely deployed and constitutes the most popular tool for network monitoring at the flow level, with the flow

definition ranging from 5-tuple to entire network prefixes. NetFlow allows a ubiquitous and cheap deployment of monitors (it is integrated in routers), provides visibility over the entire network. Moreover, it addresses performance limits by aggregating packets belonging to the same flow in one record. Netflow records contain several information on the flow as its size in bytes and packets and its source and destination addresses. NetFlow misses however information on packet arrivals within a given flow, but for most management applications this is not a serious problem. NetFlow is usually deployed in edge routers. This choice is done to ease the deployment since packets are seen once and so flows can be easily formed at the collector. NetFlow still has problems in case of high speed links since it has to capture all packets and update flow records accordingly. To cope with the high rate of packets, NetFlow has a sampled variant where packets are first sampled at the physical layer with some rate $p \leq 1$, then flows are formed with the stream of sampled packets. This sampling is an efficient mechanism to reduce the overhead of NetFlow, it is known however to introduce errors and be biased against flows of small sizes (they may even disappear). Of our days, network operators configure the sampling rate in their NetFlow monitors at low static and predefined rates (in general between 0.01 and 0.001) and infer the volume of original flows by dividing the reported NetFlow flow volumes by the sampling rate p (known to be the most likelihood estimator).

While the deployment of NetFlow lowers the cost of processing and storage resources, it clearly reduces the accuracy of measurements and entails a loss of information. This is unfortunately unavoidable given the actual disagreement between the increasing speed of links and the router resource constraints. The main consequence of this trend is a fundamental disconnect between the existing monitoring capabilities and the increasing requirements of the network management applications in terms of accurate measurements. Many efforts have explored ways to improve NetFlow. For instance, Estan et al. present in [49] an adaptive NetFlow that dynamically adapts the sampling rate in each router to changing traffic conditions for the purpose of limiting the local resource consumption. Duffield et al. present in [47] means for dynamic control of sample volumes to minimize the overall amount of measurement traffic while reducing the variance of the estimated sum of the measurements.

Next, we will present these two most important extended works of Netflow. We will start first by presenting the adaptive Netflow solution.

2.2.2 Adaptive NetFlow

Sampled flow level measurement provides a balance between scalability and detail because performance limits can be addressed by reducing the sampling rate. Thus, most major ISPs rely on NetFlow data to provide input to traffic analysis tools that are widely used by network operators. However, sampled NetFlow has shortcomings that hinder the collection and analysis of traffic data. First, during flooding attacks router memory and network bandwidth consumed by

Table 2.2: Summary of NetFlow problems and proposed solutions.

Problem	Solution	Requirement
Number of records strongly depends on traffic mix	Adapting sampling rate to traffic	Software update
Network operator must set sampling rate		
Mismatch between flow termination heuristics and analysis	Measurement bins	Software update
Cannot estimate the number of flows	Sampling flows	Hardware addition

flow records can increase beyond what is available. Second, selecting the right static sampling rate is difficult because no single rate gives the right tradeoff of memory use versus accuracy for all traffic mixes. Third, the heuristics routers use to decide when a flow is reported are a poor match to most applications that work with time bins. Finally, it is impossible to estimate without bias the number of active flows for aggregates with non-TCP traffic.

In order to solve these shortcomings, the authors in [49] have proposed an improved version of NetFlow called *adaptive NetFlow*, deployable through an update to router software, which addresses many shortcomings of NetFlow by dynamically adapting the sampling rate to achieve robustness without sacrificing accuracy as shown in table 2.2.

The main contributions of this work are:

- Sampling rate adaptation: the adaptive solution adapts sampling rates to stay within mixed resource consumption limits while using the optimal sampling rate for all traffic mixes.
- Renormalization of flow entries: renormalization allows reducing the number of NetFlow entries after a decrease in sampling rate. Renormalization also enables a layered transmission of NetFlow data that gracefully degrades the accuracy of traffic reports in response to network congestion on the reporting path.
- Time bins: most traffic analysis tools divide the traffic stream into fixed intervals of time that we call bins. Unfortunately, NetFlow records can span bins, causing unnecessary complexity and inaccuracy for traffic analysis. our Adaptive NetFlow, by contrast, ensures that flow records do not span bins. This simple idea is essential in order to provide statistical guarantees of accuracy after operations such as renormalization and sampling rate adaptation.
- Accurate flow counting: they provide a flow counting extension to give accurate flow counts even for non-TCP flows.

This work aims to improve NetFlow and solve its problems. The main problem of NetFlow is that the memory required by the flow records and the bandwidth consumed to report them

Table 2.3: Comparison of *sampled* NetFlow and *adaptive* NetFlow.

Issue	*sampled* NetFlow	*adaptive* NetFlow
Memory usage	Variable	Fixed
Volume of flow data reported	Variable	Fixed
Behavior under DDoS with spoofed sources and other traffic mixes with many flows	Panicky flow expiration	Reduction in accuracy
Estimates of traffic in small time bins	Less accurate	Accurate
Reporting overhead when using small bins	Unaffected	Large increase
Lifetime of flow record in router memory	Min (active timeout, flow length + inactivity timeout)	Bin length
Resource usage at end of time bin	N/A	Reporting spike or extra memory
Processing intensive tasks	Counting	Counting and renormalization
Counting TCP flow arrivals (using SYNs)	Yes	Yes
Counting all active flows	No	Separate flow counting extension
Counting all active flows at high speeds	No	Hardware flow counting extension

depends strongly on the traffic mix. In particular, large floods of small packets with randomly spoofed source addresses can increase memory and bandwidth requirements by orders of magnitude. Adaptive NetFlow [49] solves this problem by dynamically adapting the sampling rate. Adaptive NetFlow divides the operation of the flow measurement algorithm into equally spaced time bins. Within each bin, the algorithm starts by sampling aggressively (high sampling probability). If memory is consumed too quickly, it switches to less aggressive sampling. It then ”renormalizes” existing entries so that they reflect the counts they would have had with the new sampling rate in effect from the beginning of the bin. At the end of the bin, all entries are reported. Using fixed size bins in *adaptive* NetFlow increases the memory utilization compared to *sampled* NetFlow and causes bursts in reporting bandwidth. Table 2.3 gives a summary comparison of *sampled* NetFlow and *adaptive* NetFlow.

In the previous section we have presented the adaptive NetFlow work. Now we move to presenting another extended work for Netflow. This work provides means for dynamic control of sample volumes to minimize the overall amount of measurement traffic while reducing the variance of the estimated sum of the measurements.

2.2.3 Control of volume and variance in network measurement

The authors in [47] have studied how the sampling distribution should be chosen in order to jointly control both the variance of the estimators and the number of samples taken? They have introduced the threshold sampling as a sampling scheme that optimally controls the expected volume of samples and the variance of estimators over any classification of flows. They have provided algorithms for dynamic control of sample volumes and they have evaluated them on flow data gathered from a commercial IP network.

The main contribution of their work is to study what makes a good flow sampling strategy. The flows are represented by the flow records exported by the routers. The goals are fourfold: *(i)* to constrain samples to within a given volume, *(ii)* to minimize the variance of usage estimates arising from the sampling itself, *(iii)* to bind the sampling parameters to the data in order that usage can be estimated transparently, and *(iv)* with progressive resampling, the composite sampling procedure should enjoy properties *(i)—(iii)*.

In order to achieve these goals, they proposed to continuously stratify the sampling scheme so the probability that a flow record is selected depends on its size. This attaches more weight to larger flows whose omission could skew the total size estimate, and so reduce the impact of heavy tails on variance. They renormalized the sampled sizes in order that their total over any key set becomes an unbiased estimator. This is achieved by coordinating the renormalization with the sampling probabilities. The sampling scheme is optimized by minimizing a cost function that expresses the undesirability of having either a large number of samples, or a large variance in estimates formed from them. Sampling with this optimal choice of sampling probabilities will be characterized by a certain threshold called threshold sampling. Finally, they used a mechanism to tune the sampling probabilities in order that the volume of sampled records can be controlled to a desired level, even in the presence of temporal or spatial inhomogeneity in the offered load of flows.

As illustrated above, NetFlow has problems in case of high speed links since it has to capture all packets and update flow records accordingly [49]. To cope with the high rate of packets, NetFlow uses sampling mechanism. While this sampling is an efficient mechanism to reduce the overhead of NetFlow, it is known however to introduce errors and be biased against flows of small sizes. Many sampling techniques have been proposed in order to provide more accurate measurements while reducing the amount of collected data.

In the next section, we will give an exhaustive survey of the different sampling techniques.

2.3 Metering and exporting processes

A metering process selects packets from the observed packet stream using a selection process, and produces as output a report stream concerning the selected packets. An exporting process sends, in the form of export packets, the output of one or more metering processes to one or more collectors.

2.3.1 Techniques for IP packet selection

Confronted with the increasing trend in network data rates compounded by the demand of measurement-based applications for increasingly fine grained traffic measurements, monitors have a serious problem of resource consumption resulting from the huge amount of measurement data. To cope with this problem, monitors require an intelligent packet selection scheme. We describe in this section, *sampling* and *filtering* techniques for IP packet selection. We will provide a categorization of schemes and defines what parameters are needed to describe the most common selection schemes

2.3.1.1 Categorization of packet selection techniques

The objective of using packet selection techniques is to generate a subset of packets from an observed packet stream in order to reduce the amount of collected and transmitted data. We distinguish two main selection techniques: *sampling* and *filtering*.

- Sampling: This is used in order to select a representative subset of packets. Once done we use this subset of collected data in order to estimate and infer properties of whole traffic. The selection can depend on packet position, and/or on packet content, and/or on (pseudo) random decisions.
- Filtering selects a subset with common properties. This is used if only a subset of packets is of interest. Filtering is a deterministic operation. It depends on packet content or router treatment is used to remove all packets that are not of interest.

Note that a common technique to select packets is to compute a hash function on some bits of the packet header and/or content and to select the packet if the hash value falls in the hash selection range. It is a filtering technique since hashing is a deterministic operation on the packet content. Nevertheless, hash functions are sometimes used to emulate random sampling. Depending on the chosen input bits, the hash function, and the hash selection range, this technique can be used to emulate the random selection of packets with a given probability p. It is also a powerful technique to consistently select the same packet subset at multiple observation points.

Table 2.4: Categorization of packet selection schemes.

Selection scheme	Deterministic selection	Content-dependent	Category
Systematic count-based	√	-	Sampling
Systematic time-based	√	-	Sampling
Random n-out-of-N	-	-	Sampling
Random uniform probabilistic	-	-	Sampling
Random non-uniform probabilistic	-	√	Sampling
Random non-uniform flow-state	-	√	Sampling
Property match filtering	√	√	Filtering
Hash function	√	√	Filtering

Table 2.4 gives an overview of the packet selection schemes and their categorization. For instance, property match filtering is typically based on packet content and therefore is content dependent. But it may also depend on router state and then would be independent of the content. It easily can be seen that only schemes with both properties are considered as filters [101].

2.3.1.2 Sampling

The deployment of sampling techniques aims at controlling the resources consumed by measurements while collecting information about a specific characteristic of the network traffic. Sampling techniques reduce and lower the cost of monitoring. In order to jointly control both the variance of estimates and the resulting overhead in terms of the number of samples taken, we need to determine the type of information or the metric to estimate and the desired degree of accuracy and/or the monitoring constraint. This can lead to plan the suitable sampling strategy. Sampling parameters and methods used for selection must be known in the estimation procedure, in order to ensure a correct interpretation of the measured results. It is important to know the type of metric that should be estimated. For example, in order to estimate the number of packets from samples gathered using *1 in N* sampling, we multiply the number of

sampled packets by N. in the same manner, to estimate the byte rate in a raw packet stream, we need just to multiply the byte rate represented in collected data by N.

We characterize sampling methods depending on the the sampling algorithm [101]. The sampling algorithm describes the basic process for selection of samples. In accordance to [23] and [31], we devide the basic sampling processes into the following subtypes.

Systematic sampling: it is a statistical method involving the selection of elements from an ordered sampling frame. It describes the process of selecting the start points and the duration of the selection intervals according to a deterministic function. This can be the selection of all packets that arrive at per-defined points in time but the most common form of systematic sampling is an equal-probability method, in which every k^{th} element in the frame is selected, where k is the sampling interval. Using this procedure, each element in the population has a known and equal probability of selection. This makes systematic sampling functionally similar to simple random sampling. However, the use of systematic sampling involves the risk of biasing results. Only equally spaced schemes are considered, where triggers for sampling are periodic, either in time or in packet count. All packets occurring in a selection interval (either in time or packet count) beyond the trigger are selected. Both schemes are content-independent selection schemes. Content-dependent deterministic selectors are categorized as filters.

Random Sampling: it refers to taking a number of independent observations from the same probability distribution. Random sampling consists on selecting a subset of data in accordance to a random process. The selection of elements must be independent. This minimizes bias and simplifies analysis of results. The sample usually is not a representative of the population from which it was drawn, this random variation in the results is known as sampling error. In the case of random samples, mathematical theory is available to assess the sampling error. Thus, estimates obtained from random samples can be accompanied by measures of the uncertainty associated with the estimate. We distinguish two methods of random sampling:

- *Simple Random Sampling:* this technique consists on selecting n elements out of population that consists of N elements, hence it is sometimes called *n-out-of-N* sampling. Applied to network management systems, a smaller set of n packets is sampled from a given, larger set of N packets. However, selecting n consecutive packets with a count period of N may introduce a serious bias in measurements. One way to minimize biasing the results entails randomly selecting the position of the first packet to be sampled. Although this reduces the probability of biasing, it does not completely resolve the biasing problem. For this kind of sampling, the sample size is fixed. Examples of implementation include sFlow [11] and random sampled Netflow [9]. Both examples use *1-out-of-N* sampling.

- *The probabilistic sampling method:* it is a method of sampling that utilizes some form of random selection. In order to have a random selection method, one must set up some

process or procedure that assures that the different units in the population have equal probabilities of being chosen. For this kind of sampling, the sample size can vary for different trials. We differentiate two methods probabilistic sampling: *(i)* uniform probabilistic sampling that uses independent selection of packets with a uniform probability p, and *(ii)* non-uniform probabilistic sampling where the sampling probabilities can depend on the selection process input. This non-uniform approach can be used to weight sampling probabilities in order to boost the chance of sampling packets that are rare but are deemed important. More details about non-uniform sampling can be found in [101].

2.3.1.3 Filtering

Filtering is a deterministic selection of packets based on the packet content. We select a packet if its content or its quantity falls into a specified range. A filter is a selector that selects a packet deterministically based on the packet content, or its treatment, or functions of these occurring in the selection state and not on their position in time or in space or on random process. Examples include match filtering, and hash-based selection.

We distinguish two filtering techniques:

- Property match filtering: this technique is used to select packets according to specific fields. A packet is selected when these specific fields are equal to a predefined value. Typical fields that can be supported are: the IP header, transport protocol header and encapsulation headers.
- Hash-based filtering: a hash function is applied to the packet content. Then, the packet is selected if the result of this function is in a specified range.

In the previous section we have presented the different selection and collection techniques. Next, we will present the IPFIX standard that defines how IP flow information is to be formatted and transferred from an exporter to a collector.

2.3.2 IP flow information export (IPFIX)

Internet **P**rotocol **F**low **I**nformation **E**xport (IPFIX) is an IETF working group. It represents a universal standard of export for Internet protocol flow information from an IPFIX Device (e.g. routers probes) to a collector. It was ceated to fulfill the requirements of several applications to IP flows information export like those used by mediation systems, accounting/billing systems, and network management systems to facilitate services such as measurement, accounting, and billing. The IPFIX standard defines how IP flow information is to be formatted and transferred from an exporter to a collector. The IPFIX standards requirements were outlined in [80, 63, 81]. The working group chose Cisco Netflow version 9 as the basis for IPFIX.

Within the context of IPFIX and similar to the Netflow, a flow is defined as a set of IP packets passing an observation point in the network during a certain time interval. All packets belonging to a particular flow have a set of common properties, e.g. ”same source, same destination, same protocol” [81]. Using IPFIX, devices like routers can inform a central monitoring station about their view of a potentially larger network. IPFIX is a push protocol, i.e. each sender will periodically send IPFIX messages to configured receivers without any interaction by the receiver.

The major difference between IPFIX and PSAMP is that the IPFIX protocol exports flow records while the PSAMP protocol exports packet reports [63]. From a pure export point of view, IPFIX will not distinguish a flow record composed of several packets aggregated together from a flow record composed of a single packet. So the PSAMP export can be seen as a special IPFIX flow record containing information about a single packet. In fact, traffic flow measurement can be separated into two stages: packet processing and flow processing. Figure 2.1 illustrates these stages.

- In stage 1, here monitoring tools act on packets. Packets are captured, timestamped, selected by one or more selection steps, and finally classified and mapped to flows. The packets' selection steps may include filtering and sampling functions.

- In stage 2, here monitoring tools act on flows. After packets are classified (mapped to flows), flows are generated (or updated if they exist already). Flow generation and update steps may be performed repeatedly for aggregating flows. Finally, flows are exported.

Packet sampling covers only stage 1 of the IPFIX architecture with the packet classification replaced by packet report export, while IPFIX covers stage 2 also, as it generates flow records out of the selected packets.

In the previous parts we have presented the main selection and exporting techniques. We have given a brief definition for each of the most used techniques. Many extended works based on these selection and exporting techniques have been proposed. The main goal is to provide better methods for traffic selection and collection.

In the next section, we will give an exhaustive survey of the different proposed sampling methods.

2.3.3 Sampling primitives

Network traffic measurement is essential for traffic engineering (e.g. link upgrades or traffic rerouting) and traffic accounting (e.g. usage based pricing). Routers offer tools such as Cisco's Netflow [30, 32, 79] that give information about the flows of packets that traverse them. However the generation of detailed traffic statistics does not scale well with link speed. This is why

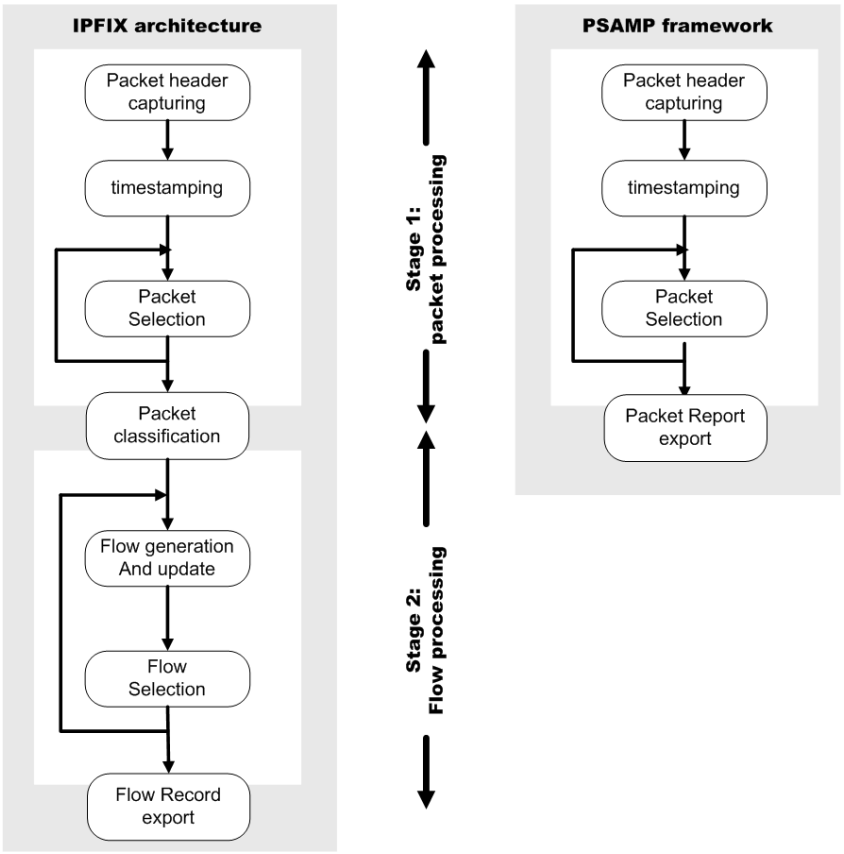


Figure 2.1: Comparison of IPFIX Architecture and PSAMP Framework.

sampling techniques are increasingly being important to export the statistics of a portion of the traffic only. We will study in this section different sampling primitives that respond to the requirements of the different monitoring applications that can be divided into two broad classes: *(i)* those that require an understanding of volume structure, e.g., heavy-hitter detection and traffic engineering that need an understanding of the number of packets/bytes per-port or per-src, and *(ii)* those that depend on the communication structure, e.g., security applications and anomaly detection application that require an understanding of "who-talks-to-whom" [93].

2.3.3.1 Packet sampling

Packet monitoring is an integral part of network measurements. In packet monitoring one is interested with the information contained within packets traversing a link.

Packet-based sampling schemes are widely used to characterize network traffic. Packet sampling uses randomness in the sampling process to prevent synchronization with any periodic patterns in the traffic. On average, 1 in every N packets is captured and analyzed. While this type of packet sampling does not provide a 100% accurate result, it does provide a result with quantifiable accuracy. We can quantify sampling error using mathematical theory. Packet sampling is suited to application that require an understanding of volume structure, e.g., heavy-hitter detection and traffic engineering that require an understanding of the number of packets/bytes per-port or per-src. Independent and identically distributed (i.i.d.) packet sampling consists of, for each packet in an independent manner, retaining the packet with probability p or discarding it with probability $1 - p$. Figure 2.2 presents the Packet sampling procedure. Routers select packets according to a sampling rate value p. Once done they aggregate the sampled packets into flow reports and send them to a collector. samples can be accompanied by measures of the uncertainty associated with the estimate given the sampling rate p. This approach allowed relatively easy monitoring of whole network traffic.

Packet sampling is a main technique for a wide range of network monitoring applications since it is a scalable alternative to achieve monitoring objectives. Packet sampling provides several information and statistics on network traffic using packet headers fields. This can serve several tasks such as traffic engineering and management, trouble shooting, capacity planning, accounting and usage profiling, and load balancing. The fundamental question regarding packet sampling is its accuracy: how accurate the different measurements done using packet sampling? This question is pertinent in the Internet environment and depends on sampling rate, application and periodicity of measured metric. Periodic events are more likely to be detected by packet sampling. Packet sampling is a scalable technique that provides accurate results for several application such as controlling congestion, detection of broken links, misconfigured devices [64]. This technique is used to meter and bill for network usage and provide a classification of top users and applications.

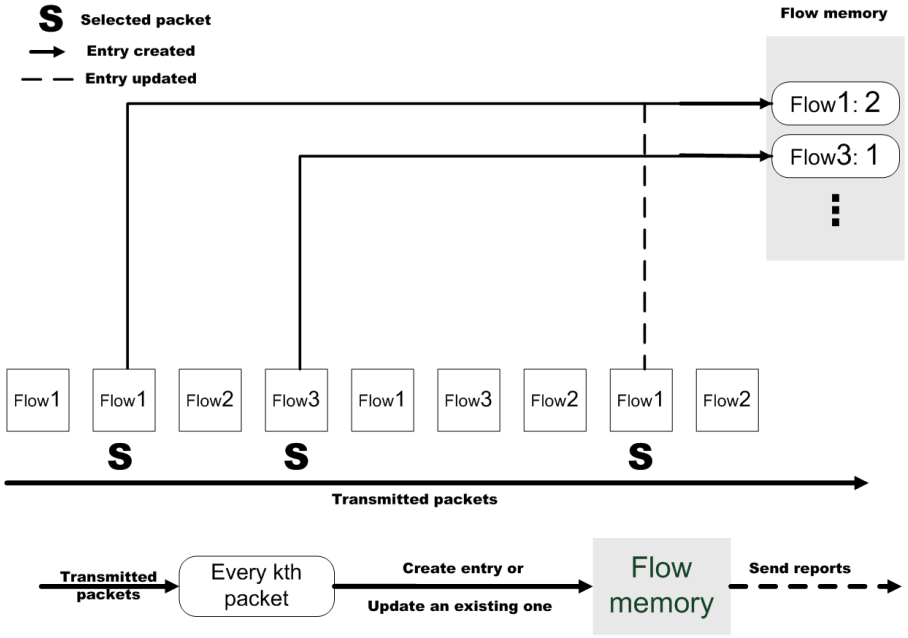


Figure 2.2: Packet sampling procedure.

2.3.3.2 Flow sampling

All the applications mentioned in the previous section can profit from packet sampling and still preserve a decent accuracy. However there are domains where packet sampling might introduce some significant errors. There are a lot of packet sampling applications in the domain of network security and anomaly detection. Examples include detections of protocol violations, user policy violations, and unauthorized access to an address from a blacklist. For this kind of anomaly sometimes only one packet is enough to raise the true alarm. For other well know network anomalies, violations and attacks, like DoS attack, P2P applications, portscans more sophisticated methods are needed in order to provide accurate results. These advanced methods include for instance a flow analysis. This new area of applications like anomaly detection needs flow sampling schemes suited to monitoring applications that depend on the communication structure, e.g., security applications and anomaly detection application that require an understanding of ”who-talks-to-whom” [93]. Flow sampling (FS) picks flows rather than packets at random [59]. As shown if figure 2.3 One way to implement FS is as follows. Each router has a sampling manifest, a table of one or more hash ranges indexed using a key derived from each packet header. On receiving a packet, the router computes the hash of the packet's 5-tuple (i.e., the flowkey). Next, it selects the appropriate hash range from the manifest and selects the flow if the hash falls within this range. If the flow is selected, then the router uses its hash as an index into a table of flows and updates the byte and packet counters for the flow. The hash

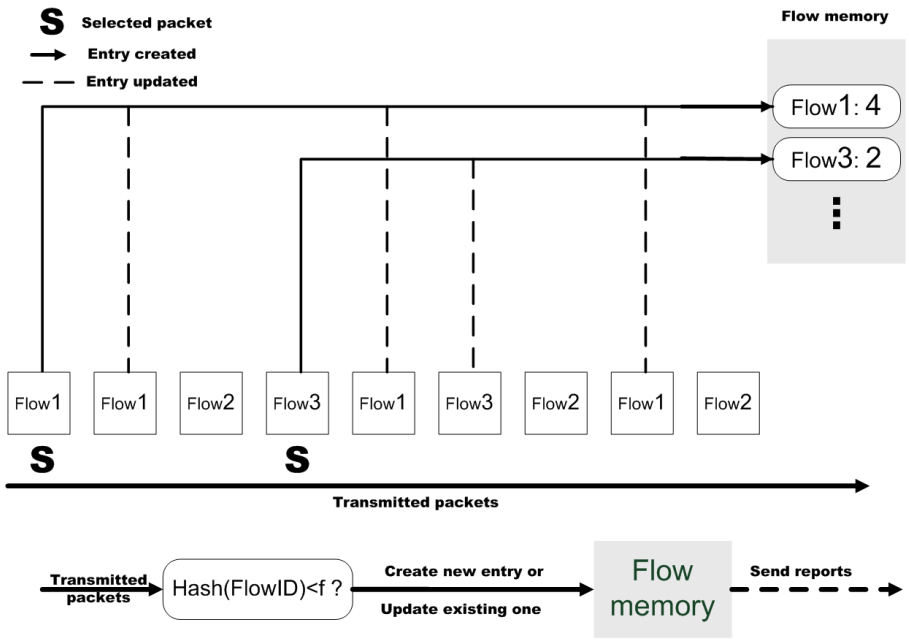


Figure 2.3: Flow sampling procedure.

function maps the input 5-tuple uniformly into the interval $[0, 1]$. Thus, the size of each hash range determines the flow sampling rate for each category of flows in the manifest. similarly to packet sampling, i.i.d. flow sampling consists of, for each flow independently, leaving the flow untouched with probability q or removing it entirely with probability $1 - q$.

2.3.3.3 Sample and hold

The idea is to sample each packet with a probability p. If a packet is sampled and the flow it belongs to has no entry in the flow memory, a new entry is created. After an entry is created for a flow, it updates the entry for every subsequent packet belonging to the flow as shown in Figure 2.4. Thus once a flow is sampled a corresponding counter is held in a hash table in flow memory till the end of the measurement interval. While this clearly requires processing (looking up the flow entry and updating a counter) for every packet, it ensures better accuracy. The reduced memory requirements allow the flow memory to be in SRAM instead of DRAM. This in turn allows the per-packet processing to scale with line speeds [65].

There are three main differences between packet sampling and sample-and-hold [53]. First, the sample-and-hold technique decides, when sampling a packet of a new flow, whether to add this flow to the memory, from that point on, it updates the flow memory with every byte the flow sends (see Figure 2.4). This allows sampling more packets belonging to the different flows and provides more accurate results than those provided by Packet sampling. Second, sampling-and-

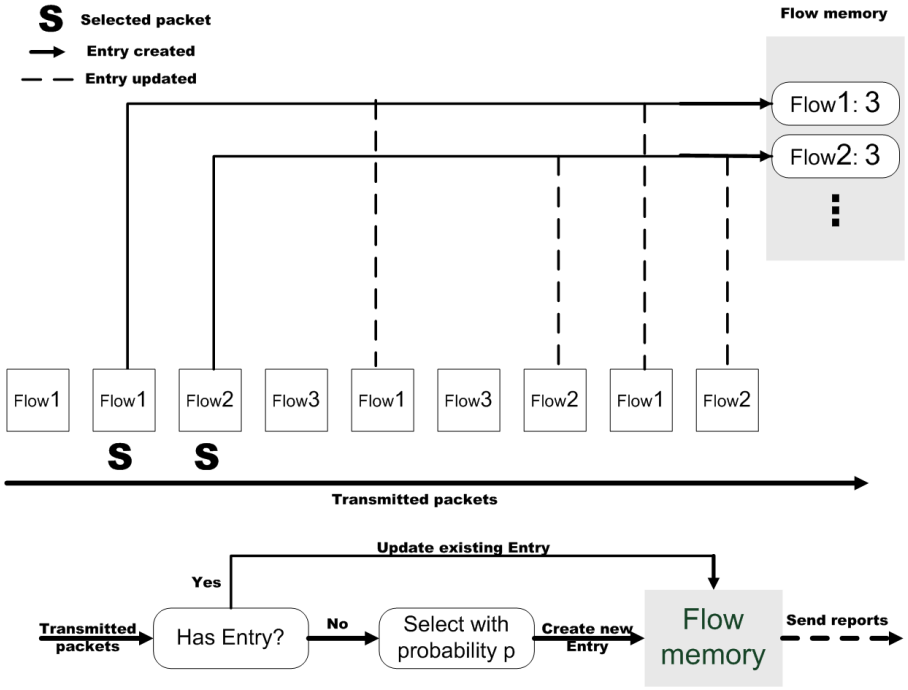


Figure 2.4: Sample and hold procedure.

hold technique avoids packet size biases, since once it decides to sample a flow it collects all its packets, unlike packet sampling which samples every k packets. Third, this technique optimizes resources consumption since it reduces the extra resource overhead (router processing, router memory, network bandwidth) for sending large reports with many records to a management station. [65, 50]

2.3.3.3.1 Packet sample and hold Packet sample and hold is well suited for traffic engineering and accounting applications that analyze volume structure [93, 50]. It keeps near exact counts of ”heavy hitters” flows with high packet counts. Packet sample and hold works as follows: For each packet, the router checks if it is tracking this packet's flowkey, defined over one or more fields of the IP 5-tuple. If yes, the router updates that counter. If not, the flowkey for this packet is selected with probability p, and the router keeps an exact count for this selected flowkey subsequently. Since this requires per-packet counter updates, the counters are kept in SRAM [50].

Sample and hold can be used to identify and accurately measure the packet hogs without keeping state for most other entries. Packets are sampled at random, for each sampled packet, an entry is created in the target table if one does not already exist, and all packets corresponding to that entry are counted from then on. A high probability gives accurate results while a lower

one reduces memory usage but allows more packets go uncounted before the entry is created.

2.3.3.3.2 Flow sample and hold Flow sample and hold is similar to packet sample and hold but its sampling function favors entries with many flows. It uses hashes to decide whether it samples a flow. It hashes the flow identifier of every packet. If the hash value is less than a given control variable or probability f, it creates a new entry in memory to collect statistics of this flow. Note that all packets of a flow have the same hash value, so the number of packets in a flow have the same hash value. Flow sample and hold is used to accurately identify the flow hogs regardless of how many packets they have [65, 67].

2.3.3.4 Smart sampling

Confronted with the unfeasibility to collect all raw flow records due to resources required for transmission, collection, and storage, Nick Duffield et al. have proposed a new sampling solution called Smart sampling. Smart sampling is controlled through a parameter known as the sampling threshold. In smart sampling, flows of size greater than the sampling threshold are always selected, while smaller flows are selected with probability proportional to their size. The motivation for smart sampling comes from the empirical fact that flow sizes have a heavy tailed distribution [56]. In this case, sampling with a uniform distribution over flow sizes is problematic for usage estimation, since the estimates are very sensitive to omission of a single large flow. Smart sampling avoids this problem by always selecting large flows. This approach was proposed in [47, 40].

Smart sampling is a technique to get a reliable estimate of detailed usage from only a subset of flow record by using the suitable sampling strategy. It exploits the fact that a large fraction of usage is contained in a small fraction of flows. By preferentially sampling larger flows over small ones, it can control the volume of statistics while simultaneously controlling the variance of statistical estimates derived from them. Smart sampling provides a tradeoff between improving measurement accuracy and limiting the resulting overhead. It entails balancing those two objectives in an optimal manner. Duffield in [40] proposes size-dependent sampling. He defines a size threshold. Flows of byte size greater than the threshold are sampled with probability 1 and the others are sample with the probability proportional to their size. He develops this sampling theory in [39, 41, 47, 43].

Nick Duffield et al. have proposed two variants to achieve these objectives:

2.3.3.4.1 Threshold sampling In a first approach they have introduced the threshold sampling techniques that represent a stream-based method suited to sampling records as they are exported or collected. For instance, in [47] they have introduced threshold sampling as a sampling scheme that optimally controls the expected volume of samples and the variance of

estimators over any classification of flows. Moreover, they have provided algorithms for dynamic control of sample volumes robust to variation in network conditions The goal is to find a good flow sampling strategy in order to constrain samples to within a given volume, to minimize the variance of usage estimates arising from the sampling itself, and to bind the sampling parameters to the data in order that usage can be estimated transparently.

To this end, they have proposed to continuously stratify the sampling scheme so the probability that a flow record is selected depends on its size. This attaches more weight to larger flows whose omission could skew the total size estimate, and so reduce the impact of heavy tails on variance. They have introduced a mechanism to tune the sampling probabilities in order that the volume of sampled records can be controlled to a desired level called threshold sampling, even in the presence of temporal or spatial inhomogeneity in the offered load of flows.

In [39], they have provided a traffic analysis platform that manages the trade-off between estimation accuracy and data volume. They have described the set of analytical tools that enable planning of resources in this platform. It enables the correct setting of sampling parameters and dimensioning of resource compatible with accuracy goals.

They have described the sampling error that arises from sampling: how it can be predicted from models and raw data, and how it can be estimated directly from the sampled data itself. Secondly, they have shown how to determine the usage of resources bandwidth, computational cycle, storage within the components of the infrastructure. These two sets of methods allow the dimensioning of the measurement infrastructure in order to meet accuracy goals for usage estimation.

2.3.3.4.2 Priority sampling Priority sampling methods sample a fixed number of ”best” records from a population. This solution is suited to database sampling for fast query execution. For instance, in [22], they have used ”priority sampling” to preprocess a large table of items, each of which carries a weight. The preprocessing assigns priorities to the items. After preprocessing, they estimate the total weight of an arbitrary subset using only a few high priority samples from the samples. Using unit weights, they could also get the number of people in a subset, and thus estimate averages. In [43], they have addressed the sampling strategy for flow records and they have proposed a correlated sampling strategy able to select arbitrarily small number of the ”best” representative of a set of flows.

This work was motivated by the need to sample within the measurement infrastructure, the flow statistics currently collected by network routers and switches. The heavy tailed nature of packets and bytes per flow forms a challenge for sampling, since omission of a record of single large flow can seriously degrade the accuracy of estimates of network traffic usage. The recent proposal for smart sampling, i.e., size dependent sampling of flow records, forms the starting point for this work. They have addressed the need to bound resource usage (bandwidth, stor-

age, aggregation caches) by the sampled flows. Any uncontrolled overflow of these resources seriously impairs estimation accuracy. They have proposed a simple algorithm that allows smart sampling from a population of flow records into a buffer of fixed size m, while maintain the ability to form unbiased usage estimates. The algorithm is simple, requiring the generation of one random number per flow, and can be implemented in constant time per flow, independent of m.

2.3.3.5 Trajectory sampling

Traffic measurement is a critical component for the control and engineering of communication networks. We argue that traffic measurement should make it possible to obtain the spatial flow of traffic through the domain, i.e., the paths followed by packets between any ingress and egress point of the domain. Most resource allocation and capacity planning tasks can benefit from such information. Also, traffic measurements should be obtained without a routing model and without knowledge of network state. This allows the traffic measurement process to be resilient to network failures and state uncertainty. **T**rajectory **S**ampling (TS) is a novel method to measure network traffic in potentially large network domains [45, 38, 75]. It is designed to provide detailed views of network traffic that can drive a wide variety of network engineering and management applications, such as traffic reporting and characterization, attack and intrusion detection and diagnosis, traffic engineering, and capacity planning. TS implements consistent packet sampling. Conceptually, each packet traversing a measurement domain is sampled either on every link traversed or on no link at all. Some highly compressed information on sampled packets are collected centrally in a collection system, which is then able to reconstruct the trajectory (path) of each sampled packet [44]. This set of sampled packet trajectories provides a complete and statistically representative view of the flow of traffic through the domain. A wide range of application specific metrics and derived views (such as traffic matrices for traffic engineering or sink trees for a DDoS attack) can be inferred from this raw trajectory information.

Trajectory sampling has been proposed in [45] as a monitoring method that enables new network management applications or enhances existing ones.

- Network Engineering: the fundamental new data provided is the path matrix. This describes not only the intensities of traffic between origins and destinations (i.e. the OD traffic matrix) but the intensities along the specific network path that traffic between a given OD pair flows. The path matrix is measured directly, without the need to track or join with network-wide routing information, as other monitoring methodologies require.

- Real-time Route Troubleshooting: routing loops are manifest as self-intersecting trajectories. The transient effects of routing and other path changes are measured directly,

without the need for up to date routing data, or assumptions concerning routing stability and convergence times. The time required for identification is limited only by the latency in reports reaching the data collector, and in the grouping and analysis of reports on a given packet.

- Passive Performance Measurement: loss packets are manifest in incomplete trajectories. If packet reports also contain a timestamp, and the monitoring points are accurately synchronized (e.g. via low cost GPS receiver) the data collector can also estimate the packet's transit delay between the monitoring points (One Way Delay).

In [45], Duffield et al. have proposed a method that allows the direct inference of traffic flows through a domain by observing the trajectories of a subset of all packets traversing the network. The key advantages of the method are that *(i)* it does not rely on routing state, *(ii)* its implementation cost is small, and *(iii)* the measurement reporting traffic is modest and can be controlled precisely. The key idea of the method is to sample packets based on a hash function computed over the packet content. Using the same hash function will yield the same sample set of packets in the entire domain, and enables us to reconstruct packet trajectories.

They have proposed a method for the consistent sampling of packet trajectories in a network. The sampling selects a subset of packets, but if a packet is selected at one link, it will be selected at every other link it traverses. On traversing the network, each packet implicitly indicates whether or not it should be sampled through its invariant part, i.e. those bits that do not change from link to link. A hash of these bits calculated at each router, and only those packets whose sampling hashes fall within a given range of values are selected. This is communicated by the sampling router to the measurement systems. This enables post sampling analysis of distinct trajectories once the samples are reported. They proved that the proposed method has a number of desirable properties: *(i)* simple processing: the only per packet operations required are the division arithmetic on a small number of bytes in the packet header. No packet classification or memory lookups are used. *(ii)* no router state is required in the per packet processing of the router: packets being processed individually. No caching is required in the measurement subsystem of the router, thus avoiding cache delay and possible biasing through the requirement of cache expiry policies. This does not exclude the possibility of having state in the reporting system in the router. It may be desirable to aggregate discrete reports to the measurement system rather than sending them individually. *(iii)* Packets are directly observed: the course of the packets through the network can be determined without a network model that specifies how they ought to be routed. This is important for debugging since routing may not easily specify current routing state of the system. Moreover, configuration or other errors may cause actual routing behavior to deviate from that specified by the model.

In their work described previously, they have assumed that measurement devices are capa-

ble of reliably exporting measurements to the collector, which simplifies the task of the collector in reconstructing a representative set of trajectory samples. However, there are circumstances where such reliable export is either not desirable or not possible. Therefore, in [46], they have assumed that measurement devices export measurement report packets unreliably, which relieves them of the burden of buffering and processing acknowledgements. However, dealing with missing reports complicates the task of the collector. Therefore, they have proposed methods for the collector to deal with such loss. They have described enhancements to reporting and reconstruction that enables measurement based applications to function even when reports are subject to loss. They have defined methods to perform robust network monitoring using trajectory sampling in the presence of report loss. They have proposed solutions to reconstruct an unambiguous set of packet trajectories from the reports on sampled packets received at a collector. They have extended the reporting paradigm of trajectory sampling to enable the elimination of ambiguous groups of reports, but without introducing bias into any characterization of traffic based on the surviving reports. Moreover, they have adapted measurement based applications to incomplete trajectories by proposing a method to join multiple incomplete trajectories for inference, and analyzing its performance. They have shown how applications can distinguish between packet and report loss at the statistical level. They have proposed an approach based on bloom filters, a data structure that compressed a set membership function into a bit array. Bloom filters are appropriate because the only errors they incur are false positives, which may lead to the elimination of some unique labels in addition to actual duplicates. While this represents a small loss of measurement data, the main property of this approach is that the duplicate elimination essentially behaves like sub sampling the original set of packets. Therefore, by applying a correction factor, the resulting set of trajectories can be treated as if it had been obtained in a collision-free way. This insulates the estimation and detection procedures fed by trajectory samples from the intricacies of duplicate elimination. Once duplicate labels have been eliminated, the resulting report stream can be passed to applications. In general, applications must be adapted to report loss. Path tracing applications must amalgamate reports from several packets in order to reconstruct complete trajectories. Passive loss measurement applications must distinguish report loss from packet loss by exploiting transmission sequence numbers in the reports to estimate report loss rates. The performance analysis of these applications shows that trajectory sampling brings substantial advantages over independent packet sampling, reducing both estimator variance and reporting bandwidth.

After providing an exhaustive survey of existing sampling methods, we conclude that the deployment of these techniques lowers the cost of processing and storage resources. However, it presents some problems since it clearly reduces the accuracy of measurements and entails a loss of information. Hence, in order to surmount these problems and to cope with the increasing trend in line speed and to improve measurement accuracy, we have to multi-

ply the measurements inside the network and to combine their measurements. Moreover, we have to reconfigure monitors in order to optimize resource consumption and improve measurement accuracy. Recently, many works have proposed network-wide monitoring systems that coordinate monitoring responsibilities and distribute the work between the different monitors e.g. [48, 86, 28].

In the next section, we will present some works and we will discuss their advantages and shortcomings.

2.4 Designing novel network monitoring architecture

Network traffic does not exhibit stationary behavior. It exhibits temporal cycles (daily, weekly) and can be easily affected by network reconfigurations (e.g. dynamic routing change), link failures and deployment of new machines and applications. Although using sampling within monitoring applications is a lightweight process that does not consume much of the network device resources (processing and storage resources), there are certain limitations with respect to the maximum number of samples the device can produce within a given period. Thus, in periods of high traffic rate the device and the network may become saturated. Moreover, some applications require high sampling rate values and some network behavior cannot be accurately detected with low sampling rates (e.g., anomaly detection). On the other hand, sampling with high sampling rates produces huge amounts of data. Hence, we need to find the best sampling rate values for each case in order to address the tradeoff between accuracy of results and the amount of collected and reported data. Some works have proposed adaptive sampling techniques. For instance, the authors in [58] have designed the fast control loop for adjusting the sampling interval. They have discussed the linear prediction and fuzzy logic approach with application to different types of traffic (normal Internet traffic and bursty video traffic). Authors had performed offline simulations and did not analyze the feasibility of embedded implementation in the device. [29] proposes to use the weighted least squares predictor and certain set of heuristic rules for determining the sampling rate. [47] describes an approach to flow sampling that allows to control the expected volume of samples and to minimize the variance of usage estimates arising from the sampling. The proposed schema (smart sampling) adapts the sampling process by associating the probability that flow is selected with the size of the flow. This process shifts focus towards the larger ”elephant” flows which have a severe influence on the traffic volume. These ideas are further extended in [43] to work under strict resource constraints by sampling into a buffer of fixed size. However, these methods require that the special sampling mechanism is present in the network devices. Vendors do not want to implement sophisticated sampling schemes that give good results under certain circumstances. They want to implement simple and robust solutions that are well described by some form of a standard (i.e.

sFlow, NetFlow). Thus proposals have introduced new monitoring systems that extend existing monitoring tools (e.g., NetFlow) with a central unit that updates sampling rates and place monitors according to network conditions and monitoring application requirements. The common objective of such distributed monitoring systems is to sample traffic in a cost-effective manner by carefully placing monitors and controlling their sampling rates. For instance, the authors in [89], have considered the problem of where to place monitors within the network and how to control their sampling rates in order to address the tradeoff between monitoring cost and monitoring coverage. They have proposed minimum cost and maximum coverage problems under various budget constraints and they have introduced greedy heuristics that provide solutions quite close to the optimal solutions. In addition, they have demonstrated that a small number of monitors is often enough to monitor most of the traffic in an entire IP network. This near-optimal monitor placement uses operating strategies in a distributed monitoring system, which operates either in sampling or non-sampling mode. Each deployment strategy determines the maximum number of monitors and their locations under a given budget constraint or determines the minimum deployment cost for a maximum number of monitors. Also, the operating strategy of each monitor determines the flow sampling rate. More specifically, they introduced a novel monitoring cost and reward models for a distributed passive monitoring system, which can accommodate both sampling and non-sampling modes of the monitoring system. Based on these models, they formulated a set of placement and operating problems assuming different constraints for budget and coverage requirements. Since the various placement problems are NP-hard, they proposed approximation algorithms based on greedy heuristics to determine placement locations and used an advanced method to get sampling rates. They showed that only a small fraction of links need be monitored to achieve a high level of monitoring reward. The authors in [28], have introduced a general framework to solve the problem of sampling traffic data in large IP networks. They proposed to combine and solve in one step the selection of traffic monitors and the setting of the sampling rates for each monitor. They provided an optimal algorithm to solve the sampling and placement problem. Moreover they have introduced additional methods that allow adapting the sampling rates to changes in the traffic due to time-of-day effects, failures or anomalies. The method receives as input the network topology, the routing matrix and the set of OD pairs of interest. It returns a set of monitors and their sampling rates that is optimal with respect to the measurement task to perform. To do so, they have defined a constrained problem and they have reformulated the optimization problem using Lagrange multipliers. To solve the problem and find the optimal sampling rate vector they used the Karush-Kuhn-Tucker (KKT) conditions. They rely on an iterative procedure to explore the solution space. They keep iterating until either reaching a point that satisfies the KKT conditions and therefore is the optimal solution, or until exceeding the maximum number of iterations. The method gives best information compared to alternative solutions since it can

configure sampling rates according to measurement accuracy and network link load.

2.5 Conclusion

In this chapter, we have provided a brief history of the most useful monitoring tools. We have provided the chronology of the different solutions and the main requirements that led to their appearance. We have started by presenting the SNMP counters and we have discussed its advantages and shortcomings. We illustrated our discussion with important monitoring applications (i.e. traffic matrix estimation, anomaly detection). After demonstrating the inefficiency of the use of SNMP counters and the need to a new more reliable monitoring tool, we have presented the most popular monitoring solution (i.e. Netflow). Furthermore, we have presented some sampling variant of Netflow. Then, we provided a brief definition for each of the most important sampling techniques. Finally, we have presented some recent extended works based on multiplying monitoring points inside the network in order to cope with the estimation error coming from sampling.

In conclusion, based on the above stated survey, we have extracted the important principles we want to follow throughout our research work. Also, we have designed and implemented a new wide-network monitoring system. This system deals with sampled Netflow and supports monitors reconfiguration. Finally, our system tries to find the optimal configuration that provides the best tradeoff between accuracy and resource consumption. Moreover, it combines measurements collected from the different monitoring points in order to provide better estimation.

3

EVALUATION METHODOLOGY FOR NETWORK TRAFFIC MONITORING SYSTEMS

There is an increasing interest in traffic measurement and monitoring solutions in order to understand the performance of network infrastructure and to efficiently manage network resources. Therefore, it is of utmost importance for the network research community to have access to tools and testbeds to explore the future directions for Internet traffic monitoring and engineering. Although many experimental solutions exist today, they tend to be highly specialized, or to have a limited availability and openness. Through this chapter, we outline the monitoring capabilities limitations of these facilities and we present our experimental platform for network wide traffic monitoring as an answer to these limitations. Our platform presents a new approach for the emulation of Internet traffic and for its monitoring across the different routers. Moreover, we provide a global analysis study in order to characterize, qualitatively or quantitatively, what impact a particular input parameter has on a system output.

Contents

3.1 Experimental Platform for Network Wide Traffic Sampling and Monitoring

3.1.1 Introduction

For the purpose of testing new applications and protocols related to traffic monitoring and traffic engineering, the network research community has a persistent demand of either having a real-time control on real measurement points or getting collected traces from these points together with the related network topology. If available, and even in the absence of real-time control, network-wide traces can be played a posteriori to simulate as close as possible the real environment. The problem is that these needs are most of time not satisfieed by ISP(s) for privacy and security reasons. Thus and as an intermediate solution, researchers make use of

either network emulators coupled with synthetic traffic generators [27] or simply limit their research to parsing and studying link-level traffic traces collected on specific links. Choosing one of these last solutions depends on a set of factors namely the available information, the context of the algorithm to evaluate, the solution realism, the facilities provided by the solution in terms of monitoring and finally its extensibility.

As other researchers, we are also facing situations where access to network-wide traces and a control on measurements points are required. In particular, we are seeking a network-wide monitoring platform that allows us to control the monitoring tools inside each router and to report measurement results to a central unit for further analysis. This is useful to study several applications as distributed anomaly detection, traffic engineering, and network-wide optimization of monitoring tools. Unfortunately, such platform is not available for researchers that do not operate real networks. One solution could be to play synthetic traffic inside network simulators. Even though it is an interesting option, we discard it in our research for its lack of realism and we focus instead on the development of an emulation platform where real monitoring tools run inside virtual routers, and where generated traffic is faithful to the one in real ISP networks. The main question we try to solve then becomes: starting from a *partial* set of collected real traces and given a network topology, how to build a platform of virtual routers respecting this topology, and how to dispatch and play over it the available traces while providing to the end-user remote controllable traffic monitoring capabilities for each router?

In this section, we present the platform [8, 68] we developed as an answer to the latter question. Our platform presents a new approach for the emulation of Internet traffic and for its monitoring across the different routers. In its current version, the traffic is sampled at the packet level in each router of the platform, then monitored at the flow level. We put at the disposal of users real traffic emulation facilities coupled to a set of libraries and tools which are capable of Cisco NetFlow [30, 2] data export, collection and analysis. Our aim is to enable running and evaluating advanced applications for network monitoring and optimization. We believe that the framework we are proposing can play a significant role in the systematic evaluation and experimentation of network-wide monitoring and optimization algorithms. Among the direct candidates figure algorithms for traffic engineering and distributed anomaly detection. Methods for placing monitors, sampling traffic, coordinating monitors, and inverting sampling traffic will find in our platform a valuable tool for experimentation.

3.1.2 Related Work

Several recent works were interested in the evaluation of traffic monitoring solutions. The majority of these works deals with the understanding of the content of the traffic on some network link. They use for their evaluation packet level traces collected by network operators or by researchers themselves, then made available after anonymization. The traces of the

Japanese MAWI project [7] are a typical example that we use in this work. Packet level traces have been used for several purposes as to study the statistical properties of Internet traffic [54], to detect anomalies and attacks [90], to validate efficient methods for application identification and classification [25], or to study the accuracy of estimating traffic statistics from a stream of sampled packets [59]. Works in this list, which is far from being exhaustive, all share the feature of only requiring the availability of traces collected on Internet links. Clearly, the more of these traces are available, the stronger is the validation.

On the other hand, one finds studies that require the existence of network-wide data and access to routers and monitors. Those studies are more challenging since their requirements in terms of validation cannot be unfortunately satisfied except for network operators who possess such data and monitors. The most common approach is to resort to simulations or emulations where synthetic traffic is played across the studied ISP topology. In [89], the authors follow this approach to optimize the placement of monitors and the sampling rates inside routers. In [86], Sekar et al. develop their own synthetic traffic generator tool; they use Monlab [8, 68] for setting the test network topology and YAF [16] as a monitoring tool to capture flow information. The existence of network-wide real data would help to strengthen the validation of these solutions even further. Such data has helped Duffield et al. in [48] to propose a solution for estimating network-level flow rates from measurements taken at multiple routers. They use sampled NetFlow records gathered from two routers in a major ISP network. Lakhina et al. [20] have used NetFlow data collected from the PoP routers of a tier-1 ISP network to detect network-wide traffic anomalies and classify them. Generally, the collected data is parsed offline with appropriate scripts implementing the proposed algorithms. It is also used to calibrate models for the generation of synthetic traffic. Unfortunately, real access to monitors to analyze the reports they send and apply the proposed algorithms on the fly is still hard if not impossible to obtain by the research community.

Our platform provides a tool for researchers to play real packet traces over a target network topology and to control the monitors installed inside routers. These monitors sample and capture the traffic, then send flow-level reports about it in real time to a central collector. The key feature of our platform is that it splits the available packet-level traces across the emulated topology without resorting to synthetic models. This guarantees realism while providing enough freedom in defining the way the real trace is split and the network topology is formed. Routers are virtual entities of very low complexity, which allow the platform to scale to large networks as long as rich packet-level traces exist. Users can control the way the traffic is sampled and monitored, and can use the real time reports sent by monitors to feed their own applications. The applications they develop over our platform can run later, and as they are, over real networks. These are required features in any platform.

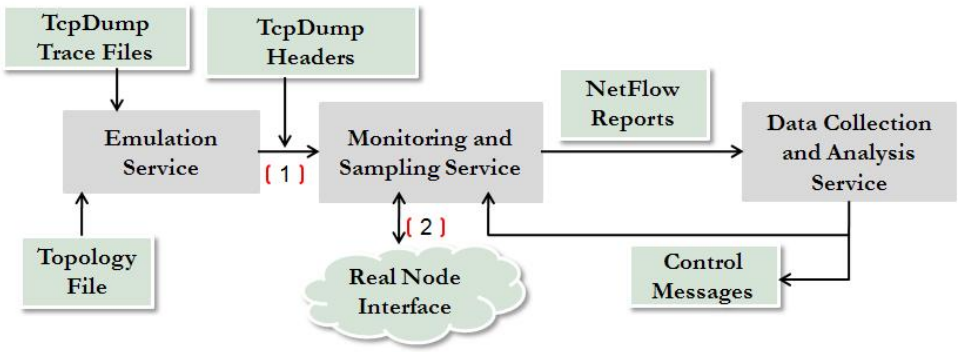


Figure 3.1: Experimental Platform Architecture.

3.1.3 Platform architecture

The objective of this work is to build a real environment for flow-level monitoring inside ISP networks starting from traffic emulation to traffic sampling and monitoring. We reproduce and approximate this real environment through our platform described in Figure 3.1. The latest depicts the interactions between the three main components of our architecture: the emulation service, the flow monitoring and sampling service and the flow data collection and analysis service.

A common usage scenario of our platform can be summarized as follows. The user starts by supplying the emulation service with two XML configuration files describing respectively the list of packet-level traces to play and the network topology. Once done, the user runs the three services of the platform either in the same machine or in three different machines for better performance [1]. The emulation service creates the virtual routers and links to emulate the described topology, then dispatches the set of IP addresses available in the traces over the emulated topology and plays the packets accordingly. The monitoring and sampling service installed in each router allows to sample the emulated traffic at the packet level, to construct flows and to export flow reports to the data collection and analyses service. Within this latest service, the user can plug and run any advanced traffic analysis algorithm. Next we describe the services of our platform with further details.

3.1.3.1 Traffic Emulation Service

The traffic emulation service provides means to describe a given network topology. The emulated network is then fed with a real traffic captured on some high speed link in a backbone transit network, before being dispatched and played over the emulated topology. Figure 3.3

[1]Towards a better scalability and as part of the future work, we intend to enhance the platform in such a way that users can emulate different set of routers over different machines.

depicts an example of a simple topology that users can describe. We provide a highly flexible configuration methodology via XML files through which users can describe their emulated network. In particular, they can describe the different ASes and their associated weights[2], the list of routers (interfaces, IP addresses), the set of characteristics of the different links as well as the list of monitors deployed inside routers. Once the emulated topology is available, our emulation service looks for IP prefixes within the available TcpDump traces and dispatches them over the stub ASes according to their weights. The packets of these traces are then played over the routers of the topology respecting their timestamps to reproduce the network wide packet-level traffic we are looking for.

Figure 3.2 shows the processing flow within the emulation service. As the traffic loader module reads packets from the TcpDump trace using the Pcap library [14], the dispatching module associates them online to the right AS based on one hand on the list of weights the user attributes to the different stub ASes, and on another hand on the prefix length specified by the user for the dispatching. The default dispatching method we are implementing is the weighted random. Suppose we have 23 ASes to which we want to associate different weights. We divide the interval $[0, 100]$ into 23 consecutive bins, where each bin represents the weight of an AS. Suppose further that the user chooses the prefix length to dispatch as being the /16. Then, each time a new prefix of the same length appears while reading the raw trace, a random number is selected within the interval $[0, 100]$ using a uniform distribution. This selected number points to the AS to which the new prefix is to be associated. All subsequent packets having the same prefix as source or destination are associated to the same AS[3]. Once loaded packets are dispatched, they are scheduled and played within the routers' emulation module. And as a final step, if there is a router within the transit network selected as a monitor, the data forwarding module enables the monitoring and sampling service described in Figure 6.2 over the interfaces of this router.

Note that towards scaling up the experimentation to large scenarios, we parallelize the maximum number of tasks within the emulation service, namely those required for trace parsing, IP prefix mapping and packet playing. The objective is to profit from all the CPU power provided by the host machine and to prevent excess latency introduction during packet scheduling. So, all the modules described in Figure 3.2 run in parallel. Furthermore, we resort to massive dynamic memory development to minimize the emulator memory footprint. Indeed, we profit from the parallelism that we have introduced to parse and play large TcpDump binary files without having to load them entirely into the memory. Instead, as loaded packets are played inside routers through the emulation module and consumed by the data forwarding module,

[2]ASes connected to the emulated network are supposed to generate different amounts of traffic with respect to the whole network traffic.

[3]Depending on users' needs, one could imagine other dispatching methods as well.

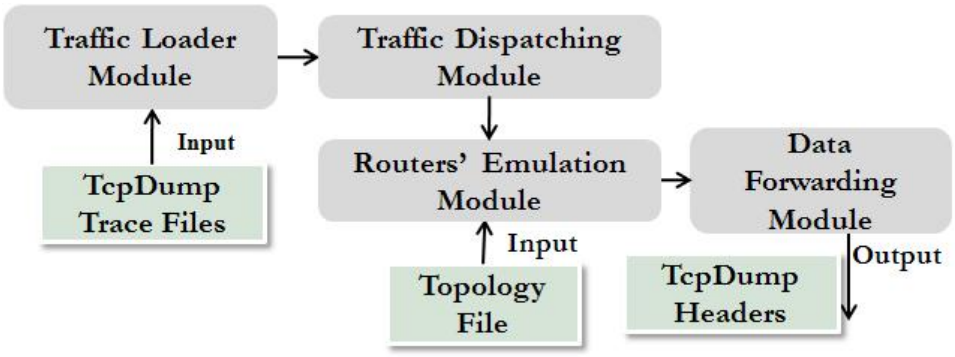


Figure 3.2: Traffic Emulation Service.

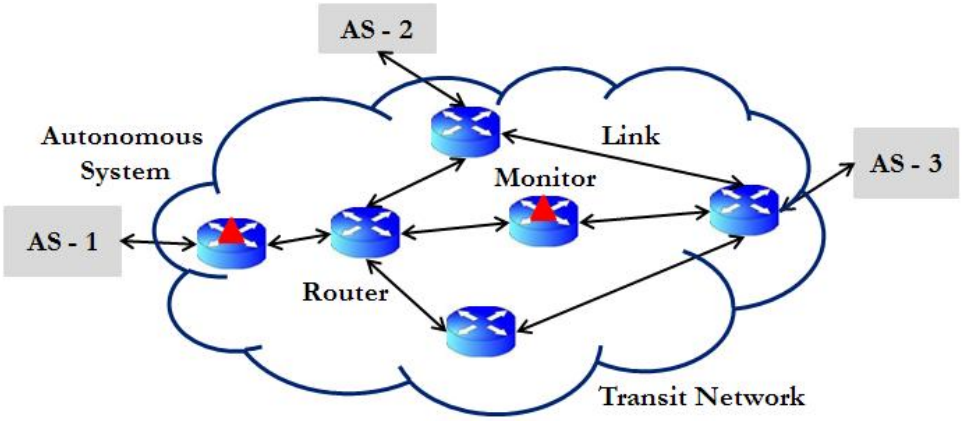


Figure 3.3: Example of emulated topology.

new ones are loaded via the traffic loader module. The amount of packets to load at once is dynamically adjusted to limit the latency introduced during packet scheduling.

In terms of extensibility, new traffic control and monitoring methods can be added without major changes to the emulation service design. In addition, users interested in studying the performance of different routing algorithms can plug their own routing module providing that they maintain the same programming interface as the default one. For the moment, we are only supporting the shortest path routing protocol based on the Dijkstra algorithm [34]. Routes are static and are set up via the XML configuration file.

3.1.3.2 Traffic Monitoring and Sampling Service

We have designed and developed the traffic monitoring and sampling component as an extension of Softflowd [12]. Indeed, Softflowd is a flow based network traffic analyzer capable of Cisco NetFlow data export. Note that Softflowd does not include support for packet sampling neither fixed nor adaptive. Our extensions to Softflowd add this sampling functionality, which

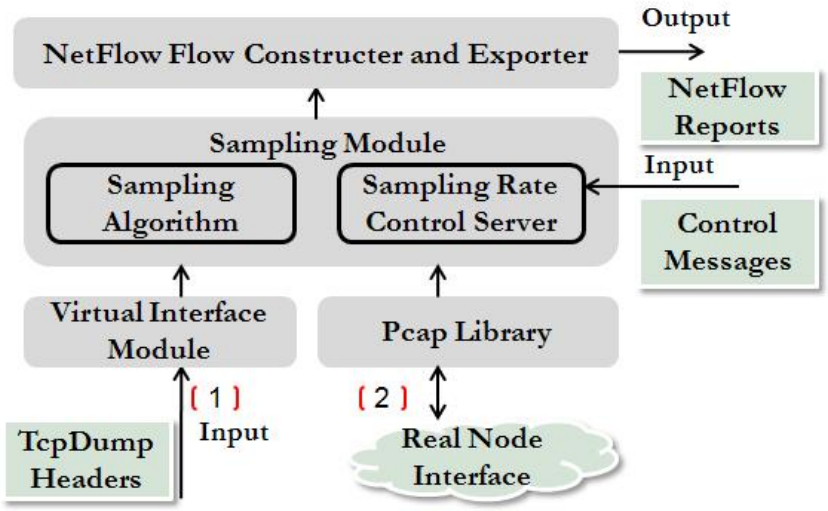


Figure 3.4: **Monitoring and Sampling Service.**

is essential for monitoring scalability. They also allow the integration of this tool over virtual nodes fed by our emulated traffic, while being able to run over real routers by sniffing packets directly on their interfaces. The last functionality enables users to first run and evaluate their sampling and monitoring methods on a set of nodes within a controlled environment and then use these methods within real routers without any extra development effort. Next we describe these extensions through two typical usages scenarios.

- ***Experimenting with emulated nodes:*** The monitoring and sampling service communicates with the emulation service via the virtual interface module described in Figure 3.4. So, a user willing to plug the monitoring and sampling service to the emulation service has to run it by specifying the port number on which the virtual interface module will listen to incoming packets (per-monitor TcpDump headers) from the emulator. Then, received packets are sampled via the sampling module and forwarded (if chosen) to the NetFlow flow constructer and exporter module, which takes in charge the construction of the flows and the NetFlow reports to be sent to the data collection and analysis service described in Figure 3.1.

- ***Experimenting with real nodes:*** Our monitoring and sampling service is also designed to be plugged directly to a real node interface. In this case, it captures traffic promiscuously using the Pcap library [14] as described in Figure 3.4. Note that it is likely that the sampling module will place additional load on hosts or gateways on which it runs. Our implementation has been designed to minimize this load as much as possible. Indeed, in order to decide either to consider the packet being read or to reject it, the sampling module makes a decision each time the Pcap library returns a handle to a new packet and before the packet is being loaded to the memory.

If the result of the sampling algorithm is to capture the packet, the entire packet is then loaded and the maintained flow list is updated via the NetFlow flow constructer module, otherwise the packet is simply discarded.

As described in Figure 3.4, the sampling module encloses a sampling algorithm as a core and a sampling rate control server. The default sampling algorithm we are providing is the following: if a user chooses a sampling rate of A/B (A packets among B packets, $A \leq B$, $B > 0$, $A \geq 0$), then every B packets, the sampling module generates randomly a set S of A numbers within the interval $[1, B]$. Packets with numbers outside the set S are rejected and only the remaining packets are considered for 5-tuple flow construction. Generated flows are then encapsulated within NetFlow reports and are exported via the NetFlow flows constructer and exporter module. The sampling rate control server enables users to change remotely the sampling rate of a given monitor whether it is in a real network or downstream the emulation service. The remote monitor controller, which we will describe later, proceeds to change the local sampling rate to each monitor. This remote control functionality allows users to control and change online the sampling rate of one or multiple monitors, or simply offline from one experiment to the other.

3.1.4 Data Collection and Analysis Service

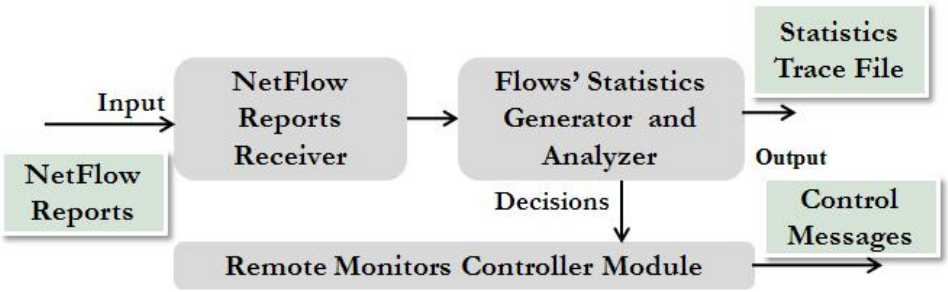


Figure 3.5: **Data Collection Service.**

Figure 3.5 depicts the principal modules of the data collection and analysis service. We design and develop the data collection and analysis service starting from the functionalities proposed by Flowd [3]. As described in Figure 3.5, we enclose the Flowd capabilities within the NetFlow reports receiver module around which we develop other modules namely the flow statistics generator and analyzer and the remote monitor controller modules. Depending on user needs, one can easily enhance this module. For example, one can implement anomaly detection based on the collected NetFlow reports or introduce quality of service algorithms that improve traffic routing as a function of the monitored network status. And independently of the process of data analysis, the user can decide to change the sampling rate of one or more

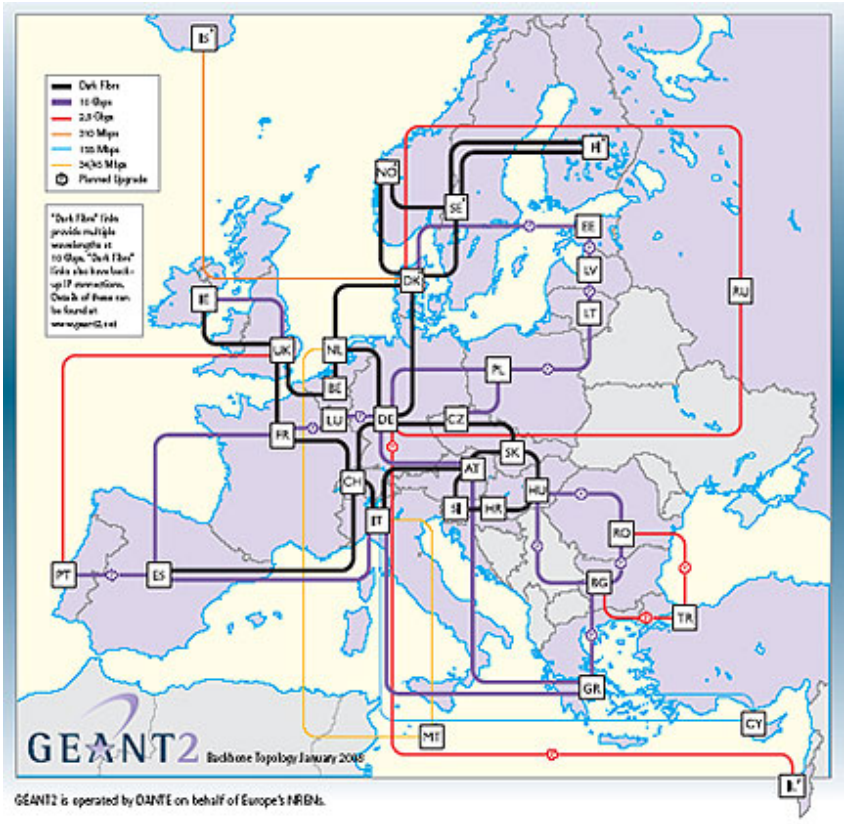


Figure 3.6: Geant topology.

monitors to improve the accuracy of the algorithms implemented at the analyzer, either offline or online. For this, he has to use the remote monitor controller. The latest module provides an API to send control messages to a monitor specific control server, which proceeds to modify the sampling rate of the given monitor.

3.1.5 Validation of the platform operation

We validate the effectiveness of the dispatching module (described in Section 3.1.3.1) by tracking the number of sampled flows and the number of packets across the different edge routers to check whether they follow the access network weights (or traffic matrix) predefined in our experiment configuration file. For that we use a trace of 15 minutes.

3.1.5.1 Test Environment

To emphasize the realism and the scalability of the platform we are proposing, we choose to reproduce the GEANT backbone topology [15].

The GEANT network 3.6 is a pan-European backbone that connects Europe's national research and education networks via 23 routers. In another hand, we choose to play different traces collected at a trans-pacific line from the MAWI working group traffic archive [7]. Traffic traces are made by TcpDump, and then, IP addresses in the traces are scrambled by a modified version of Tcpdpriv [13]. For traffic dispatching over the different access networks (European countries in this experiment), we associate different weights to the 23 ASes connected to GEANT based on the importance of the traffic they are supposed to generate in reality. We infer these weights from the populations of the countries represented by these ASes and the capacities of the links that connect them to their respective access routers in GEANT. The three services of the platform run on one machine each. (machines are connected via an Ethernet network). Indeed, the traffic collection and analysis service is supposed to be a centralized component that collects NetFlow reports sent by the monitoring and sampling service. The latter is connected to the 23 emulated routers via its virtual interface module.

3.1.5.2 Results

Figure 3.7(a) depicts the evolution of the network-wide monitored number of flows function of the sampling rate in routers that we vary while maintaining it the same for all routers. As expected, the number of flows decreases linearly as we decrease the sampling rate in all the access routers. Indeed, if S is the size of a given flow, then the probability that this flow is sampled given a sampling rate p is equal to $1-(1-p)^S$, which can be approximated by $p.S$ for small p. The number of sampled flows N_p can then be approximated by $p.E[S].N$, where N is the total number of original flows. This latter quantity clearly decreases linearly with the sampling rate. Next, we look at the effectiveness of the emulation service and especially of its dispatching mechanism. We try to answer the following question: does each AS generate as much traffic as the importance of the weight associated to it? Towards that, we start by plotting in Figure 3.7(b) the number of prefixes per AS function of the weights associated to ASes. The resulting curves remain linear for different prefix lengths. So, we conclude that our emulator dispatches the prefixes to ASes without any bias. Then, we look at the number of generated flows, ingoing and outgoing packets per AS in Figures 3.7(c), 3.7(d) and 3.7(e), respectively. We notice that for the three prefix lengths (/16, /24 and /32), the number of generated flows as well as the number of ingoing/outgoing packets scale with the AS weights but do not fit a perfect line. Nevertheless, the fitting improves when the prefix granularity becomes finer. This indeed comes from the presence of very large prefixes of different volumes that, when dispatched over the ASes, cause such deviations in the traffic; the coarser the prefix the more important this phenomenon. We illustrate it on prefixes of length /32 (IP addresses), for which we plot the distribution of their sizes (in packets) on a log-log scale in Figure 3.7(f). Clearly, there is a power-law behavior leading to very large prefixes compared to the average prefix size

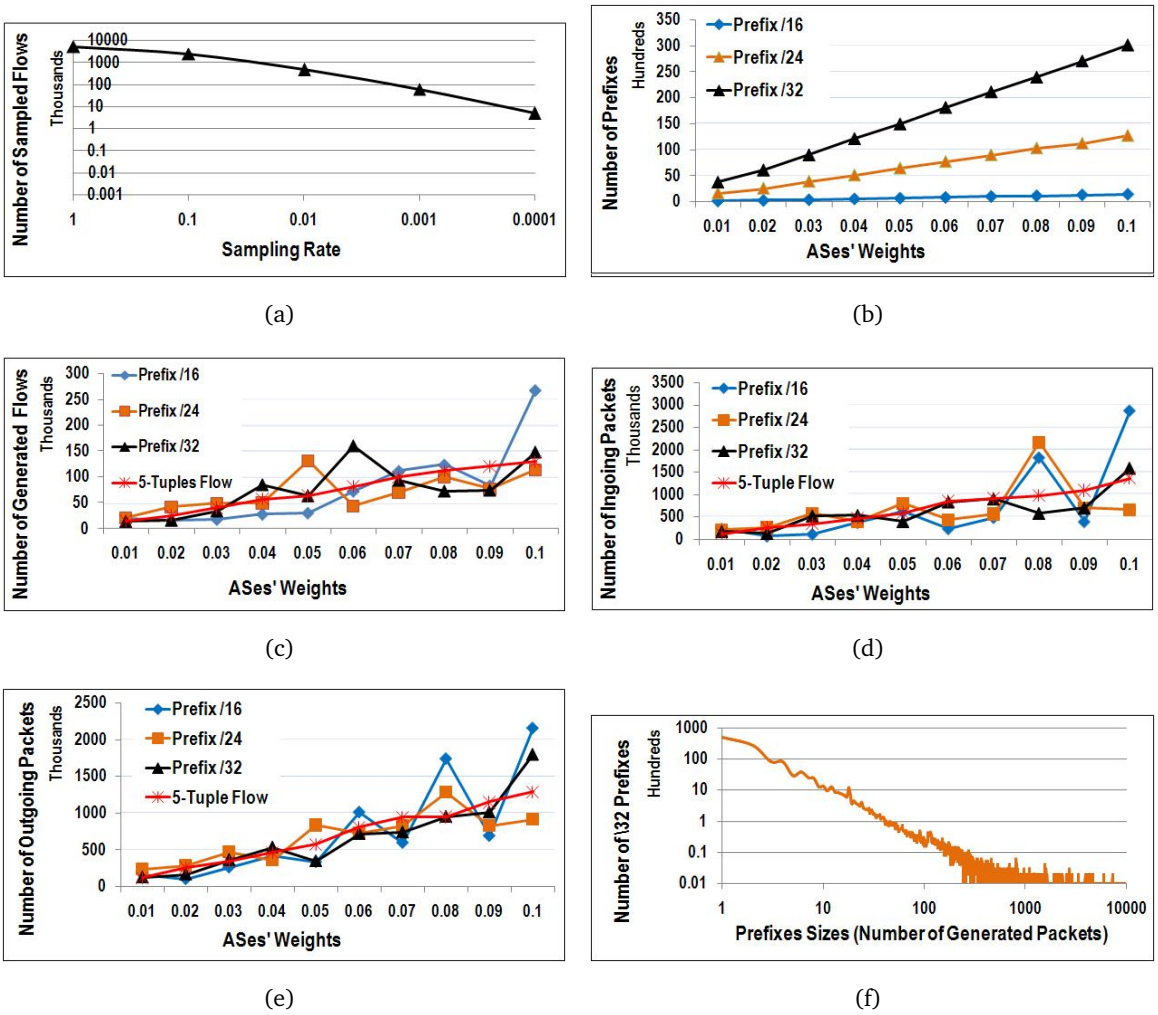


Figure 3.7: Validation Results.

(these are servers, heavy users, etc). To improve further the fit, one can run the dispatching at another finer granularity, the 5-tuple level. This finer granularity enables our dispatching algorithm to split a big set of 5-tuple flows generated by (or destined to) a single /32 prefix to different ASes. As we can see in Figures 3.7(c), 3.7(d) and 3.7(e), the number of generated flows and ingoing/outgoing packets fits now better a line. Even though it preserves the notion of connections, the 5-tuple dispatching still has the problem of altering the pattern of activity of servers and end hosts. The choice of the best prefix length should be decided by this tradeoff established between traffic realism and AS weight respect.

3.1.6 Summary

In this section, we have described our experimental platform for network wide traffic sampling and monitoring. The architecture that we propose contains three main components: an emulation service, a monitoring and sampling service, and a data collection and analysis service. The design of these services takes into consideration the constraints of environment conservation, scalability and extensibility. Our platform offers a complete set of features towards the development and evaluation of solutions for network monitoring and management. Namely, it offers the possibility to reproduce real backbone network topology, to monitor and sample the packets being forwarded in a given router and finally the ability to analyse the collected flows. Our platform allows users to remotely tune the sampling rate of a given monitor. Users can also easily introduce traffic engineering methods methods within the monitors.

In order to provide an exhaustive evaluation methodology for monitoring applications, we have used a global sensitivity analysis that allows studying the performance of the proposed system and the impact of the different parameters on its behavior. Next, we give a detailed description of this advanced variance-based analytical method.

3.2 Performance Analysis

3.2.1 Sensitivity Analysis Overview

Large-scale experimentation models (simulations, emulations, etc) often involve a large number of parameters, making it prohibitive to run more than a small fraction of all potentially relevant cases. In this context, sensitivity analysis attempts to identify how responsive the results of an experimental model are to changes in its parameters: this is an important tool for achieving confidence in experimentation and making its results credible. The general goal of Sensitivity Analysis is to characterize, qualitatively or quantitatively, what impact on a system a particular variable will have if it differs from what was previously assumed. In other words, by using Sensitivity Analysis, the analyst can determine how changes in one or several parameters will impact the target variable. Sensitivity analysis quantifies the dependence of system behavior on the parameters that affect the modeled process and in particular its dynamics. It is used to determine how sensitive a model is to *(i)* changes in the numerical value of the model parameters: parameter sensitivity analysis aims at determining the uncertainty associated with the numerical values of model parameters (resulting thus in parameter estimation but also prediction). Hence, sensitivity analysis is used to increase the confidence in the model and its predictions, by providing an understanding of how the model responds to changes in its parameters, and *(ii)* changes in the structure of the model.

3.2.2 Sensitivity Analysis Methods

Perturbation Theory based methods: They study a set of models which are different from a nominal model by some small terms. Sensitivity Analysis is closely linked with Perturbation Theory. Perturbation Theory comprises mathematical methods that are used to find an approximate solution to a problem which cannot be solved exactly, by starting from the exact solution of a related problem. Perturbation Theory can be applied if the problem under study can be formulated by adding a "small" term to the mathematical description of the exactly solvable problem. Thus, Perturbation theory can be viewed as a tool for Sensitivity Analysis. Furthermore, it can be classified as an analytic tool for the Sensitivity Analysis. The main types of mathematical models for perturbation methods are:

- Linear Algebraic Systems
- Non-linear Algebraic Systems
- Mathematical programming

The advantage of analytic Perturbation Theory based methods is that these methods are based on a solid theoretical ground. The disadvantage of the analytic methods is that typically the deviations of parameters need to be small and a good knowledge of the system's structure/dynamics is required.

The other class of tools for Sensitivity Analysis is Sampling based methods. Analytical methods require a good knowledge of the system and might require tedious calculations. The sampling based methods are designed to overcome these disadvantages. Sampling methods are particularly well suited to withstand the changing one-factor-at-a-time (OAT) paradox.

- FAST (Fourier Amplitude Sensitivity Test): A method which deals with static models.

  The main idea of FAST [36, 37] is to assign to each parameter a distinct integer frequency (characteristic frequency). Then, for a specific parameter, the variance contribution can be singled out of the model output with the help of the Fourier transformation. FAST is considered to be one of the most efficient methods in sensitivity analysis [82, 84]. Among its advantages are: fast implementation, deals with non-monotonic models, allows arbitrary large variations in input parameters, and does not require the knowledge of the mathematical model. The latter two features are in particular positively distinguishing FAST from analytical methods. However, FAST suffers from computational complexity for a large number of inputs. Moreover, the basic FAST method can only be applied to static models with independent parameters. As, in many cases the parameters are correlated with one another, extended FAST (EFAST) has been proposed for models with correlated parameters i.e. EFAST can address higher order interactions [83].

- Path based Sensitivity Analysis (of Markov Chains) for dynamical systems: The key idea in path-based sensitivity analysis of Markov chains is the observation that a sufficiently long sample path contains enough random deviations to test the system sensitivity.

Sampling based methods do not require access to model equations or even the model code. These methods require running a series of experiments. Experiments can be either real-life or numerical. The disadvantage of the sampling based methods is that the number of experiments required can be very large.

In the previous part, we have presented an exhaustive overview on performance analysis methods. The goal is to find an accurate solution that gives the impact of parameter changes on the results of our experimental model. In this thesis, we choose to use FAST method within our experimentation models. In the next section, we will give a detailed presentation of the FAST method. We will present how it characterizes the impact of the different parameters on the system output.

3.2.3 Fourier Amplitude Sensitivity Test

The main idea of FAST is to assign to each parameter a distinct integer frequency (characteristic frequency). Then, for a specific parameter, the variance contribution can be singled out of the model output with the help of the Fourier transformation. Therefore, FAST is also referred to as variance based sensitivity analysis.

Specifically, let us consider a nonlinear model $y = f(x_1, x_2, ..., x_n)$ where x_n are parameters. We emphasize that the FAST method does not require the analytic knowledge of the function $f(\cdot)$.

Various search functions have been proposed. The search function must let the parameter x_i to oscillate with frequency ω_i. For instance, the authors of [85] have proposed the search function

$$x_i = \frac{1}{2} + \frac{1}{\pi} \arcsin(\sin(\omega_i s)), \tag{3.1}$$

which is a particular case of a more general search function [72]

$$x_i = F_i^{-1}\left(\frac{1}{2} + \frac{1}{\pi} \arcsin(\sin(\omega_i s))\right), \tag{3.2}$$

where $F_i^{-1}(\cdot)$ is the inverse cumulative distribution function for x_i. To make more efficient use of the model evaluations, the authors of [85] have suggested the following slight modification

$$x_i = \frac{1}{2} + \frac{1}{\pi} \arcsin(\sin(\omega_i s + \varphi_i)), \tag{3.3}$$

where φ_i is a random phase-shift chosen uniformly in the interval $[0, 2\pi)$.

The model output becomes a periodic function with period 2π. Thus, we can represent the model with a Fourier series

$$y = f(x_1, x_2, ..., x_n) = A_0 + \sum_{k=1}^{\infty} [A_k \cos(ks) + B_k \sin(ks)].$$

If we denote a sample of size N as

$$S = \{s_1, s_2, ..., s_N\},$$

then, using either (3.2) or (3.3) as a search function, we can obtain the sampled values of the parameters

$$X_i = \{x_{i1}, x_{i2}, ..., x_{iN}\},$$

and the discrete Fourier transform coefficients

$$A_0 = \frac{1}{N} \sum_{j=1}^{N} f(s_j),$$

$$A_k = \frac{2}{N} \sum_{j=1}^{N} f(s_j) \cos(s_j k),$$

$$B_k = \frac{2}{N} \sum_{j=1}^{N} f(s_j) \sin(s_j k),$$

where $f(s_j) = f(x_{1j}, ..., x_{nj})$ and $k = 1, ..., (N-1)/2$.

The variance of the model output can be decomposed into variance components at the integer frequencies

$$V = \frac{1}{2} \sum_{k=1}^{(N-1)/2} [A_k^2 + B_k^2],$$

By summing the spectrum values $\Lambda_k = [A_k^2 + B_k^2]/2$ for the characteristic frequencies ω_i and their higher harmonics, the partial variance in model output arising from the uncertainty of parameter x_i, V_i, can be estimated by

$$V_i = \sum_{p} \Lambda_p \omega_i,$$

where $p\omega_i \leq (N-1)/2$. The ration V_i/V measures the contribution of parameter x_i. This ratio is also referred to as the first-order sensitivity index [87].

Because the characteristic frequencies are integers, there will be an aliasing effect if one frequency is a linear combination of the others. It is said that a frequency set is free of interferences to an order M if

$$\sum_{i=1}^{n} a_i \omega_i \neq 0,$$

$$\sum_{i=1}^{n} |a_i| \leq M + 1,$$

where a_i is an integer and M is a design integer (usually 4 or 6). In order to avoid the interference effect the maximal value of p in calculating V_i should be M. In [37] the authors have proposed the following empirical formula for calculating the characteristic frequencies free of interference up to order $M = 4$

$$\omega_1 = \Omega_n,$$

$$\omega_i = \omega_{i-1} + d_{n+1-i}, \quad i = 2, ..., n.$$

The parameters Ω_n and d_k can be found in a table provided in [37]. Below we give several line from that table.

Dimension, n	Ω_n	d_n	Minimal number of points, N_{min}
1		4	
2		8	
3	1	6	38
4	5	10	78
5	11	20	142

Then, for instance, for the case of four input parameters we obtain the following values of the characteristic frequencies

$$\omega_1 = \Omega_4 = 5,$$

$$\omega_2 = \omega_1 + d_3 = 11,$$

$$\omega_3 = \omega_2 + d_2 = 19,$$

$$\omega_4 = \omega_3 + d_1 = 23.$$

3.3 Conclusion

Confronted with the increasing trend and popularity of passive monitoring at multiple locations within an IP network, several monitoring systems have emerged. However, there is a lack of a universal experimental platform for monitoring applications. Given the inadequacy of current validation solutions, we have introduced in this chapter an exhaustive methodology for the evaluation of network monitoring solutions. We have implemented a real experimental platform for traffic sampling and monitoring using real traffic traces and real monitoring tools. We have also introduced a global analytical study based on sensitivity analysis to assess the performance of these systems and to characterize the impact of the different input parameters on their behavior.

4

System Architecture

Traffic measurement and monitoring are important activities in order to understand the performance of a network infrastructure and to efficiently manage network resources. However, the remarkable growth of the Internet infrastructure, the tremendous success of the Internet-based applications and their rapidly changing characteristics have made the management and monitoring of ISP networks a complex process. Therefore, the design of a new monitoring system that takes into account the requirements of multiple monitoring tasks and variations in the traffic is becoming an inevitable trend.

In this chapter, we present the design of an adaptive centralized architecture that provides visibility over the entire network through a network-wide cognitive monitoring system. Practically, given a set of measurement tasks (e.g. flow size estimation, flow counting) and a constraint on the volume of collected information, the proposed architecture drives the sampling rates on the interfaces of network routers to achieve the maximum possible accuracy, while adapting itself to any change in network traffic conditions.

Contents

4.1 Introduction

With the remarkable growth of the Internet connectivity in terms of infrastructure, capacity, size and the number of connected users, the networks operators are facing many problems challenges in order to determine the composition of network traffic, to monitor the performance of the deployed network infrastructure and to understand the behavior of users. In fact, users go from occasionally connected to always connected, the Internet infrastructure is also growing in capacity and in number of network elements. The Internet infrastructure growth makes its manageability and reconfigurability increasingly complex. It is thus expected that the monitoring and management cost of the Internet technology will start to increase more than proportionally to the number of nodes. This results in increasing complexity and decreasing maintainability of user's satisfaction while keeping an openly accessible, neutral, and generic Internet infrastructure.

In this context, the main objective of this thesis is to introduce a new monitoring system architecture based on a cognitive component that allows to transpose the high-level objectives and constraints. This new architecture will introduce a centralized adaptive architectural component that provides visibility over the entire network, improves measurement accuracy and respects monitoring constraints. The introduction of this cognitive engine implementing machine learning techniques is expected to improve and extend the overall Internet controllability capabilities as well as reducing their resulting cost.

4.2 Challenges and objectives

We illustrate the interest of our architecture with the help of NetFlow since it is the mostly deployed traffic monitoring tool. In its current deployment, NetFlow is activated on the edge routers such that each flow is captured one time in each direction. Moreover, each NetFlow router independently selects packets to monitor with a low sampling rate to satisfy local router resource constraints and to reduce the processing and computation cost. In practice NetFlow sets sampling rates to low values (between 0.01 and 0.001). However, this router-centric approach can cause an inefficient use of resources to make redundant measurements, since they are completely independent, and can lead to several local and noisy estimations, as the sampling rate should be very low.

The problem is then how to keep passive measurements technologically feasible and operate within the routers' constraints (i.e. deploy sampling rates at a low values) while improving the accuracy of measurements. We propose to adopt a centralized network-wide approach that explicitly coordinates monitoring responsibilities and shares the work between the different routers. Furthermore, it provides means to leverage the possibility of parallel measurements of network flows in the different routers they cross inside the network. Those parallel measurements of each flow can be combined together to provide a finer estimation of flow statistics while distributing the load over the different routers instead of focusing on the edge. Note that the load distribution comes from lowering the sampling rates in the routers. In addition, the fact that a flow can be monitored everywhere along its path leads to possible optimizations of the distribution of the sampling rates across routers that can satisfy both global and local objectives as improving the estimation accuracy and minimizing the load. That should give a richer configuration scheme where the standard configuration (i.e. deploying routers in edge routers) is no other than one particular case.

Two major challenges arise in the development of our approach:

- The first one is how to combine sampled and noisy traffic measurements, carried out in the different routers using various sampling primitives, in a way to provide network-wide sampled NetFlow records having a better accuracy than in the single edge router approach. This combination should be motivated by the need to minimize the amplitude of estimation errors given the monitoring objective.

- The second challenge we face is how to coordinate responsibilities across the different monitors (routers) in order to increase global accuracy while avoiding unnecessary measurements.

We argue that a centralized system that collects and combines local measurements and that coordinates monitoring responsibilities across different monitors can significantly increase the

flow monitoring capabilities of a network. In this thesis, we aim to build an adaptive system that realizes network-wide flow measurement objectives. Our system should adjust its configuration according to network conditions and measurements accuracy. Given a set of measurement tasks (in the form of a filter to apply on the collected traffic at the central unit) and a target overhead value $\mathcal{TO}$, defined as the desired total number of flow reports that can be exported in the entire network (processing and storage resources), our network-wide system should be able to:

- Collect and obtain data from different monitors using various sampling primitives deployed in network routers. All monitors integrate one or more sampling primitives for traffic capturing as well as reporting capabilities for exchanging information with the collector.
- Support a cognitive component that processes the collected measurements from the different routers, correlates them and calculates the targeted metric together with its estimated error. This component also measures the amount of overhead caused by the actual configuration of monitors (volume of collected data, CPU, memory, disk space, etc).
- Drive its own deployment by automatically and periodically reconfiguring the different monitors in a way that improves the overall accuracy (according to monitoring application requirements) and reduces the resulting overhead (respecting some resource consumption constraints).
- Support a general class of monitoring applications by using various sampling primitives.

In the next section we will present our system that achieves all these monitoring objectives and we will highlight its main components.

4.3 System architecture

In order to provide network-wide capabilities, we need a framework that assigns monitoring responsibilities across routers to satisfy network-wide monitoring goals. At the same time, our system should be resource-aware; i.e., respect the resource constraints (i.e. the maximum amount of exported data). As discussed in the previous section, we adopt a centralized network-wide approach that extends existing local monitoring tools with a central cognitive component. This component coordinates monitoring responsibilities across different routers in order to significantly increase the flow monitoring capabilities of a network. Such a centralized system simplifies the process of specifying and realizing network-wide flow measurement objectives. Moreover, by using different sampling primitives, the proposed system is capable to support a large spectrum of monitoring applications.

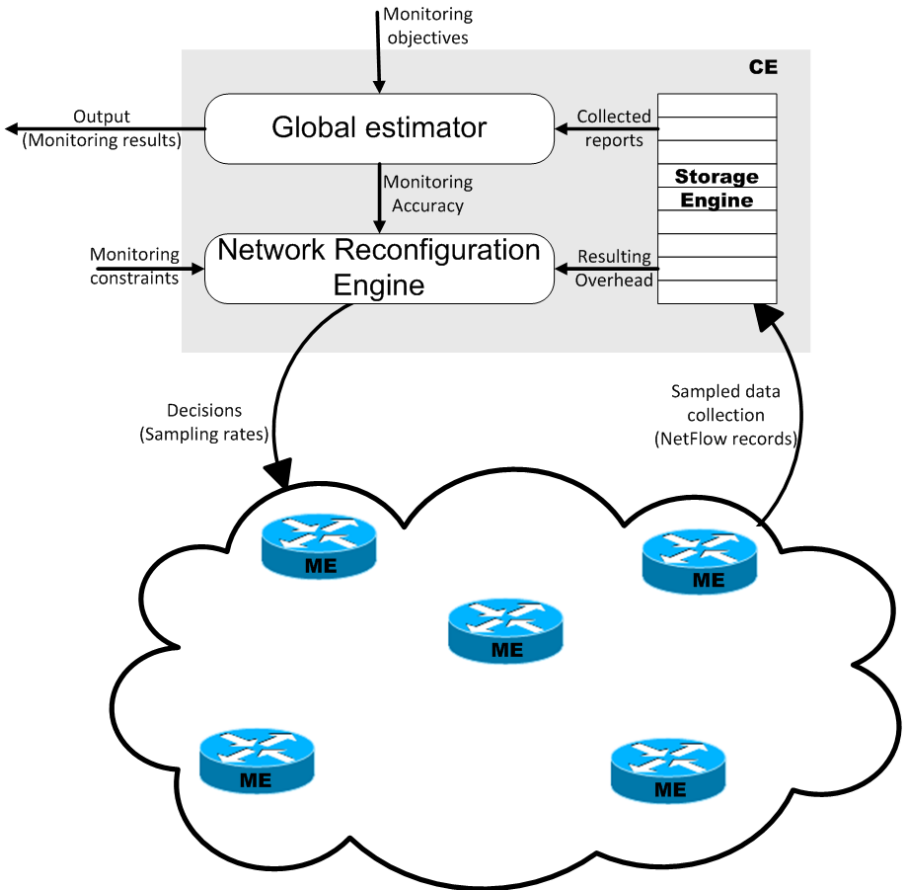


Figure 4.1: System architecture.

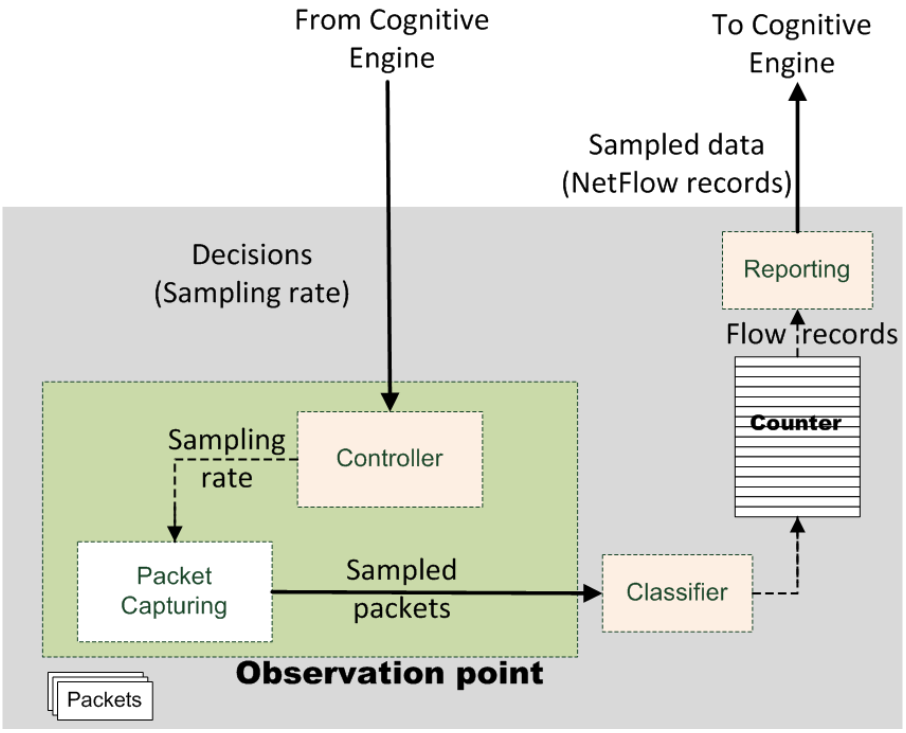


Figure 4.2: Functional architecture of the monitoring engine.

Figure 4.1 depicts the basic functional components of the proposed monitoring system and the interactions among them. This system relies on local NetFlow-like measurement tools (Monitoring Engine (ME)) deployed in network routers as well as on the reporting capabilities for exchanging information and decisions with the central unit (Cognitive Engine (CE)). The targeted applications provide the input to the system in the form of monitoring objectives to apply on the collected data at the central unit. The system adjusts the sampling rates in routers to answer the application needs with the best accuracy while respecting monitoring constraints. Next we give a detailed description of the architecture components.

4.3.1 Monitoring Engine (ME)

This engine runs in each router and aims at sampling and capturing traffic at the interfaces of the router using one or more sampling primitives. It then exports NetFlow records to the central collector. As depicted in Figure 4.2, one can observe four main modules:

- **Packet capturing:** This module listens to the network interface and samples data using a sampling primitive (e.g., packet sampling, flow sampling) at a given sampling rate. This sampling rate is configured each time by the Cognitive Engine (CE) next to the optimization it carries out after correlating measurements from all routers.

- **Classifier:** Once a packet is sampled by the packet capturing module, the classifier identifies flows by a key (in our case this key corresponds to the 5-tuple). The classifier then determines if a flow is active or if it is a new flow. If the flow is active, it updates real-time statistics on that flow such as the number of packets and bytes. If it is a newly observed flow, it inserts a new flow record for this new packet's key. The ME maintains the keys of flows forwarded by the router to the collector together with the statistics on those flows. A flow is declared finished by the classifier in one of three cases: *(i)* when observing a FIN or an RST packet (TCP control), *(ii)* when a timeout expires after the record for that flow was created, and finally *(iii)* when the number of records exceeds a given threshold requiring to release memory.

- **Reporting:** Once collected, flow records are exported using UDP messages to the Cognitive Engine via the CM (Cognitive-Monitoring) interface.

- **Controller:** Based on the collected data applies machine learning methods, the cognitive engine (CE) takes a decision on how to tune the sampling rates for the different sampling primitives and then sends it back to the ME in each router. The router controller receives the decision and updates its sampling rate accordingly.

4.3.2 Cognitive Engine (CE)

This component is motivated by the need to extend the local existing monitoring tools (MEs) with a network-wide cognitive engine able to:

- Investigate the measurements collected from the different routers (local views) and then construct a global view of the traffic and the network state.

- Automate and enhance network-wide monitoring control while decreasing their resulting cost. The automation of the control of sampling rates is achi-eved by learning experiences from the accuracy of the collected data and the resulting overhead.

As described in the previous section, our network-wide monitoring system adopts an adaptive centralized approach to coordinate responsibilities across monitors by adjusting their sampling rates. Our system extends the existing NetFlow-like monitoring tools with a cognitive engine that correlates collected measurements from all routers to infer a better global view of the network traffic. Moreover, it automatically reconfigures the sampling rates in the different monitors for different sampling primitives according to the monitoring application requirements and resource consumption constraints. The automation of the control of sampling rates is achieved by learning experiences from the accuracy of the collected data and the overhead of measurements. Figure 4.3 outlines the monitoring process of the cognitive engine. Given

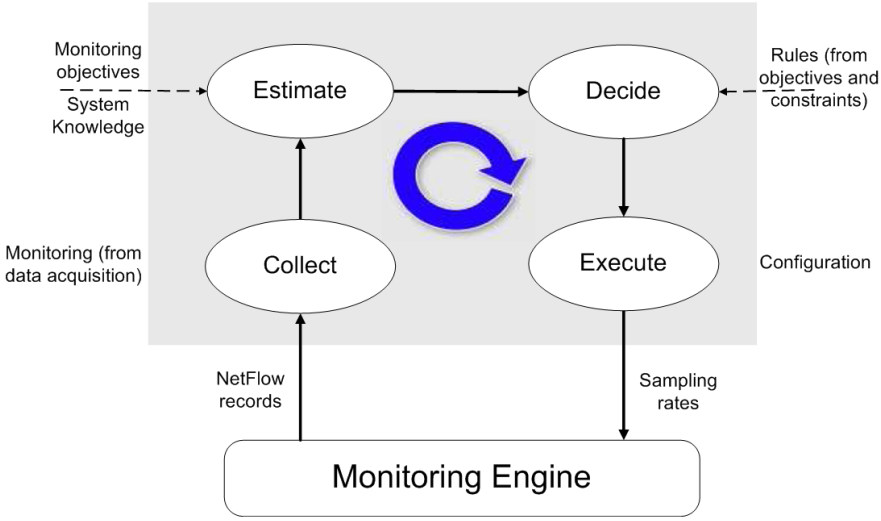


Figure 4.3: Monitoring process.

the set of monitoring objectives $\mathcal{T}$ to achieve and the monitoring constraints to respect ($\mathcal{TO}$), our adaptive finds the optimal configuration of network monitors that addresses the tradeoff between the monitoring accuracy and the resulting overhead.

Figure 4.3 highlights the main stages of the monitoring procedure. First, the cognitive engine starts by collecting NetFlow records from the different monitoring engines deploying different sampling primitives in the monitors inside the network. Once NetFlow records have been collected, the cognitive engine investigates these local estimators in order to provide visibility over the entire network. It provides means to combine the different local and noisy measurements to form reliable and accurate global estimations. In this way, our system leverage the possibility of parallel measurements of network flows in the different routers they cross inside the network. Those local and parallel measurements are combined together to provide a finer estimation. Next, our system drives its own deployment and configuration to the optimal by diagnosing the reported network traffic, learning about the status of flows and the accuracy of estimators, and taking the best adjustment decisions on the sampling rates in monitors that can satisfy both global and local objectives. Finally, the cognitive engine executes these decisions by deploying the new sampling rates in the different monitors.

The CE is composed of two main modules:

4.3.2.1 Network Reconfiguration Engine

This engine is motivated by the need to coordinate responsibilities across the different monitors according to network condition changes and measurement accuracy while respecting the monitoring constraints in order to improve monitoring capabilities and achieve network-wide

monitoring goals. The optimization methods should address a number of key concerns in the design of a network-wide monitoring system including responsiveness to changes in network conditions, avoidance of the introduction of unnecessary noise and robustness of monitoring solutions. While the main goal is to provide a responsive control that quickly react to network condition changes, the smooth control exhibits contained oscillations and avoid the introduction of unnecessary details and information in network traffic in order to keep resource consumption within a desired bound.

Therefore, the optimization algorithm for configuring monitors and periodically calculating the sampling rate vector is a key design issue in network-wide monitoring system. It determines the tradeoffs between responsiveness to network condition changes and the avoidance of oscillations or unnecessarily abrupt shifts in the network traffic.

We provide two different network configuration methods:

- In a first method we provide a reactive method inspired from the dynamics used by TCP for the adjustment of its congestion window. We continuously control the amount of collected overhead. Then, we propose one of two actions: *(i)* either to increase the sampling rate of the different monitors when the resulting overhead is lower than the $\mathcal{TO}$, *(ii)* or in the opposite case to decrease the sampling rate of the least significant monitors. The least significant monitors are those providing the minimum loss when playing with their sampling rates and they are identified using the gradient projection method (GPM) [1].

- The second method presents a proactive approach that provides smoother control of sampling rates based on overhead prediction. In order to smooth the reaction to network changes, we predict the values for the overhead and determine the sampling rate vector accordingly. We smooth the predicted overhead using an exponentially weighted moving average. The weight of the prediction determines the responsiveness of the sampling rates to changes.

4.3.2.2 Global Network Traffic Inference Engine

This component is motivated by the need to extend local existing monitoring tools (MEs) with a network-wide engine that combines their measurements to support a large spectrum of applications and provide more accurate results. The global estimator module relies on a

[1] Gradient Projection Method (GPM) is among the most widely used of all function approximation methods and is particularly well suited to reinforcement learning. This optimization method uses the derivative of the function and the idea of steepest descent. The derivative of a function is simply the slope. GPM is an attractive optimization method in that it is conceptually straightforward and often converges quickly. Its drawbacks include the fact that the derivative of the function must be available, and it converges to a local minimum rather than a global minimum

Table 4.1: Parameters in the model

Parameter	Definition
$\mathcal{T}$	Set of measurement tasks
$\mathcal{M}$	Set of network monitors
$\mathcal{S}$	Set of sampling primitives
p_k^s	Set of sampling rates of the sampling primitive s in the monitor k
V	The global normalized estimation variance
$\hat{T}_{ik}^s$	Set of estimations of the task T_i done in the monitor k using the sampling primitive s
V_{ik}^s	Set of normalized estimation variances of the tasks (T_i) done in the monitor k using the sampling primitive s
$\mathcal{TO}$	The target overhead value (monitoring constraint)
SR_{min}	Minimum sampling rate value
SR_{max}	Maximum sampling rate value
d	Computation period

set of measurement tasks $\mathcal{T}$ to realize, a set of network monitors $\mathcal{M}$, and a set of sampling primitives $\mathcal{S}$ deployed in the different monitors. It investigates the local measurements made by the different routers using various sampling primitives to have a global and more reliable view. We summarize the important flow and monitoring parameters in Table 5.4.

4.3.2.2.1 Estimation procedure Consider the set of available sampling primitives $\mathcal{S}$. Let $(p_k^s)_{(k\in\mathcal{M},s\in\mathcal{S})}$ be the sampling rate vectors of the different sampling primitives in the different monitors. The objective of the cognitive engine is to find the optimal sampling rate vectors that minimize the global normalized estimation variance V (variance of estimator).

Local estimation

For each monitoring task $T_i \in \mathcal{T}$ and given the sampling rate value p_k^s of the sampling primitive $s \in \mathcal{S}$, the monitor k samples data according to the sampling rate value and derives a local estimation $\hat{T}_{ik}^s$ of the task T_i. Moreover, it calculates analytically its corresponding local normalized estimation variance V_{ik}^s.

Combining measurements

The central unit collects the different local estimations $\hat{T}_{ik}^s$ with their corresponding normalized variances V_{ik}^s and combines them in order to get a global estimation for each sampling primitive $\hat{T}_i^s$. Furthermore, it calculates their normalized estimation variances V_i^s.

Global estimation

In order to support a large spectrum of monitoring applications and to minimize the vari-

ance of the global estimation error, we combine the global estimations of the different sampling primitives $\hat{T}_i^s$ to construct the global estimator of the task T_i. This global estimator is defined as a weighted sum of the sampling primitive estimators. This weighted summation of the different estimators is known to be the best linear combination in terms of mean square error [48],

$$\hat{T}_i = \sum_{s \in \mathcal{S}} \chi_s \hat{T}_i^s \quad \text{with} \quad \chi_s = \frac{\frac{1}{V_i^s}}{\sum_{l \in \mathcal{S}} \frac{1}{V_i^l}}. \tag{4.1}$$

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

Monitors reconfiguration

The monitor reconfiguration module takes as input the global estimator variance V and the sampling rate vectors in order to identify the optimal sampling rate vectors. It calculates the marginal loss or gain in the global accuracy when reconfiguring a sampling rate value. Then, it finds the best sampling rates vectors by testing all monitors [70, 61, 69].

In order to explain the operation of our system and how it configures the different monitors and combines their local measurements, we explain in the next section the estimation of flow sizes using a single monitoring primitive: the packet sampling. We consider in this section the reactive network reconfiguration method.

4.3.2.2.2 Case study: Traffic accounting We explain in this section using a concrete example how the central cognitive engine, based on the collected measurements, can decide on the way to tune the sampling rates over the network using the reactive network reconfiguration method. We consider for this purpose an accounting application: the estimation of the volume of some chosen network flows [2]. Given a set of flows to monitor, the cognitive engine should progressively tune the sampling rates in routers in such a way to minimize the global estimation error.

Definitions

Consider N traffic aggregate flows whose volumes in packets are labeled F_1, F_2, ..., F_N. Denote by $\hat{F}_1$, $\hat{F}_2$, ..., $\hat{F}_N$ the corresponding estimators. Let $P = (p_k)$ be the vector of packet sampling rates in the different monitors of the network (a monitor is equivalent to a router interface). There are in total M monitors. The target of the system is to find the vector P that

[2] A flow is an aggregate of packets sharing some common features. The basic definition is what is called the 5-tuple definition where packets share the source and destination IP addresses and port numbers plus the protocol number.

minimizes the sum of normalized estimation errors

$$\sum_i Var(\hat{F}_i)/F_i^2.$$

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. One can then transform the optimization problem into minimizing the sum of the normalized estimation errors of the sizes of the 5-tuple flows.

$$\sum_i \sum_j Var(\hat{S}_{ji})/F_i^2$$

As long as there are available resources (i.e. The resulting overhead is less than the TO threshold), the system periodically increases all sampling rates to improve results' accuracies. Once the $\mathcal{TO}$ value is reached the system triggers a decrease in the sampling rate of the least significant monitors. This continues until the overhead is again below this $\mathcal{TO}$ value. The least significant monitors are the ones having the smallest absolute values for the following partial derivative sum:

$$\sum_i \sum_j \frac{\partial Var(\hat{S}_{ji})}{\partial log(p_k)} . \frac{1}{F_i^2}. \tag{4.2}$$

In the following we show how such estimators for the 5-tuple flow sizes are formed and how the partial derivatives of their variances are obtained. For the F_i themselves, which are unknown, we simply substitute them by their estimations, i.e. $\hat{F}_i = \sum_j \hat{S}_{ji}$. Note that we consider the volumes of flows as measured in packets. The passage to bytes can be made by multiplying the size in packets by the average packet size, which we suppose true for large flows.

Local flow size estimation

Consider a 5-tuple flow S_{ji} crossing monitor k whose sampling rate is p_k. Let s_{kji} be the number of packets sampled from this 5-tuple flow in the monitor (this number could be zero). With this information, one can derive a first estimation for the flow size of the flow. The estimator that maximizes the likelihood is known to be [41]:

$$\hat{S}_{kji} = s_{kji}/p_k. \tag{4.3}$$

Under independent sampling of packets with probability p_k, the number of packets s_{kji} sampled from an original 5-tuple flow S_{ji} follows a binomial distribution whose variance is well known and equal to $S_{ji}.p_k.(1-p_k)$. It follows that this local estimator for the size of a 5-tuple flow has a variance equal to

$$Var(\hat{S}_{kji}) = S_{ji}.(1-p_k)/p_k. \tag{4.4}$$

Combining measurements

The information on a 5-tuple flow comes from all monitors along its path. Though, some of them may not sample any of the packets of the flow, either because their sampling rate is low, or because the volume of the 5-tuple flow is small in terms of packets. We propose to identify these monitors related to a 5-tuple flow with the help of routing information. Largely deployed link-state protocols like OSPF and IS-IS provide such information. If such routing information is not available at the central unit, one has to limit the observations to monitors that have seen the flow knowing well that this might cause a bias against 5-tuple flows that got unsampled. This bias is expected to be small when aggregating over aggregate flows F_i.

According to Equation (4.1), we estimate the volume of a 5-tuple flow as being the sum of the weighted sum of the local estimators done in the monitors along its path. This gives the following global estimator for 5-tuple flow j belonging to aggregate flow F_i,

$$\hat{S}_{ji} = \sum_{k \in \varphi_{ji}} \lambda_k s_{kji}/p_k, \quad \text{with} \quad \lambda_k = \frac{\frac{1}{Var(\hat{S}_{kji})}}{\sum_{l \in \varphi_{ji}} \frac{1}{Var(\hat{S}_{lji})}}, \tag{4.5}$$

φ_{ji} is the set of monitors on the path followed by S_{ji}. Replacing the variances by their expressions given in the previous section, substituting the second equation into the first one, and simplifying by S_{ji}, we get,

$$\hat{S}_{ji} = \frac{1}{\alpha_{ji}} \sum_{k \in \varphi_{ji}} \beta_{kji} s_{kji}, \tag{4.6}$$

with

$$\alpha_{ji} = \sum_{l \in \varphi_{ji}} \frac{p_l}{(1-p_l)} \quad \text{and} \quad \beta_{kji} = \frac{1}{(1-p_k)}$$

Note in particular how the α_{ji} and the β_{kji} are the same for all 5-tuple flows that follow the same path, which eases a lot the calculation. As for the variance of this estimator of 5-tuple flow sizes, it is simply equal to

$$Var(\hat{S}_{ji}) = S_{ji}/\alpha_{ji}. \tag{4.7}$$

The original flow size being unknown, we can simply substitute it by its global estimator $\hat{S}_{ji}$.

Reconfiguring monitors

As shown in the previous section, the variance (or mean square error) of 5-tuple flow size estimation is very important for the determination of the global system accuracy and for the identification of the monitors that should be reconfigured. For 5-tuple flow S_{ji} and monitor k we can write,

$$\frac{\partial Var(\hat{S}_{ji})}{\partial log(p_k)} = \frac{-S_{ji}.p_k}{\alpha_{ji}^2(1-p_k)^2}. \tag{4.8}$$

This represents the marginal gain in the accuracy (loss in the variance) when the logarithm of the sampling rate of monitor k is increased by a small step δ and this is from the perspective of estimating the size in packets of flow S_{ji}. As expected, this gain is positive when someone

increases the sampling rate (more sampling means more accuracy). It also decreases when p_k increases, which suggests that the estimation error follows well a continuously decreasing and convex function with the sampling rate, a condition required for the uniqueness of solution in non-linear optimization theory.

By plugging the above expression in Equation (4.2) we obtain the utility function of the monitor k, which sums the accuracy and normalizes it over all 5-tuple flows forming the traffic of interest, we can easily find the total gain (resp. the loss) in accuracy when the sampling rate of monitor k is tuned up (resp. down) by a multiplicative step (additive in the logarithmic scale). By testing all monitors, we can find the best sampling rates to tune down in case of saturation. We choose to decrease the monitors having utility function values less than the average over the different monitors. Note that the sum in (4.2) can be calculated online as long as more reports are received. The parameters α_{ji} can be calculated only once for each configuration and for all possible paths across the network.

4.4 Conclusion

There is an increasing interest in passive monitoring systems. Existing solutions, however, are inadequate for monitoring application requirements and fail to meet the increasing demands for fine-grained flow-level measurements. To meet these growing demands, we argue the need for a centralized monitoring system that takes a network-wide approach to flow monitoring.

In this chapter, we have presented a network-wide monitoring system that adopts an adaptive centralized approach to coordinate responsibilities across the different monitors by adjusting their sampling rates. Our system extends the existing NetFlow-like monitoring tools with a cognitive engine that correlates collected measurements from all routers to infer a better global view of the network traffic. Moreover, it automatically reconfigures the sampling rates in the different monitors according to the monitoring application requirements, resource consumption constraints and network condition changes. After a general presentation, we have described the details of our system by the help of a traffic accounting application, i.e. estimating the volume of some well defined flows. Furthermore, we have discussed the different needs that address a number of key concerns like addressing the tradeoff between improving monitoring accuracy, respecting monitoring constraints and reacting to network condition changes.

In the next chapter, we will describe two different approaches for network monitoring reconfiguration. We will illustrate our work with more monitoring applications and primitives. Moreover, we will provide experimental results to validate the functioning of the proposed system and methods.

5

NETWORK RECONFIGURATION METHOD

As we have seen in the previous chapter, an important part of the proposed monitoring system architecture is dedicated to the reconfiguration of monitors according to measurements accuracy and network condition changes. The optimization methods should address the trade-off between responsiveness to changes in network conditions and smoothness defined as avoidance of unnecessary oscillations and avoidance of the introduction of unnecessary noise and robustness of monitoring solutions. In this Chapter, we will provide two different network reconfiguration methods: A *reactive optimization method* and a *proactive optimization method*. We will give an exhaustive description of each method. Furthermore we will study their performance using a single sampling primitive, packet sampling and two monitoring applications, flow size estimation and heavy hitter detection. We will validate the operation of each method and we will compare their performances.

Contents

5.1 Reactive Network Reconfiguration Method

A key characteristic of adaptive monitoring systems is their responsiveness to network conditions changes and measurement accuracy requirements. This means that the proposed monitoring systems should rely on a responsive optimization method. This responsive control should update its configuration as a result of experiencing resulting overhead made by the storage engine and measurement accuracy calculated by the global estimator. The degree to which the

optimization method updates its sampling rates should depend on the value of these two metrics.

The goal of this section is to introduce a reactive optimization method able to continuously and gradually react to monitoring requirements. The degree of this reaction should depend on the aggressiveness of changes of network conditions and measurement accuracy.

We propose a network-wide cognitive monitoring system that profits from advances in machine learning techniques and flow-level monitoring tools. The system starts from the NetFlow monitoring capabilities deployed in routers, tunes them in an adaptive and optimal way, and combines their results to answer to the best the monitoring application needs. We aim at finding the best configuration of sampled NetFlow that provides the best accuracy while respecting router and collector resource constraints. Our system is centralized and proceeds in optimizing the configuration in small steps based on collected reports from inside the network until the optimal configuration is reached. Our system can drive its own deployment and configuration to the optimal configuration by diagnosing the reported network traffic, learning about the status of flows and the accuracy of estimators, and taking the best adjustment decisions on the sampling rates in monitors.

5.1.1 Optimization method description

This engine is motivated by the need to coordinate responsibilities across the different monitors in order to increase the global accuracy while avoiding unnecessary measurements. To do so, we proceed by an adaptive centralized control of sampling rates based on the estimation of the measurement error and the reporting overhead (as shown in Algorithm 1). We resort to a dynamics inspired from the one used by TCP for the adjustment of its congestion window. Starting from an initial sampling rate vector $\mathcal{P}_{\mathrm{init}}$, the Network Reconfiguration Engine is fed with the estimation of the different tasks (T_i) and their corresponding errors (E_i), as well as the resulting overhead ($\mathcal{O}$), i.e. rate at which flow records arrive. If the overhead $\mathcal{O}$ is less than the target overhead $\mathcal{TO}$, the system keeps increasing periodically (each time period d) the sampling rates of the different monitors. Once $\mathcal{TO}$ is reached, the system triggers a decrease in the sampling rates of the least significant monitors. In this way the system strives to keep the reporting overhead at $\mathcal{TO}$ flow records per second and fully profits from the available resources. Note that setting the sampling rate to a very low value in a router (SR_{min}) is equivalent to turning it off for the purpose of monitoring while we don't let the sampling rate exceeds some maximum value (SR_{max}) to respect local router constraints.

To increase or to decrease sampling rates, we use increments in the logarithmic scale in order to give more flexibility to our system and to get a fast scan of the sampling rate interval $[0, 1]$. For reconfiguring the sampling rate of, let's say monitor k and sampling primitive s, we set $\log(p_k)$ to $\log(p_k) \pm \delta$ depending on where the decision is to increase or decrease p_k. This

gives in the normal scale $p_k = p_k(\gamma)^{\pm 1}$ where $\gamma = exp(\delta)$. In our experiments, we measure $\mathcal{O}$ and we set the value of γ at $\min\{1 + \sigma|\frac{\mathcal{TO}-\mathcal{O}}{\mathcal{TO}}|, 3\}$, so that this value varies between 1 and 3. $\mathcal{O}$ is the number of flow records received since the last update, divided by the time since this last update. It is immediately noticed that the value of γ depends on the value of $\mathcal{O}$: small adjustments when $\mathcal{O}$ converges to $\mathcal{TO}$ and large adjustments when $\mathcal{O}$ deviates from $\mathcal{TO}$. σ is a constant parameter of the control that represents a balance between convergence speed and stability.

As described in 4, the least significant monitors are identified using the Gradient Projection Method (GPM). From the perspective of the set of tasks $\mathcal{T}$ to realize, the least significant monitors are the ones providing the least increase in the global estimation variance V, when the logarithmic of their sampling rates are decreased by step δ. To be identified, one has first to write analytically the expression of V as a function of the sampling rates in routers of interest, then calculate the utility function of the different monitors by differentiating this expression with respect to $log(p_k)$, where p_k is the sampling rate of the monitor k: $U_k = |\partial E/\partial log(p_k)|$, $k = 1 \dots M$. Given the current configuration of sampling rates, we choose the least significant monitors as being those having utility function values less than the average of the utility functions values over all the monitors.

5.1.2 Validation results

In order to explain the operation of our system and how it configures the different monitoring primitives and combines their measurements, we consider two monitoring applications: flow size estimation and heavy hitter detection. The analysis and validation are done using a single sampling primitives: packet sampling.

We divide the validation results section into three parts. First, we study the efficiency and convergence of our adaptive solution and its ability to adapt to the heterogeneity of flow rates and to the predefined collected traffic overhead. Second, we show the practical benefits of deploying our optimization approach by comparing it to the common static configuration approach where sampling is only performed at the edge of the network. Last but not least, we present a global sensitivity analysis of the importance of the different parameters of our algorithm and we calculate their influence on the system behavior.

5.1.2.1 Validation scenarios

MonLab [8, 68] requires the definition of a network topology over which it dispatches and replays a real traffic. This topology is supposed to connect an AS at each of its POP (Point-Of-Presence) routers. We chose to experiment over network topologies similar to the ones of well known tier-1 transit networks. Two topologies, described in Figures 5.1 and 5.2, were chosen

Data: The global estimation $\hat{T}$ with its estimation variance V, and the sampling rate vector $\mathcal{P}$
Result: The new sampling rate vector $\mathcal{P}$
begin
  Initialize the sampling rate vector at $\mathcal{P}_{init}$;
  $\mathcal{P} \leftarrow \mathcal{P}_{init}$;
  while True **do**
    /* If $\mathcal{O}$ exceeds $\mathcal{TO}$, the system triggers a decrease in the sampling rates of the least significant monitors */
    if $\mathcal{O}$ *exceeds* $\mathcal{TO}$ **then**
      calculate $\gamma = \min\{1 + \sigma|\frac{O-TO}{TO}|, 3\}$;
      foreach $p_k \in \mathcal{P}$ **do**
        calculate $U_k = |\partial E/\partial log(p_k)|$;
      end
      calculate $AvgUtility = Avg_{p_k \in \mathcal{P}} U_k$;
      foreach $p_k \in \mathcal{P}$ **do**
        if $U_k < AvgUtility$ **then**
          $p_k \leftarrow \max\{\frac{p_k}{\gamma}, SR_{min}\}$;
        end
      end
      return $\{\mathcal{P}\}$ rst(d, $\mathcal{O}$) ;
    end
    /* If d expires, we increase the sampling rate of the different monitors.*/
    if d *expires* **then**
      calculate $\gamma = \min\{1 + \sigma|\frac{O-TO}{TO}|, 3\}$;
      foreach $p_k \in \mathcal{P}$ **do**
        $p_k \leftarrow \min\{\gamma p_k, SR_{max}\}$;
      end
      return $\{\mathcal{P}\}$ rst(d, $\mathcal{O}$) ;
    end
  end
end

Algorithm 1: The adaptive centralized control algorithm.

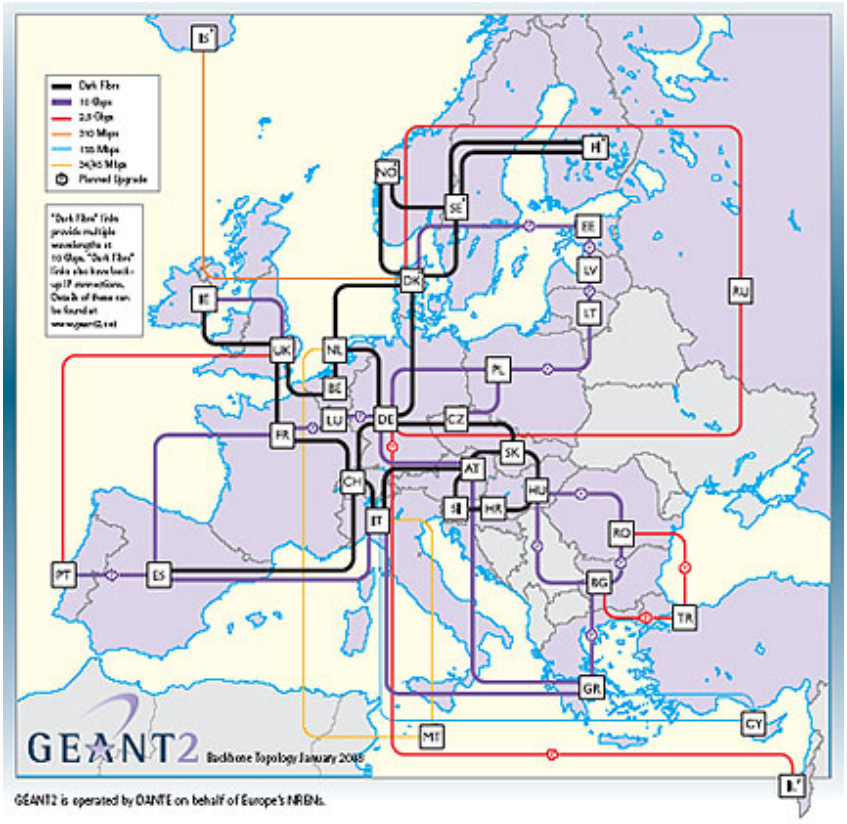


Figure 5.1: Geant topology.

for their widely use, the Geant topology (TOPG) [4] and the Abilene one (TOPA) [1].

The weights of ASes needed for traffic dispatching are set according to the sizes of stub ASes in Geant and Abilene (we make sure these weights sum to 1). An AS of weight w will then see itself attributed $100.w\%$ of the prefixes available in the trace and will see its traffic (ingoing 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, we replay over each emulated topology different traces collected at a transpacific link by the Japanese MAWI working group [7]. Traffic traces are made by TcpDump, and then, IP addresses in the traces are scrambled by a modified version of Tcpdpriv [13]. The default scrambling configuration preserves network prefixes and IP address classes. In this paper, we present results for two traces among the many ones in this data archive: Trace S collected on 03/03/2006 during the night making the traffic relatively smooth, and Trace V collected on 03/03/2006 during the day featuring more important traffic variability. Table 5.1 provides summary information on these two traces.

We run the three services of the MonLab platform on one machine each. Machines are fast

Figure 5.2: Abilene topology.

Table 5.1: Traffic traces summary

Trace	Start time	End time	Avg Rate	# of flows	# of pkts
S	00:30	02:30	26.34 Mbps	3250616	56178542
V	13:00	15:00	30.26 Mbps	3278041	69499589

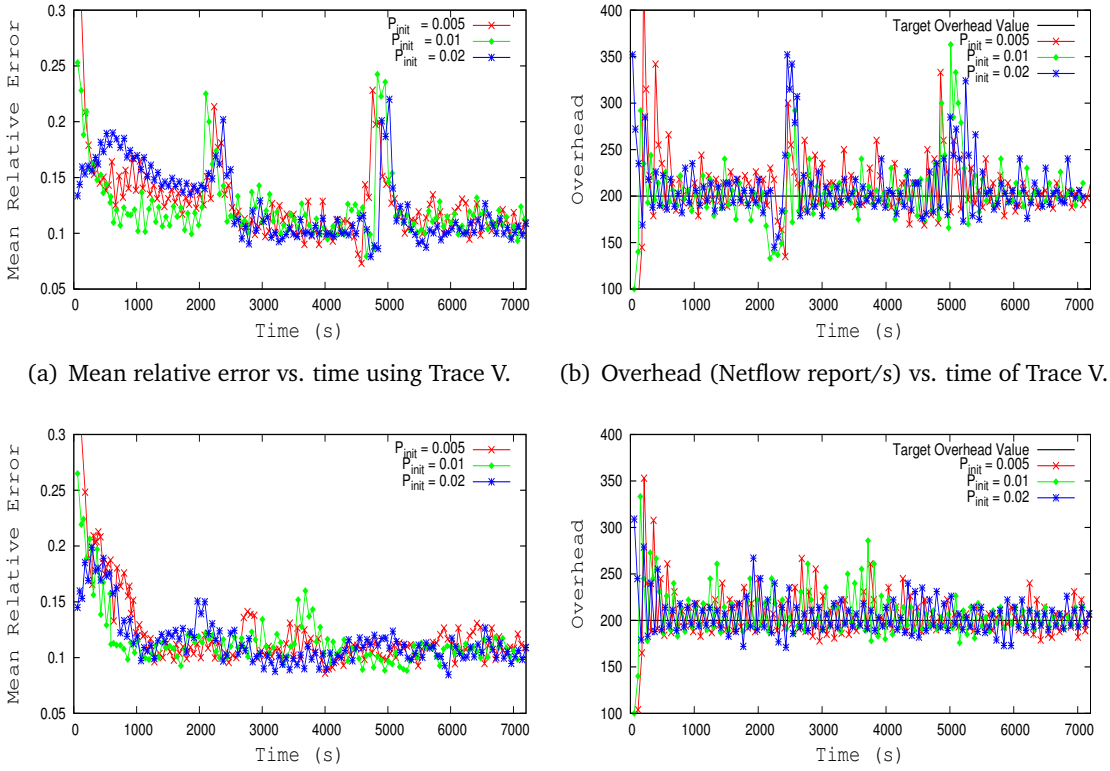
enough to follow in real time the stream of packets in the replayed TcpDump traces. There is one machine for dispatching and replaying traffic, a second machine for topology emulation and flow monitoring, and a third machine for measurement collection. This latter machine emulates the central unit; it collects NetFlow reports and implements the sampling rate adaptation algorithm.

As target application, we consider the estimation of flow sizes as described in Section 4.3.2.2.2. We recall that a flow F_i is 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. The target of the system is to find the vector P that minimizes the sum of normalized mean relative errors.

5.1.2.2 System efficiency, adaptability and convergence

In this section we aim to address the following points:

- *Convergence:* Starting from any initial configuration value P_{init} of all sampling rates (usually a low value), we want to know if our system is able to converge to an equilibrium in its configuration and hence in the realized monitoring accuracy. To detect this equilibrium, we will experiment the same scenario for different initial sampling rates and check



(a) Mean relative error vs. time using Trace V.

(b) Overhead (Netflow report/s) vs. time of Trace V.

(c) Mean relative error vs. time using Trace S.

(d) Overhead (Netflow report/s) vs. time of Trace S.

Figure 5.3: The evolution of the mean relative error and the resulting overhead $\mathcal{O}$ (Netflow report/s) using two different traces.

the final state of the system. The system converges when the mean relative error stabilizes and stops improving.

- *Reactivity:* Any change in the network traffic should bring the system temporary out of its convergence state before it converges again toward a new equilibrium. We want to observe if our system can detect changes in the network traffic and if it is able to find quickly this new equilibrium.

- *Target Overhead:* We want to test the capability of our system to respect the imposed constraint on the rate of measurement records. Among the set of configurations resulting in a rate of records equal to the imposed constraint, the system should be able to find the one minimizing the error on the target measurement task.

The above points will be addressed next by real experiments over the two network topologies Geant-like and Abilene-like and by the help of the two traces S and V described in Table 5.1.

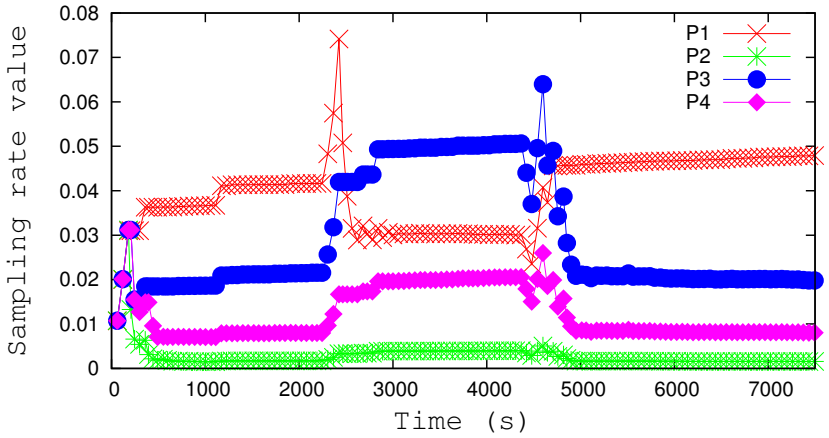


Figure 5.4: Evolution of some sampling rates vs. time using trace V and TOPG.

In Figure 5.3, we plot the evolution of the mean relative error obtained over all AS-to-AS flows (on the left hand side) and the resulting overhead in NetFlow-records/s (on the right hand side) over time using the two traces S and V. Each point in the graphs corresponds to an update of the sampling rates, either in the increase (overhead $\mathcal{O}$ less than the target value $\mathcal{TO}$) or in the decrease (overhead $\mathcal{O}$ larger than the target value and the buffer $\mathcal{B}$ is full). For this experiment, we set the timer d for updating sampling rates to 1 minute, the regulator σ to 2, 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.

Three initial sampling rates are considered: 0.005, 0.01 and 0.02. We can immediately observe that the system keeps improving the global accuracy while fully profiting from the available resources for measurement collection. At the beginning, the system exponentially increases sampling rates until the $\mathcal{TO}$ is reached. Once done, it keeps improving the accuracy of the estimation while maintaining the overhead around its target value. After few iterations, the system reaches an equilibrium where the mean relative error tends to oscillate around its minimum value. For the smooth trace S, the equilibrium does not change much along the trace. For the other variable trace V however, we can see in the middle of the trace sudden increases in the error caused by sudden changes in the traffic. The system adapts to these changes by recalculating a new optimal configuration, always at a constant overhead. Note how the behavior is almost identical for the three initial sampling rates illustrating the stability of our system and its ability to converge in few iterations (few minutes here) to an equilibrium that only depends on traffic conditions and monitoring target and not on the initial configuration of sampling rates. These results are illustrated in Figure 5.4, presenting the evolution in different routers of sampling rates over the time starting from an initial configuration P_{init} equal to 0.005 and using the trace V. We can observe the ability of our system to converge to an equilibrium

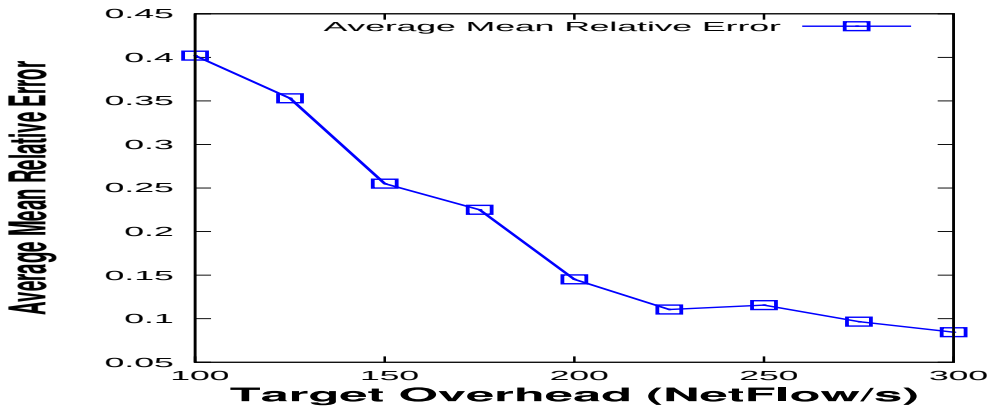


Figure 5.5: Average mean relative error for different TO values.

in its configuration. Once done, it keeps oscillating around this optimal configuration until the network conditions change. Moreover, we notice the capability of our system to track any change in the network conditions as well as to adapt sampling rates to move smoothly towards a new optimal configuration.

Figure 5.5 shows the value of the mean relative error for different target overhead values.

Each point in the graphs corresponds to an experimentation. We run the experimentation while calculating periodically the MRE. Then, we calculate the mean value of these calculated MRE values to get the Average Mean Relative Error (AMRE). These results are for topology TOPG and Trace V. One can immediately notice the impact of the TO on the traffic estimation accuracy. There is a clear reduction of the overall measurement error from 0.402 for a TO equal to 100 NetFlow-records/s, to 0.08 for a TO equal to 300 NetFlow-records/s. Indeed, for each TO value, the system tries to find the best configuration that minimizes the traffic estimation error. When TO is low, the system has to lower the sampling rates in the least significant monitors with the objective to reduce the rate of collected measurement records without much compromising the estimation accuracy. Allowing more overhead gives the system more freedom in increasing the sampling rates of the most significant monitors looking for better estimation of the sizes of the target flows. The main strength of our system is that it is able to cope with any TO value and provides for this value the best configuration of monitors. Now, this configuration might not satisfy the administrator in terms of the accuracy of the measurement, in this case the only remaining solution is to increase the value of TO. In a future research we will be working on an enhanced version of our system that adapts the TO in such a way to realize the measurement task with some predefined minimum accuracy. For now, we suppose the TO is a constraint set by the administrator and we let our system find the best configuration that maximizes accuracy.

To illustrate the capacity of our system to maintain the measurement overhead around the TO, we plot in Figure 5.6 the measured overhead in terms of collected NetFlow-records/s as a

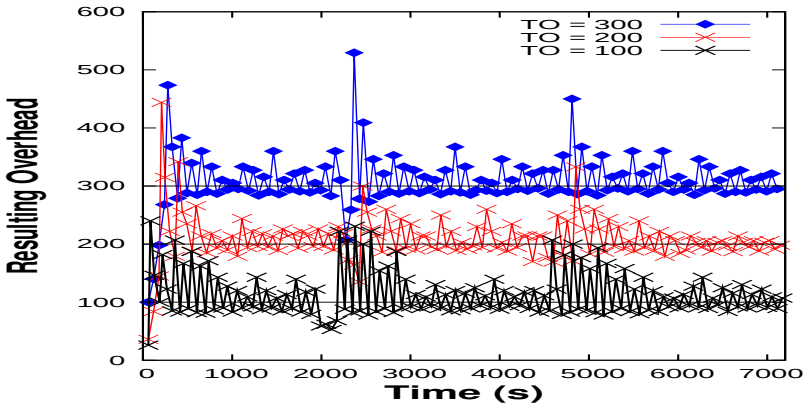


Figure 5.6: Resulting overhead vs. time using three different TO values.

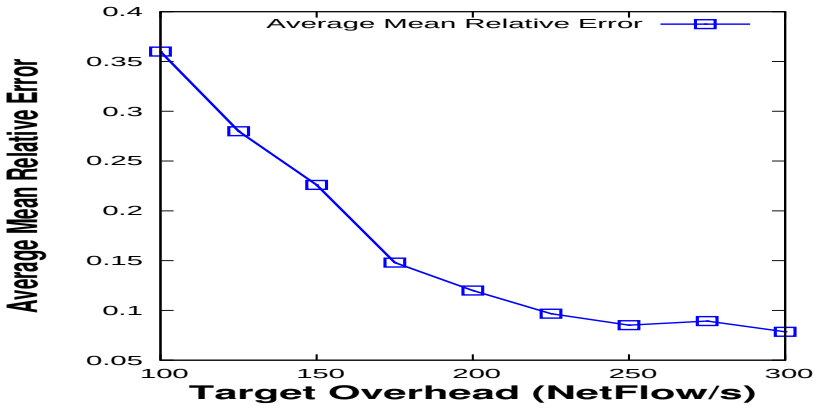


Figure 5.7: Average mean relative error vs. target overhead for the Abilene-like topology.

function of experimentation time and this is for three TO values. We can clearly see how for each experiment, whose traffic estimation accuracy is reported in Figure 5.5, the real overhead is maintained around the target value and how our system is able to converge and adapt to variations in traffic conditions along the trace lifetime.

The experiments over the Abilene-like topology confirm the same findings about the performance of our system. To give a sample of the obtained results, we plot in Figure 5.7 the mean relative estimation error averaged over all flows as a function of the TO. This figure is the equivalent to Figure 5.5 for the Geant-like topology. We can notice how the two figures look the same. The error for the Abilene-like topology is slightly smaller which comes from the smaller size of this topology and hence the larger volume of flows. Note that both experiments are conducted (on TOPG and TOPA) at equal total traffic driven by the same Trace V.

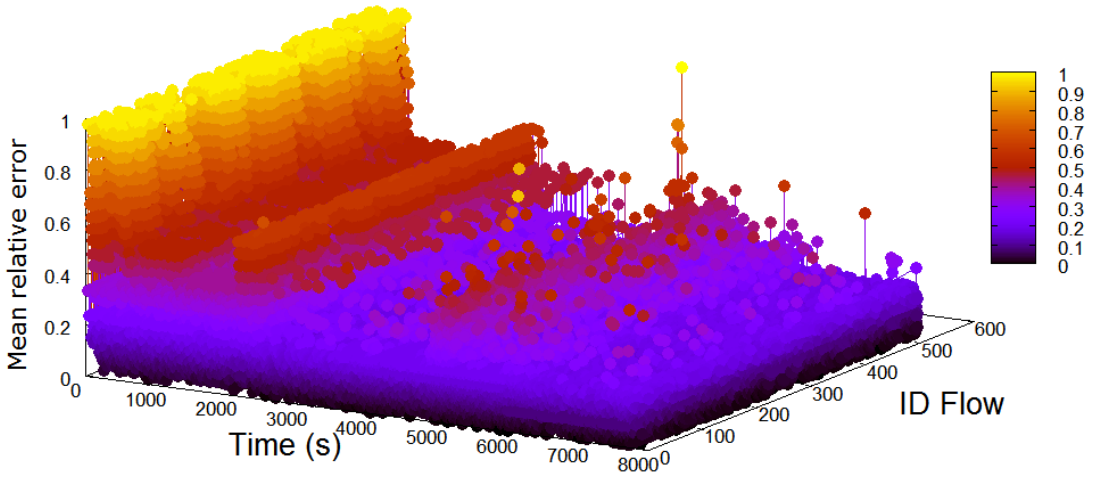


Figure 5.8: The evolution of the mean relative error of all the flows vs. time.

5.1.2.3 Fairness and comparison with the static edge method

We argue that an adaptive system that coordinates sampling responsibilities between the different monitors can considerably improve the flow monitoring capabilities of the network. In this section, we are interested in comparing our adaptive solution with the standard static configuration of NetFlow in order to assess the ability of our system to avoid unnecessary measurements while tracking efficiently the target flows at constant overhead.

By minimizing the sum of estimation relative errors for flow sizes, our system gives the same weight to each flow independently of its volume. This should naturally lead to a fair allocation of sampling rates that homogenizes estimation errors over target flows. This ability to track indifferently small and large flows constitutes one of the main strengths of our system. Any static configuration of sampling rates does not provide this fairness feature. Note that we are talking here about aggregate flows F_i. Each aggregate flow is composed of a set of 5-tuple flows whose total volume is estimated.

In order to illustrate the fairness of our approach with respect to target flows, we plot in Figure 5.8 the evolution of the mean relative error of all the flows over time. Starting from a large mean relative error, we can clearly note how our system keeps reducing this estimation error for all flows at almost the same rate even though these flows span different volumes. Some of the flows unfortunately still suffer from a large relative error because of their very small volume.

Then, we move to the comparison of the performance of our system with the widely deployed NetFlow solution, which consists of monitoring traffic at the edge of the network with static sampling rates. For this latter solution, each flow is monitored only one time at the input

interface of the edge router of its originating AS. This has the advantage that every sampled packet belongs to one of the flows of interest thereby flows can be easily formed at the collector. The problem is that this offers few options to sample a flow, and thus small flows that get mixed at their input interface with large flows suffer from a low sampling rate. Our approach has the nice feature of giving more choices for where to sample a flow, hence the protection of small flows. One has to add the dynamic feature of our approach and its ability to combine multiple measurements for the same flow and to limit the overhead. We use for the comparison two specific accounting applications:

- Traffic matrix estimation: All AS-to-AS flows are considered as described in section Section 4.3.2.2.2.
- AS traffic estimation: The focus is on the total volume of traffic generated by each stub AS.

For this experimentation, we use the Geant-like topology and the variable traffic trace V. The parameters of the experimentation are set as in the previous sections. For the sampling rate of the static edge configuration, denoted by p, we set it in such a way that the resulting reporting overhead is the same as in our network-wide adaptive case, and this for the main purpose of fairness between the two approaches. If N_S is the total number of 5-tuple flows in the trace, D the duration of the trace, $\pi(S)$ the probability to sample a 5-tuple flow of size S packets, S being a random variable, then the sampling rate p is given by:

$$\frac{N_S.\pi(S)}{D} = \frac{N_S.E[1-(1-p)^S]}{D} = TO.$$

The term on the right-hand side is no other than the target overhead of our adaptive architecture. The term on the left-hand side is an estimation of the rate of collected records in the static edge configuration.

Traffic matrix estimation: While giving on average close performance to our approach, the edge solution presents sampling bias against small flows as we can see in Figure 5.9 for the case of the smallest 20 flows. This figure plots the average mean relative error as a function of the TO. With the edge solution, small flows dilute within large flows and suffer from low estimation accuracy. If this happens, no other choices are available to sample them elsewhere. However, with our approach, we are able to track small flows on other lightly loaded links inside the network and combine measurements from different routers together without incurring much overhead on the system. As we can see, in order to track small flows using the edge solution with a similar accuracy to the one we obtain using the adaptive solution, we have to use a TO value larger than 150% of the value used by the adaptive solution.

AS traffic estimation: We change our objective and instead of defining a flow as being the volume of traffic from one stub AS to another stub AS, we define it as the total volume of traffic

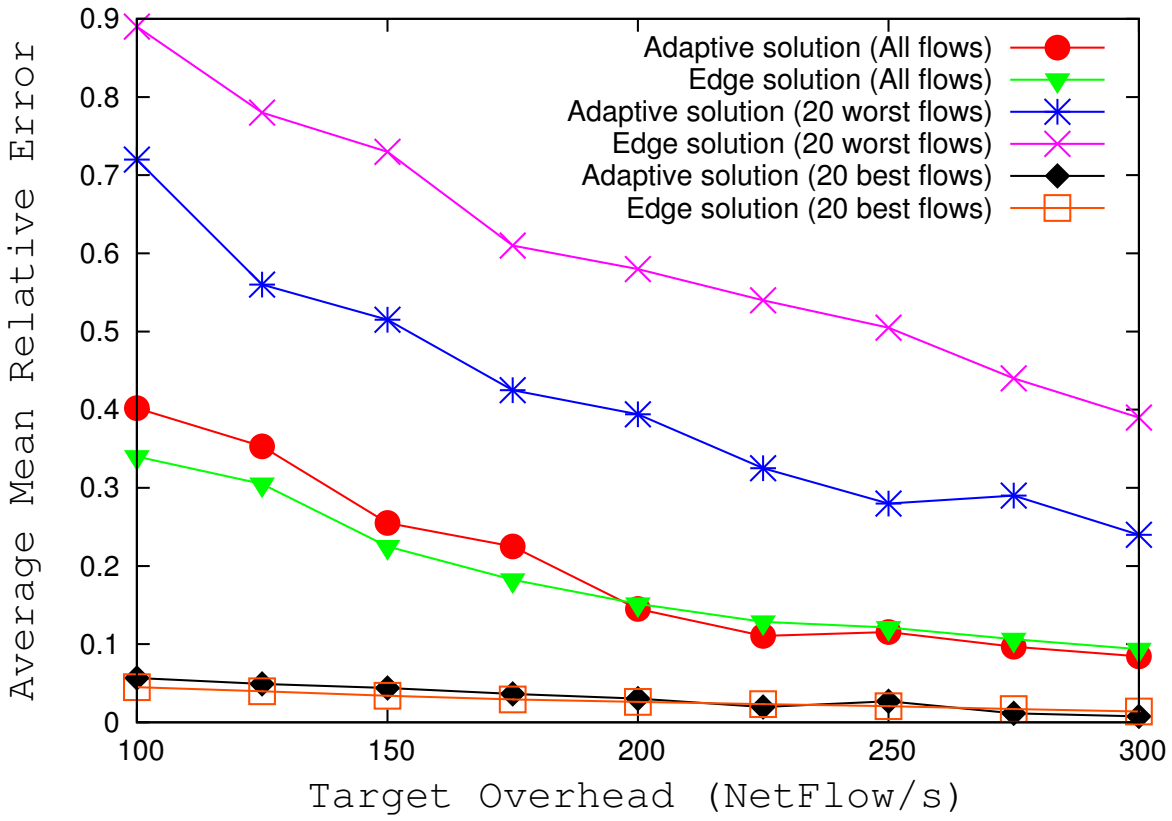


Figure 5.9: The mean relative error of flow measurements: Our approach vs. static edge one.

Table 5.2: Comparing AS traffic volume estimations.

AS	Affected weight	Without sampling		Edge sloution		All ASes		Large ASes	
		AS ratio	# of pkts	AS ratio	# of pkts	AS ratio	# of pkts	AS ratio	# of pkts
2	10	9.926	13797278	8.91	14273284	10.88	12937707	9.35	14040661
5	8	6.66	9258345	5.31	8499160	7.08	8415835	6.568	9860137
4	7	8.13	11302633	6.616	10590567	8.74	10387119	7.928	11901672
7	6	7.23	10049975	5.782	9256026	7.704	9155527	7.1	10663023
8	5	6.03	8381660	4.749	7602165	6.376	7577020	5.968	8959994
11	4	3.04	4228675	1.807	2892413	2.975	3535172	0.45	676588
9	2	1.87	2599884	0.633	1013954	1.542	1832918	0.155	233989
Total number of packets		138999178		160057553		118830397		150119112	

generated by each stub AS. We count both the outgoing and ingoing traffic for each AS. The best configuration is the one that minimizes the sum of mean square relative errors of AS traffic estimators. Changing target measurement is very easy in the context of our approach; one has to correctly define an aggregate flow, the configuration of sampling rates follows automatically. In this scenario, the traffic volume of an AS should be proportional to the weight attributed to it during the trace dispatching phase.

To further prove the generality and efficiency of our approach in comparison to the standard static edge one, we perform two tasks within this scenario. In a first time, we estimate the traffic volume of the different ASes (All ASes task). Then, we estimate the traffic volume of ASes contributing to more than some threshold of the total traffic trace (Large ASes task). We present results for a 6% threshold. Some ASes are smaller than this threshold but there traffic might still be reported to the central collector, yet it is not included in the optimization loop and is not returned to the monitoring application. The main purpose of our architecture is to reduce this volume of undesirable traffic. The All ASes task is a particular case of this second general task and can be obtained by setting the threshold to 0%.

Table 5.2 presents a summary of the experimental results for a selection of ASes. The first two columns present the AS number and its associated weight. The other columns present the AS traffic volume estimation in number of packets and the ratio of this volume with respect to the total estimated network traffic. Four configurations are presented, the one without sampling as a reference configuration, the static edge one, the adaptive All ASes one, and the adaptive Large ASes one. A set of observations can be made from these results. The first observation is that the All ASes adaptive configuration provides more accurate results for all AS traffic volumes independently of their sizes. The traffic volume of ASes is better estimated than with the static edge configuration, especially for small ASes whose traffic get diluted within the traffic of large ASes if only sampled at the edge. The second observation we can make is that for large ASes, one can even get a better estimation by only focusing in the optimization on the sizes of these large ASes. As requested, small ASes contributing to less than 6% of the total network traffic got ignored by our optimization, hence the decrease in their accuracy. Indeed, the overhead these small ASes generate with the All ASes configuration and the static edge one is used to better sample the large ASes and to better estimate their traffic volumes. These results illustrate the adaptive nature of our approach and its capacity to cope with the monitoring application needs, always at constant monitoring overhead.

5.1.2.4 Global sensitivity analysis

In the previous two parts, we gave a particular attention to the impact of the overhead target value. Yet, the system has other parameters and it is important to evaluate their impact as well. In this section we use the sensitivity analysis method described in 3.2 to demonstrate

indeed that, apart from the overhead target value, the other parameters have minor impact on system performance.

The goal of global sensitivity analysis is to characterize, qualitatively or quantitatively, what impact an input parameter has on a system output and how it compares with the impact of the other parameters. In other words, using Sensitivity Analysis, one can determine how changes in one or several parameters will impact the target output variable. In this section we apply Sensitivity Analysis to study the importance of the parameters of the proposed architecture. Fourier Amplitude Sensitivity Test [36, 37] is considered to be one of the most efficient methods in sensitivity analysis [82, 84]. Among its advantages are: fast implementation, possibility to deal with nonmonotonic models, arbitrary large variations in input parameters, and no need for the knowledge of the mathematical model.

The main idea of FAST is to assign to each parameter a distinct integer frequency (characteristic frequency). Then, for a specific parameter, the variance contribution can be singled out of the model output with the help of the Fourier transformation. Therefore, FAST is also referred to as variance based sensitivity analysis.

We have applied the method FAST to our system in order to characterize the impact of the different parameters used in experimentations on results. Table 5.4 summarizes the different evaluated parameters with their ranges. The last column presents the impact of each parameter on the system output.

For the case of six input parameters we obtain the following values of the characteristic frequencies

$$\omega_1 = \Omega_6 = 1$$

$$\omega_2 = \omega_1 + d_5 = 21$$

$$\omega_3 = \omega_2 + d_4 = 31$$

$$\omega_4 = \omega_3 + d_3 = 37$$

$$\omega_5 = \omega_4 + d_2 = 45$$

$$\omega_6 = \omega_5 + d_1 = 49$$

It is immediately noticed that the parameter having most important impact on the system output is the target overhead TO while the other parameters have a light impact on results in the order of 1% or less. Indeed, for some value of TO, there is an optimal configuration of monitors, and our system will converge to this optimal configuration in a robust manner with respect to the other parameters. It is only by changing the value of TO that the system will converge to another optimal configuration yielding another measurement precision.

Table 5.3: Parameters of the experiment.

Parameter	symbol	range	impact
Target Overhead	TO	$[20, 500]$	0.58
Computation period	d	$[60s, 300s]$	0.0142
γ regulator	σ	$[1, 10]$	0.00747
Initial sampling rate value	P_{init}	$[0.005, 0.02]$	0.01179
Minimum sampling rate value	SR_{min}	$[0.0005, 0.005]$	0.00691
Maximum sampling rate value	SR_{max}	$[0.02, 1]$	0.00721

5.1.3 Summary

In this section, we presented a responsive optimization method that automatically reconfigures the sampling rates in the different monitors according to the monitoring application requirements and resource consumption constraints. By using this optimization method, the network operator just has to select a measurement task and a monitoring resource constraint ($\mathcal{TO}$). Our self-configuring system will then iterate measurements and adjust sampling rates in small steps in order to address the tradeoff between monitoring accuracy and overhead.

Experimental results proved the ability of our system to continuously improve the monitoring accuracy while limiting the overhead to its target value. Moreover, the system provides a fair allocation of sampling rates over monitors so that measurement errors are homogeneously distributed among flows independently of their volumes. Compared to static edge configuration, our network-wide adaptive system has shown its advantages in better capturing network flows especially for small flows.

5.2 Proactive Network Reconfiguration Method

We have presented in the previous section a reactive optimization method for network monitors reconfiguration. While this solution provides a responsive control of sampling rates and quickly reacts to network condition changes and measurement accuracy requirements, it disrupts the system with unnecessary details specific to a particular observation period since it tracks fine-grained changes in the traffic.

In this section, we introduce the notion of smoothness for network configuration and define it as the degree or the aggressiveness to which the system should react to changes of network conditions. By tracking smoothly changes in network conditions and measurement accuracy we

exhibit unnecessary oscillations and avoid the introduction of unnecessary details and information in network traffic in order to keep resource consumption within a desired bound. Our solution relies on an overhead prediction module in order to track long-term and short-term variations in the traffic using an exponentially weighted moving average. The different weights of this method determine the responsiveness to changes and address the tradeoff between responsiveness and smoothness.

In this section, we will present the optimization procedure using a single sampling primitive: packet sampling and two monitoring tasks: heavy hitter detection and flow size estimation.

5.2.1 Challenges and objectives

Recently, many proposals try to design a network-wide monitoring infrastructures that coordinate monitoring responsibilities between different monitors e.g. [86, 28]. However, despite these available solutions, monitoring applications still present some shortcomings including the problem of overhead prediction and that of improving accuracy of multiple tasks.

The optimization of monitoring applications requires the estimation of overhead in order to find the appropriate configuration that keeps the overhead within a target value while providing the best possible accuracy. The majority of existing solutions uses information about links' load and defines the overhead as being the total number of packets that can be sampled in the entire network. Clearly, such approach leads to an inefficient use of resources since the load of links varies over time. These variations can either degrade the accuracy of measurements or increase the resource consumption.

We argue that an advanced module for overhead prediction can significantly increase the monitoring capabilities of a network and cope with short-term and long-term variations in the traffic. In this section, we introduce an adaptive monitoring system that adjusts its configuration according to network conditions and measurement accuracy. Our system relies on an optimization method consisting of: *(i)* an overhead prediction based on an Exponential Weighted Moving Average filter to track long-term and short-term variations in the traffic, *(ii)* a global weighted utility function to deal with multiple monitoring tasks at the same time, and *(iii)* an optimization algorithm that configures monitors to address the tradeoff between resource consumption and accuracy of different tasks.

5.2.2 Optimization method description

Given a list of measurement tasks $\mathcal{T}$ and an overhead constraint (Target Overhead $\mathcal{TO}$), our system adaptively adjusts its configuration to answer the requirements of multiple tasks while tracking 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 a prediction of the overhead and in a way to optimize the accuracy of the considered measurement tasks.

In order to address the tradeoff between responsiveness and smoothness, the proactive approach relies on advanced method for overhead prediction. This method relies on the exponentially weighted moving average in order to track variations in the traffic. The different weights of this method determine the responsiveness and the aggressiveness of our system to changes in network conditions. In this section, we present the architectural ideas behind our system.

5.2.2.1 Overhead prediction

The optimization procedure requires the estimation of overhead in order to find the optimal configuration that keeps the overhead within a target value and shares resources between the different monitoring primitives while providing the best possible accuracy. Hence, in order to efficiently use resources we predict the value of the resulting overhead.

$$\mathcal{O} = \sum_{m \in \mathcal{M}} \sum_{s \in \mathcal{S}} (\mathcal{O}_m^s), \tag{5.1}$$

where $\mathcal{O}_m^s$ is the number of NetFlow records generated by the sampling primitive $s \in \mathcal{S}$ in the monitor $m \in \mathcal{M}$. In order to track variations in the traffic, we use the Exponentially Weighted Moving Average (EWMA) which is a memoryless moving average whose weights are exponentially decreasing from more recent historical samples to older ones. The weight for each older data point decreases exponentially with time, giving much more importance to recent observations while still not discarding older observations entirely. Next, we will explain the prediction procedure using a single sampling primitive: packet sampling and two monitoring tasks: heavy hitters detection and flow size estimation.

Consider the set of paths $\mathcal{A}$ in a network. Each path $a \in \mathcal{A}$ consists of a set of monitors. Let $\overline{N}_a$ and $\overline{\Sigma}_a$ be respectively the smoothed version of the number of flows along the path a and their mean size in terms of packets. Hence, we update as follows:

$$\overline{N}_a \leftarrow \delta N_a + (1-\delta)\overline{N}_a, \tag{5.2}$$

$$\overline{\Sigma}_a \leftarrow \delta \Sigma_a + (1-\delta)\overline{\Sigma}_a, \tag{5.3}$$

where N_a and Σ_a are the last observations which can be approximated by $\hat{N}_a$ and $\hat{\Sigma}_a$. $\delta = \frac{2}{(n+1)}$ 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 of tracking variations in the traffic. For instance, if we want to track changes on hourly scale (i.e. $\tau = 3600s$), we calculate the window length $n = \frac{\tau}{d}$, where d is the period of configuration updates.

Next, we will give the expression of $\hat{N}_a$ and $\hat{\Sigma}_a$ using the single packet sampling primitive data.

Consider n_a^p to be the number of packet-sampled flows crossing the path a. Let S be the size of a given flow in terms of packets, then the probability that this flow is packet-sampled along the 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 $S.\pi_a^p$, where $\pi_a^p = \sum_{k \in a} p_k^p$ for small p_k^p. The 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 the path a. Hence, we can give an estimation for N_a:

$$\hat{N}_a^p = \frac{n_a^p}{\pi_a^p.\Sigma_a}. \tag{5.4}$$

This estimator has a variance equal to

$$Var(\hat{N}_a^p) = \frac{Var(n_a^p)}{(\pi_a^p)^2.(\Sigma_a)^2} = \frac{(1 - \pi_a^p.\Sigma_a).N_a}{\pi_a^p.\Sigma_a}. \tag{5.5}$$

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

Let s_a be the total number of sampled packets along the path a. The estimator of the total number of packets crossing the path a, that maximizes the likelihood is known to be: $\hat{S}_a = \frac{s_a}{\pi_a^p}$. Hence, we can derive a first estimation of the mean flow size crossing the path a.

$$\hat{\Sigma}_a = \frac{\hat{S}_a}{\hat{N}_a} \tag{5.6}$$

Equations (5.4) and (5.6) constitute a system of two equations with two unknowns that we can solve to find the values of $\hat{N}_a$ and $\hat{\Sigma}_a$.

Using equations (5.2), (5.3) and (5.1), we can now give the analytical expression of the overhead prediction:

$$\mathcal{O} = \sum_{m \in \mathcal{M}} \sum_{a \in \Gamma_M} \pi_a^p.\overline{N}_a.\overline{\Sigma}_a, \tag{5.7}$$

where $\Gamma_M \subset \mathcal{A}$ is the subset of paths containing the monitor M. This overhead prediction is no other than the smoothed version of the number of sampled flows.

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. For this, we use the Algorithm 2 implementing the EWMA filter for prediction according to Equation (6.18). 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. We have to find the suitable time scale that addresses the tradeoff between these two extremes.

Data: *Measured flows and packets* in the different paths, (n_a^p) and (s_a).
The previous estimations: $\overline{N}_a$ and $\overline{\Sigma}_a$, and the previous sampling rate vector (p_k^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 $\hat{N}_a = \frac{n_a^p}{\pi_a^p.\overline{\Sigma}_a}$;
    \\Estimate the number of 5-tuple flows.
    $\overline{N}_a \leftarrow \alpha\hat{N}_a + (1-\alpha)\overline{N}_a$;
    calculate $\hat{S}_a = \frac{s_a}{\pi_a^p}$; calculate $\hat{\Sigma}_a = \frac{\hat{S}_a}{\hat{N}_a}$;
    \\Estimate the mean size of 5-tuple flows.
    $\overline{\Sigma}_a \leftarrow \alpha\hat{\Sigma}_a + (1-\alpha)\overline{\Sigma}_a$;
    \\Predict the expression of the overhead
    $O_a \leftarrow \overline{N}_a.\overline{\Sigma}_a.\pi_a^p$;
  end
  \\Derive the global overhead prediction expression.
  $\mathcal{O} = \sum_{M\in\mathcal{M}} \sum_{a\in\Gamma_M} \pi_a^p.\overline{N}_a.\overline{\Sigma}_a$;
  return $\{\mathcal{O}\}$
end

Algorithm 2: Overhead prediction method.

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

$$U = \sum_i \gamma_i Var(\hat{T}_i), \tag{5.8}$$

under the following constraints:

$$\mathcal{O} \leq \mathcal{TO} \tag{5.9}$$

$$p_k \leq SR_{max} \qquad \forall k \in \mathcal{M}, \tag{5.10}$$

$$p_k \geq SR_{min} \qquad \forall k \in \mathcal{M}, \tag{5.11}$$

SR_{min} and SR_{max} are respectively the minimum and maximum sampling rate values we allow in monitors.

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 (6.20), (6.21) and (5.11). We solve this Lagrangian by an iterative procedure using the Newton method (refer to [26], Chapter 9.5). The idea of the method, summarized in Algorithm 3, 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 find the best step value σ_b. We continue until we either reach the global minimum or we exceed the maximum number of iterations.

5.2.3 Validation results

In this paragraph, we study the efficiency of our adaptive solution. We then provide a global sensitivity analysis to study the importance of the different parameters.

We chose to study the performance of our system by emulating Geant 5.1, the European Research network [4]. As target, we consider two accounting applications or tasks: *(i)* traffic matrix estimation (T_1) which consists in jointly estimating edge-to-edge flow sizes. A flow F_i is the set of 5-tuple flows that share the same AS source and AS destination; *(ii)* large AS traffic estimation (T_2) where we aim to estimate the volume of the greediest ASes (those contributing to more than some percentage of the total traffic, we take 7% as example). Except when explicitly mentioned, we assign to these tasks respectively the equal weights $\gamma_1 = \gamma_2 = 0.5$ (see Equation 5.8).

Data: The expression $L(P^{t+1})$ and the previous sampling rate vector $P^t = (p_k^t)$
Result: The new sampling rate vector P^{t+1}
begin
  $i \leftarrow 0$; $\overrightarrow{P}_i \leftarrow P^t$;
  while $i < max_{iteration}$ *and* $min_{found} == false$ **do**
    Evaluate the gradient vector $\nabla L(\overrightarrow{P}_i)$;
    Evaluate the Hessian matrix $H(\overrightarrow{P}_i)$;
    \\Compute the search direction
    $\overrightarrow{S}_i \leftarrow -\nabla L(\overrightarrow{P}_i)(H(\overrightarrow{P}_i))^{-1}$;
    \\Calculate the best step σ_b value using line \\search algorithm
    $\sigma_b \leftarrow lineSearch(\overrightarrow{P}_i + \sigma \overrightarrow{S}_i)$;
    \\calculate the next point
    $\overrightarrow{P}_{i+1} \leftarrow \overrightarrow{P}_i + \sigma_b \overrightarrow{S}_i$;
    \\Perform the termination test for minimization
    if $L(\overrightarrow{P}_{i+1}) - L(\overrightarrow{P}_i) < precision$ **then**
      $min_{found} \leftarrow true$;
    end
    $i \leftarrow i + 1$;
  end
  return $\{\overrightarrow{P}_i\}$
end

Algorithm 3: Optimization procedure.

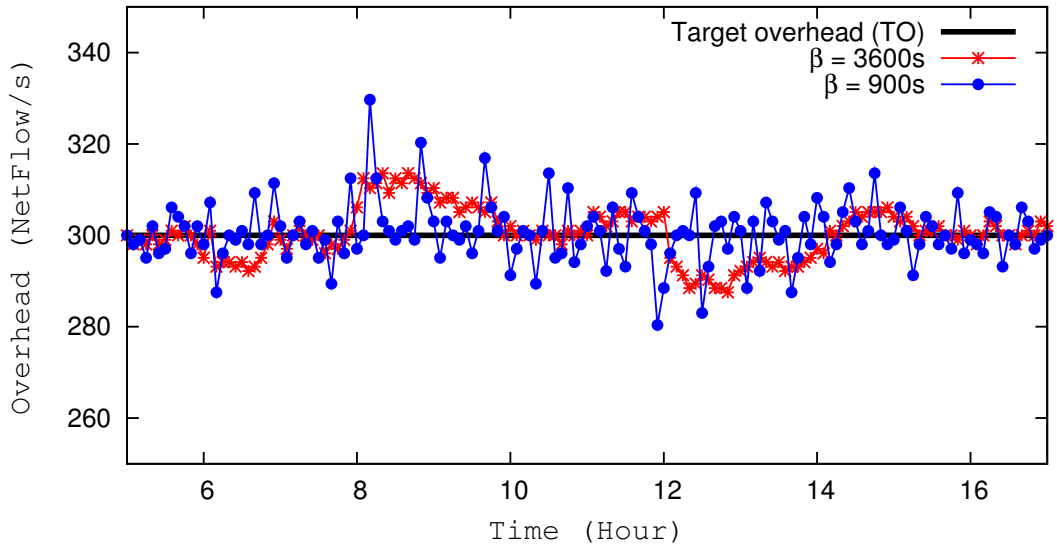


Figure 5.10: Resulting overhead vs. time using two time scales to track variations in the traffic.

5.2.3.1 System efficiency

In this part, we aim to address the performance of our system. For this experiment, we set the update period d to 5 minutes, the time scale τ to 3600s, the minimum sampling rate SR_{min} to 0.0005 and the maximum one SR_{max} to 1. The $\mathcal{TO}$ is set to 300 NetFlow-records/s.

In order to evaluate the performance of the overhead prediction method, we plot in Figure 5.10 the evolution of the measured overhead (exported NetFlow records) over time. We observe that the system profits from the available resources to provide the best possible accuracy. For the two considered time scale values, the system maintains the overhead around the $\mathcal{TO}$. The use of a small time scale ($\tau = 900$s) leads to an oscillating behavior of the overhead since the system tracks more details and fine-grained changes in the traffic. On the contrary and because of a coarser aggregation of NetFlow reports, tracking changes on hourly scale leads to a more stable behavior of the overhead.

Figure 5.11 shows the value of the average mean relative error for different $\mathcal{TO}$ values. We notice the impact of the $\mathcal{TO}$ on the global estimation accuracy. There is a clear reduction of the overall measurement error when the target overhead increases. The error drops from 0.284 for a $\mathcal{TO}$ equal to 100 NetFlow-records/s, to 0.0279 for a $\mathcal{TO}$ equal to 400 NetFlow-records/s. Indeed, the system tries to provide the best possible accuracy given a monitoring constraint ($\mathcal{TO}$). Allowing more overhead (resources) gives the system the possibility to increase some sampling rates and to collect and export more data looking for better estimation. The main strength of our system is that it is able to cope with any $\mathcal{TO}$ value and provides for this value the best configuration of monitors.

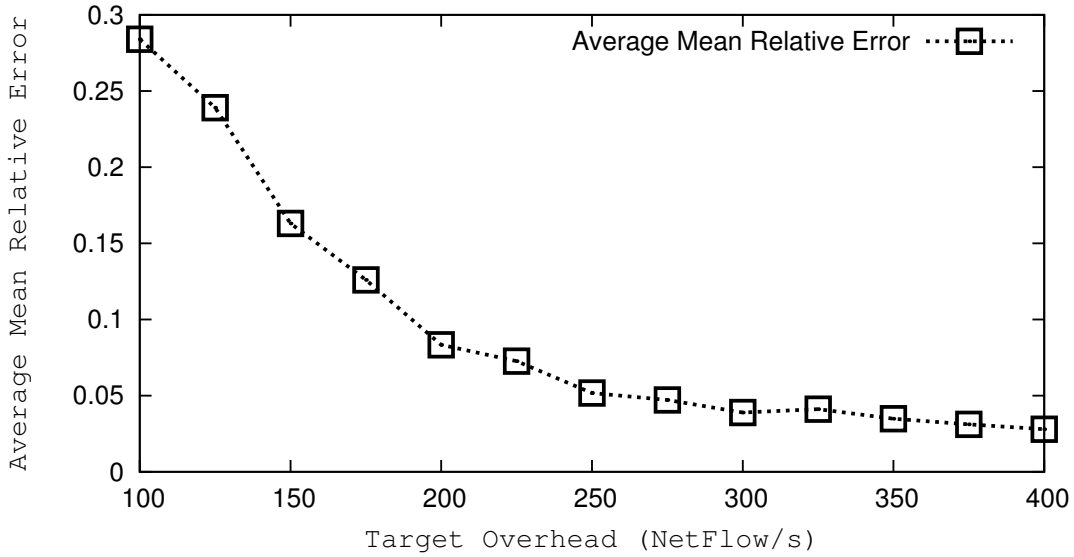


Figure 5.11: Average mean relative error vs. ($\mathcal{TO}$).

Now, we want to study the performance of this global optimization as a function of the weights assigned to each task T_i. We run experimentations using two tasks T_1 and T_2 while changing their assigned weights (γ_1 and $\gamma_2 = 1-\gamma_1$). We plot in Figure 5.12 the average mean relative error of T_1 as a function of its assigned weight γ_1. As expected, γ_1 has a clear impact on the accuracy of T_1. The mean relative error varies between 0.0296 and 0.18 for different values of γ_1. For instance, setting γ_1 at a large value (i.e. setting γ_2 at a low value) gives more importance to T_1 in the optimization procedure and decreases the impact of the estimation error of T_2 on the global accuracy. In this manner, the optimal solution that maximizes the global accuracy is the one that satisfies especially the accuracy of T_1. This result confirms the flexibility of our method where the operator can set the weights according to the importance of tasks. By setting the weights to different values, one can achieve high accurate measurements for important tasks at the expense of less important ones.

5.2.3.2 Global sensitivity analysis

In the previous part, we have studied the performance of our system and the influence of some parameters on results. In this part, we use global sensitivity analysis to characterize, qualitatively and quantitatively, what impact an input parameter has on the system output and how it compares with the impact of the other parameters.

We have applied the method FAST 3.2 to our system to characterize the impact of the different parameters used in experimentations on results. Table 5.4 summarizes the different evaluated parameters with their ranges. The last column presents the impact of each parameter

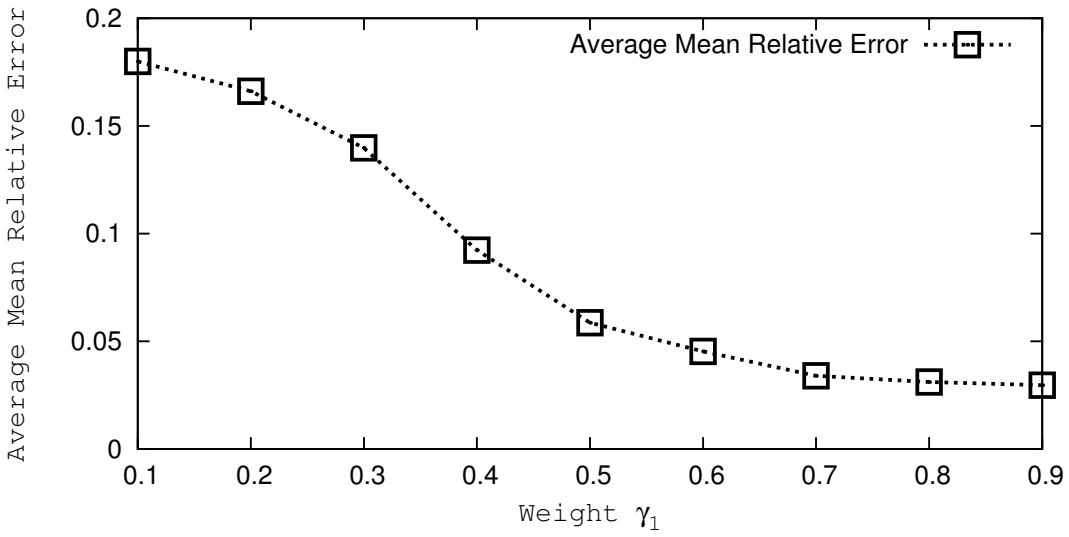


Figure 5.12: Average mean relative error of T_1 vs. the weight γ_1.

Table 5.4: Parameters of the experiment.

Parameter	symbol	range	impact
Target Overhead	$\mathcal{TO}$	$[20, 500]$	0.431
Time scale	τ	$[60s, 7200s]$	0.1147
Computation period	d	$[60s, 300s]$	0.0234
Min sampling rate	SR_{min}	$[0, 0.01]$	0.0876
Max sampling rate	SR_{max}	$[0.01, 1]$	0.0935

on the system output in terms of measurement accuracy.

For the case of five input parameters we obtain the following values of the characteristic frequencies

$$\omega_1 = \Omega_5 = 11;$$

$$\omega_2 = \omega_1 + d_4 = 21;$$

$$\omega_3 = \omega_2 + d_3 = 27;$$

$$\omega_4 = \omega_3 + d_2 = 35;$$

$$\omega_5 = \omega_4 + d_1 = 39;$$

It is immediately noticed that the parameter having most important impact on the system output is the target overhead $\mathcal{TO}$. In our system, the $\mathcal{TO}$ is a monitoring constraint set by the operator and can be changed to achieve a given accuracy. We also observe the important impact of the time scale τ parameter. Thus, it is so important to set this parameter at a suitable value in order to address the tradeoff between the long-term and short-term variations and to improve results. The other parameters have a light impact on the behavior of the system (less than 10%).

5.2.4 Summary

We have presented a Proactive network reconfiguration method that coordinates responsibilities between the different monitors in order to achieve the best possible accuracy while respecting monitoring constraints. Our optimization method based on overhead prediction to track short-term and long-term variation and global weighted utility function to deal with multiple tasks.

Experimental Results proved the ability of the Proactive method to keep the resulting overhead around a target value. We also demonstrated that our system is practical: it provides an efficient method to achieve multiple monitoring objectives using a weighted utility function and it relies on a flexible method to track variations in the traffic according to an adaptable time scale. Moreover, we provided a global study of the impact of the different parameters on the behavior of the system.

5.3 Reactive optimization method vs. Proactive optimization method

In this section we aim to compare the performance of the two different network reconfiguration methods (i.e., Reactive and Proactive optimization methods) described previously.

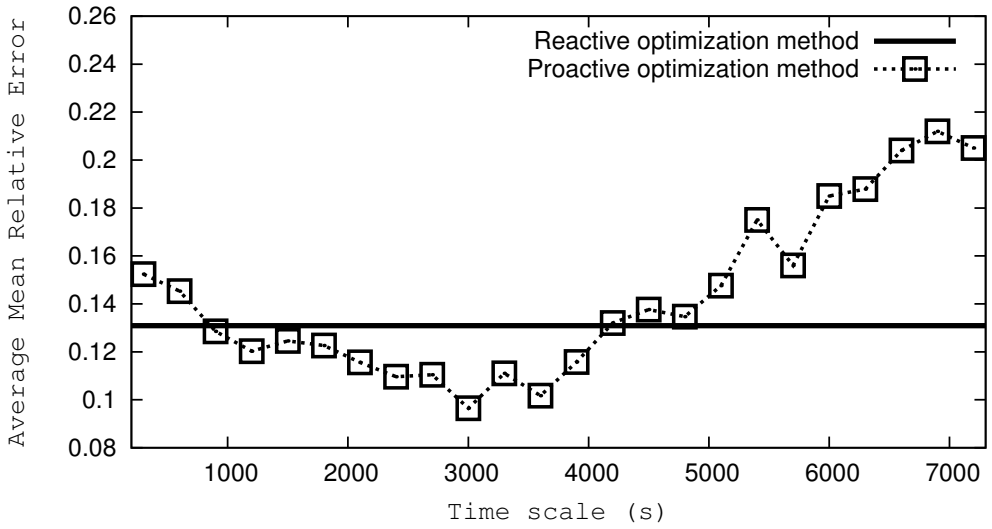


Figure 5.13: Reactive optimization method vs. Proactive optimization method for different values of time scale τ.

We study the performance of these optimization methods in terms of measurement accuracy, resulting overhead and responsiveness to changes in the network traffic.

We validated the performance of our system over MonLab [8, 68]. We chose to study the performance of our system by emulating Geant. As target, we consider the two accounting applications or tasks described above: The traffic matrix estimation (T_1) and large AS traffic estimation (T_2). We assign to these tasks respectively the equal weights $\gamma_1 = \gamma_2 = 0.5$. For this experiment, we set the update period d to 5 minutes, the time scale τ to 3600s, the minimum sampling rate SR_{min} to 0.0005 and the maximum one SR_{max} to 1. The $\mathcal{TO}$ is set to 300 NetFlow-records/s. We divide the performance evaluation into two parts. First, we study the accuracy of measurements provided by each method for different values of input parameters. We evaluate their performance for different values of the monitoring constraint values. Then, we compare the efficiency and convergence of these two methods. We study their resulting overhead and their ability to detect changes in network traffic and to react to these changes.

5.3.1 Measurement accuracy study

We plot in Figure 5.13 the evolution of the mean relative error of the two methods. For the Proactive optimization method, we have used different values of the time scale of tracking changes in network traffic. We observe for small values of the time scale, the Proactive optimization method is close to the behavior of the reactive method. In fact, for small time scale the Proactive method tracks more details in the traffic and becomes sensitive to the different changes in network conditions even those corresponding to a specific period of time. However,

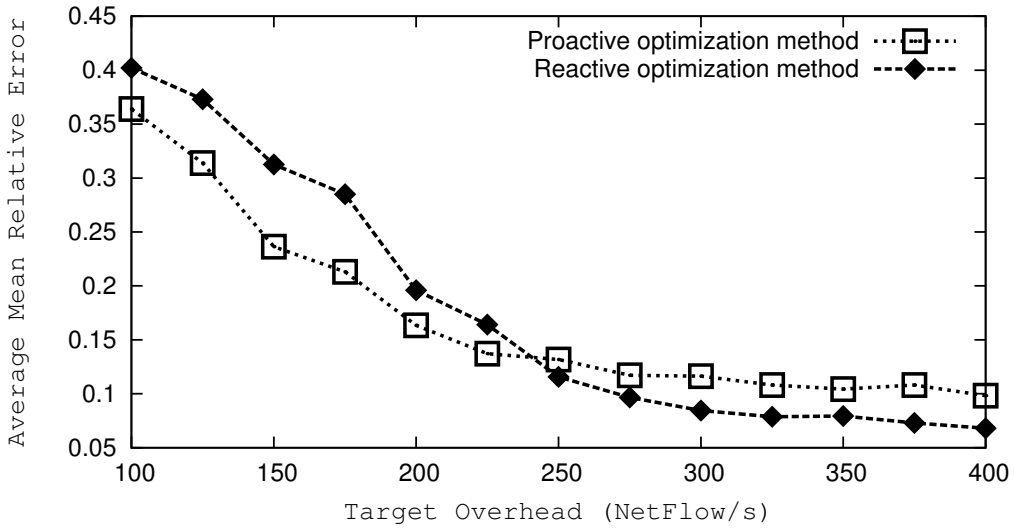


Figure 5.14: Comparing the performance of the Reactive and Proactive optimization methods for different values of $\mathcal{TO}$.

for large time scale values the Proactive method provides inaccurate measurements compared to those provided by the Reactive method. Nonetheless, using the suitable value of the time scale parameter, the Proactive method performs better results than the Reactive method.

Figure 5.14 shows the value of the mean relative error for different target overhead values. We observe that the performances of the two optimization methods look the same. However, for small target overhead values, the Proactive method provides better performance and copes with any monitoring constraint. However, the Reactive method needs more resources in order to converge and achieve the optimal configuration of monitors. Moreover, for big values of target overhead it provides better performance than the proactive method.

5.3.2 System efficiency

In order to better study the performance of the optimization methods, we plot in Figure 5.15 the evolution of the resulting overhead over time. For this experiment, we set the timer d for updating sampling rates to 5 minutes and the $\mathcal{TO}$ is set to 200 NetFlow-records/s. For the Proactive method we use two different time scale values. We notice the ability of the reactive method to detect changes in the traffic. The system continues collecting NetFlow records while controlling the resulting overhead. Once it detects a change in the rate of traffic, it triggers a new optimization in order to calculate a new sampling rate vector to adapt to this change. However, this reactivity characteristics requires collecting and detecting all the changes in the traffic even unnecessary ones corresponding to a specific period. This leads to an oscillating behavior and can decrease the accuracy of results. The Proactive method adds smoothness

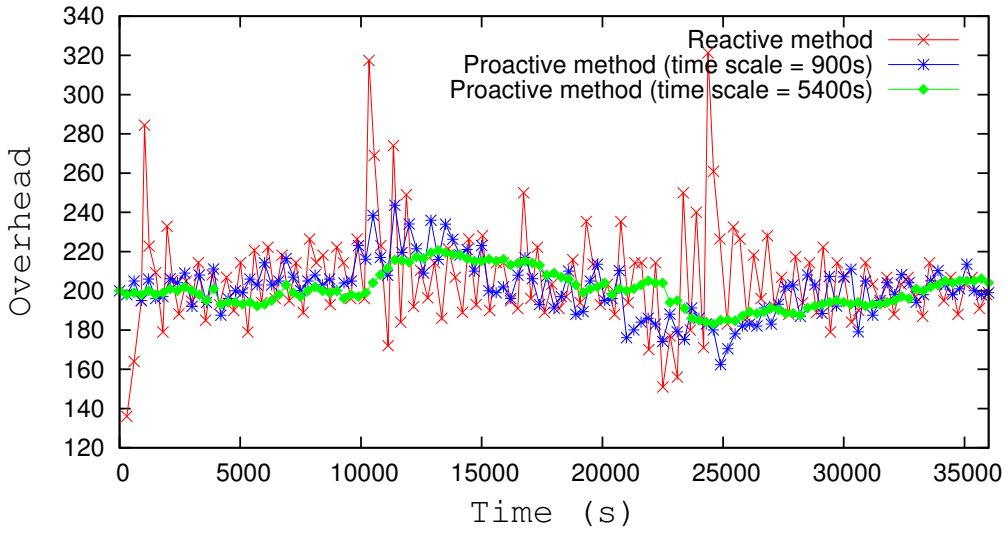


Figure 5.15: Resulting Overhead (Netflow-records/s) of the Reactive optimization method and the Proactive optimization method using two different time scales.

to the system behavior. In fact, by tracking smoothly changes in network traffic conditions, the system can avoid unnecessary details in the traffic. This can lead to a smooth behavior and allows respecting the monitoring constraint. However, by using a large time scale we lose responsiveness and ability to detect changes in network traffic. We can observe that for a time scale equal to 5400s, the system cannot detect changes because of coarse aggregation of NetFlow reports. For a small time scale (i.e., 900s), we observe an oscillating behavior close to the behavior of the reactive method.

5.4 Conclusions

In this chapter, we have provided two different network reconfiguration methods: *(i)* A Reactive optimization method inspired from dynamics used by TCP for the adjustment of its congestion window. This method provides a reactive solution able to track changes in network traffic conditions. It is suited for monitoring applications that depend on the communication structure e.g., security applications and anomaly detection applications that analyze communication structure. These monitoring applications need a continuous analysis of the network traffic. In order to detect and react to traffic changes, the optimization should track all the details. This leads to an oscillating behavior and can degrade the accuracy of measurements. *(i)* A Proactive optimization method based on overhead prediction. This method provides more smoothness and reacts less aggressively to changes in network traffic changes. This solution is able to monitoring applications that require an understanding of volume structure; e.g., heavy-

hitter detection and traffic engineering that require an understanding of the number of packets/bytes per-port or per-src. These monitoring applications don't require tracking all changes and details in the network traffic. We need just to fix a time scale to track these changes. Once done, the optimization method reconfigures monitors accordingly.

In this chapter, we have presented two complementary network configuration methods that use a single sampling primitive (i.e., Packet Sampling) to achieve some monitoring tasks. We argue that it is possible to investigate the different sampling primitives in order to support a large spectrum of applications. In the next chapter, we introduce an adaptive system that combines two sampling primitives, packet sampling and flow sampling, and that is able to satisfy multiple monitoring tasks.

6

SAMPLING PRIMITIVES COMBINATION

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 realize some performance objective. It becomes 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 the previous Chapter (Chapter 5) we have validate our system presented in Chapter 4 through experimental scenarios and using the packet sampling primitive. In this Chapter, and to prove the interest of the joint approach, we extend the presentation of our system using two sampling primitives, packet sampling and flow sampling. Our system consists of two main functions: *(i)* a global estimator that investigates measurements of 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. In this Chapter we will use the proactive optimization method described in Chapter 5.

Contents

6.1 Introduction

The importance of traffic measurements and passive monitoring 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 large covering among others flow sampling [60], sample and hold [51] and packet sampling [28]. However, network management applications require accurate estimates of a wide range of flow-level traffic metrics. Given the inadequacy of current solutions, several application-specific monitoring algorithms have emerged. While these provide better accuracy for the specific applications they target, they increase router complexity and require vendors to commit to hardware primitives without knowing how useful they will be to meet the needs of future applications. These application-specific systems still present some drawbacks including the problem of tightly coupling target applications and sampling primitives (i.e. they focus on achieving a

specific application using a single sampling primitive). For instance, the authors in [28] use the packet sampling primitive and reconfigure periodically the different sampling rates in order to calculate the traffic matrix, while the authors in [86] use the flow sampling primitive for flow counting. The main consequence of this trend is the deployment and the commitment of monitoring systems using single sampling primitives for the achievement of specific monitoring tasks without thinking of a combination of these primitives for a broader usage. Hence, it becomes complicated to achieve a general class of monitoring tasks using such application-specific systems. In order to solve these limitations, some proposals have presented simple combination of existing sampling primitives in order to achieve larger class of tasks. For instance, the authors in [94] combine a small number of simple and generic router primitives that collect flow-level data to estimate traffic metrics, while the authors in [66] use a combination of flow sampling and sample-and-hold to provide traffic summaries and detect resource hogs. The system proposed in this thesis should be able to combine different sampling primitives. More importantly it should 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.

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

Three main challenges arise in the development of such a system:

- How to deal with multiple monitoring objectives and how to combine independent measurements collected 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 Chapter, we extend the presentation of our proposed system. We will show the ability of our system 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 primitives: *(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 method 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 method is based on an overhead prediction method to track sustainable changes while removing unnecessary variations in the traffic and a global weighted utility function to deal with multiple monitoring tasks.

6.2 System architecture

As described in Chapter 6.2, our system extends local existing monitoring tools (MEs) with a network-wide cognitive engine (CE) in order to drive its own deployment by automatically and periodically reconfiguring the different monitors in a way that improves the overall accuracy (according to monitoring application requirements) and reduces the resulting overhead (respecting some resource consumption constraints).

Figure 6.1 depicts a descriptive presentation of the proposed monitoring system presented before in the Figure 4.1. We present in this Figure the basic functional components of our system and the interactions among them. Moreover, we give a detailed description of the monitoring engine deploying two different sampling techniques (i.e. packet sampling and flow sampling). We chose to use two complementary sampling primitives: *(i)* Flow Sampling (FS) which is well suited for security and anomaly detection applications that analyze flow communication structure, and *(ii)* Packet Sampling (PS) which is well suited for traffic engineering and accounting applications that analyze 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-src [94]. While packet sampling consists in capturing a subset of packets independently of each other, flow sampling consists in capturing flows independently of each others. 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.

6.2.1 Global Estimator Engine

In order to explain the operation of our system and how it configures the different monitoring primitives and 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.

6.2.1.1 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

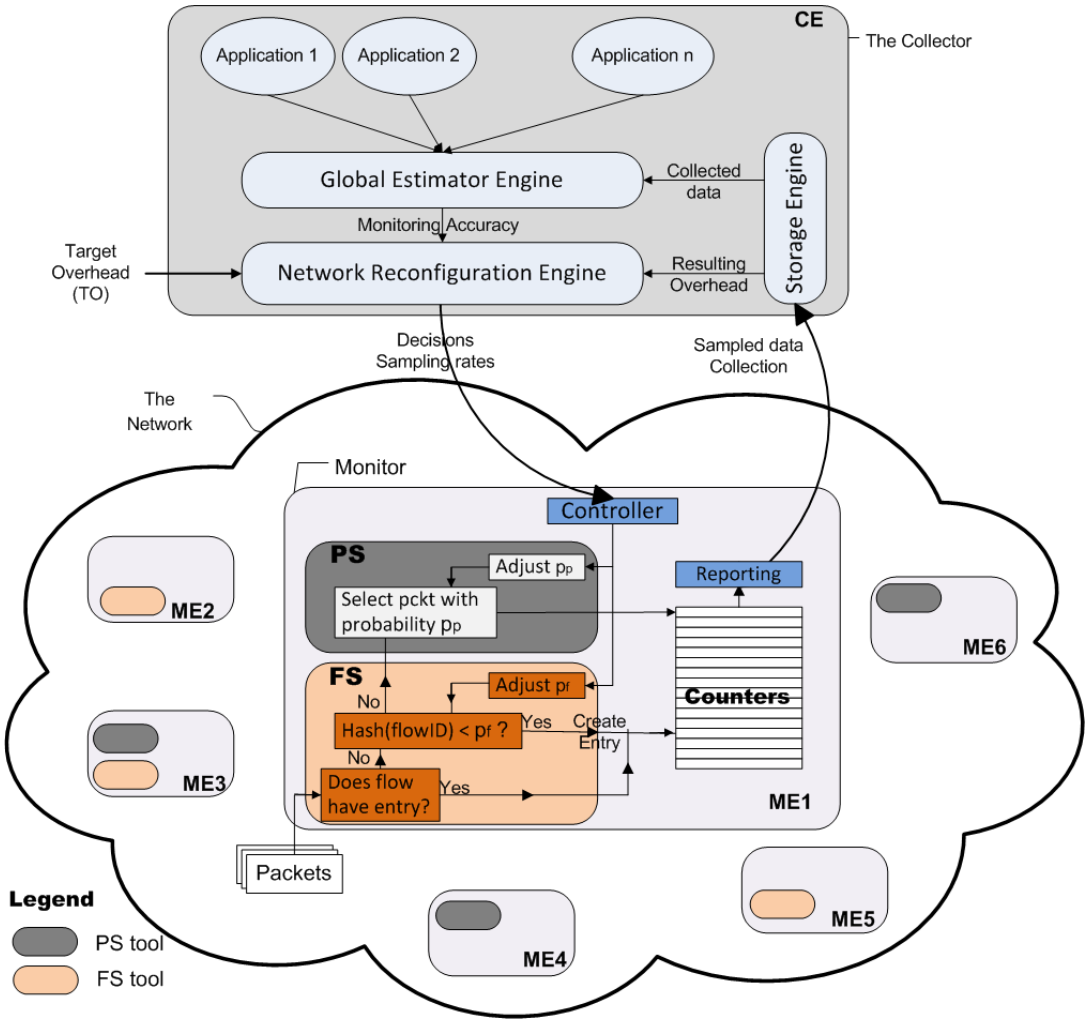


Figure 6.1: System architecture.

entire network during the time period d.

Consider the set of 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 (4.1), we can derive a better estimation of N_a as a linear combination of these two estimators:

$$\hat{N}_a = \chi_p \hat{N}_a^p + \chi_f \hat{N}_a^f, \tag{6.1}$$

where:

$$\chi_p = \frac{\frac{1}{Var(\hat{N}_a^p)}}{\frac{1}{Var(\hat{N}_a^p)} + \frac{1}{Var(\hat{N}_a^f)}} \quad \text{and} \quad \chi_f = \frac{\frac{1}{Var(\hat{N}_a^f)}}{\frac{1}{Var(\hat{N}_a^f)} + \frac{1}{Var(\hat{N}_a^p)}}. \tag{6.2}$$

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 $P^p = (p_k^p)_{k \in \mathcal{M}}$ and the flow sampling rate vector $P^f = (p_k^f)_{k \in \mathcal{M}}$. $\mathcal{M}$ is the set of monitors in the network, each monitor k is tuned by 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 the 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 estimator for the number of flows along the path a that maximizes the likelihood:

$$\hat{N}_a^f = \frac{n_a^f}{\pi_a^f}. \tag{6.3}$$

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 has a variance equal to:

$$Var(\hat{N}_a^f) = \frac{N_a^{(t)}.(1 - \pi_a^f)}{\pi_a^f}. \tag{6.4}$$

Now we will give the expression of the estimator for N_a using the PS primitive (denoted above $\hat{N}_a^p$). Consider n_a^p to be the number of packet-sampled flows crossing the path a. Let S be the size of a given flow, then the probability that this flow is packet-sampled along the 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 $S.\pi_a^p$, where $\pi_a^p = \sum_{k \in a} p_k^p$ for small p_k^p. The 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 the path a. Hence, we can give a second estimation for N_a:

$$\hat{N}_a^p = \frac{n_a^p}{\pi_a^p.\Sigma_a}. \tag{6.5}$$

This estimator has a variance equal to

$$Var(\hat{N}_a^p) = \frac{Var(n_a^p)}{(\pi_a^p)^2.(\Sigma_a)^2} = \frac{(1-\pi_a^p.\Sigma_a).N_a}{\pi_a^p.\Sigma_a}. \tag{6.6}$$

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:

$$\hat{\Sigma}_a = \frac{O_a - \pi_a^f.\hat{N}_a}{(1-\pi_a^f)\pi_a^p.\hat{N}_a}. \tag{6.7}$$

Eq. (6.1) and (6.7) constitute a system of two equations with two unknowns that we can solve to find the values of $\hat{N}_a$ and $\hat{\Sigma}_a$.

6.2.1.2 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 (4.1), 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, $\hat{F}_i^f$:

$$\hat{F}_i = \chi_p \hat{F}_i^p + \chi_f \hat{F}_i^f. \tag{6.8}$$

Where:

$$\chi_p = \frac{\frac{1}{Var(\hat{F}_i^p)}}{\frac{1}{Var(\hat{F}_i^p)} + \frac{1}{Var(\hat{F}_i^f)}} \text{ and } \chi_f = \frac{\frac{1}{Var(\hat{F}_i^f)}}{\frac{1}{Var(\hat{F}_i^f)} + \frac{1}{Var(\hat{F}_i^p)}} \tag{6.9}$$

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:

$$\hat{S}_{ji}^p = \sum_{k\in a} \lambda_k \hat{S}_{kji}^p, \quad \text{with} \quad \lambda_k = \frac{\frac{1}{Var(\hat{S}_{kji}^p)}}{\sum_{l\in a} \frac{1}{Var(\hat{S}_{lji}^p)}}, \tag{6.10}$$

where $\hat{S}^p_{kji} = \frac{s^p_{kji}}{p^p_k}$ 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 $Var(\hat{S}^p_{kji}) = S^p_{ji}.(1 - p^p_k)/p^p_k$. It follows that the estimator $\hat{S}^p_{ji}$ is equal to:

$$\hat{S}^p_{ji} = \frac{1}{\varphi_a}.\sum_{k\in a}\frac{s_{kji}}{1-p_k}, \quad \text{where} \quad \varphi_a = \sum_{l\in a}\frac{p^p_l}{1-p^p_l}. \tag{6.11}$$

The variance of this estimator is simply equal to:

$$Var(\hat{S}^p_{ji}) = S_{ji}/\varphi_a. \tag{6.12}$$

Hence, under independent sampling of packets in the different monitors, we can derive the expression of $\hat{F}^p_i = \sum_j \hat{S}^p_{ji}$ and its corresponding variance $Var(\hat{F}^p_i) = \sum_j Var(\hat{S}^p_{ji})$. Now we move to deriving the estimator of flow size using flow sampling. Consider s^f_{ji} to be the number of flow-sampled packets of the 5-tuple flow S_{ji}. Hence:

$$\hat{F}^f_i = \frac{\sum_j s_{ji}}{\pi^f_a}. \tag{6.13}$$

This estimator has a variance equal to:

$$Var(\hat{F}^f_i) = \frac{1-\pi^f_a}{\pi^f_a}.\sum_j (Sji)^2 = \frac{1-\pi^f_a}{(\pi^f_a)^2}.\sum_j (sji)^2. \tag{6.14}$$

6.2.1.3 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 Chapter, 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.

6.2.2 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.

6.2.2.1 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:

$$\mathcal{O} = \sum_{M \in \mathcal{R}} \sum_{a \in \Gamma_M} (\pi_a^f . N_a + (1 - \pi_a^f)\pi_a^p . N_a . \Sigma_a), \tag{6.15}$$

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:

$$\overline{N}_a \leftarrow \delta N_a + (1 - \delta)\overline{N}_a, \tag{6.16}$$

$$\overline{\Sigma}_a \leftarrow \delta \Sigma_a + (1 - \delta)\overline{\Sigma}_a, \tag{6.17}$$

where N_a and Σ_a are the last estimations provided by Eq. (6.1) and (6.7) 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 (6.16) and (6.17) and a slightly simplified version of (6.15), 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.

$$\mathcal{O} = \sum_{M \in \mathcal{R}} \sum_{a \in \Gamma_M} (\pi_a^f . \overline{N}_a + \pi_a^p . \overline{N}_a . \overline{\Sigma}_a). \tag{6.18}$$

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 4 explains how the EWMA filter is implemented for prediction according to Eq. (6.18). 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.

6.2.2.2 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 (e.g. flow size estimation, heavy hitter detection), their associated weights γ_i, and 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:

$$U = \sum_i \gamma_i MRE(\hat{T}_i), \tag{6.19}$$

under the following constraints:

$$\mathcal{O} \leq \mathcal{TO}, \tag{6.20}$$

$$p_k \leq SR_{max} \quad \text{and} \quad p_k \geq SR_{min}. \tag{6.21}$$

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 [26], 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

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 $(s_{kji}^p)_{k\in\mathcal{M}}$.
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 $\hat{N}_a = \chi_p \hat{N}_a^p + \chi_f \hat{N}_a^f$;
    \\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.
  $\mathcal{O} = \sum_{M\in\mathcal{M}} \sum_{\mathcal{A}\in\Gamma_M} O_a$;
  return $\{\mathcal{O}\}$
end

Algorithm 4: Overhead prediction method.

a better direction while using a sophisticated line search algorithm to find the best step value. We continue until we either reach the global minimum or we exceed the maximum number of iterations.

6.3 Validation results

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

6.3.1 Comparison with application-specific methods

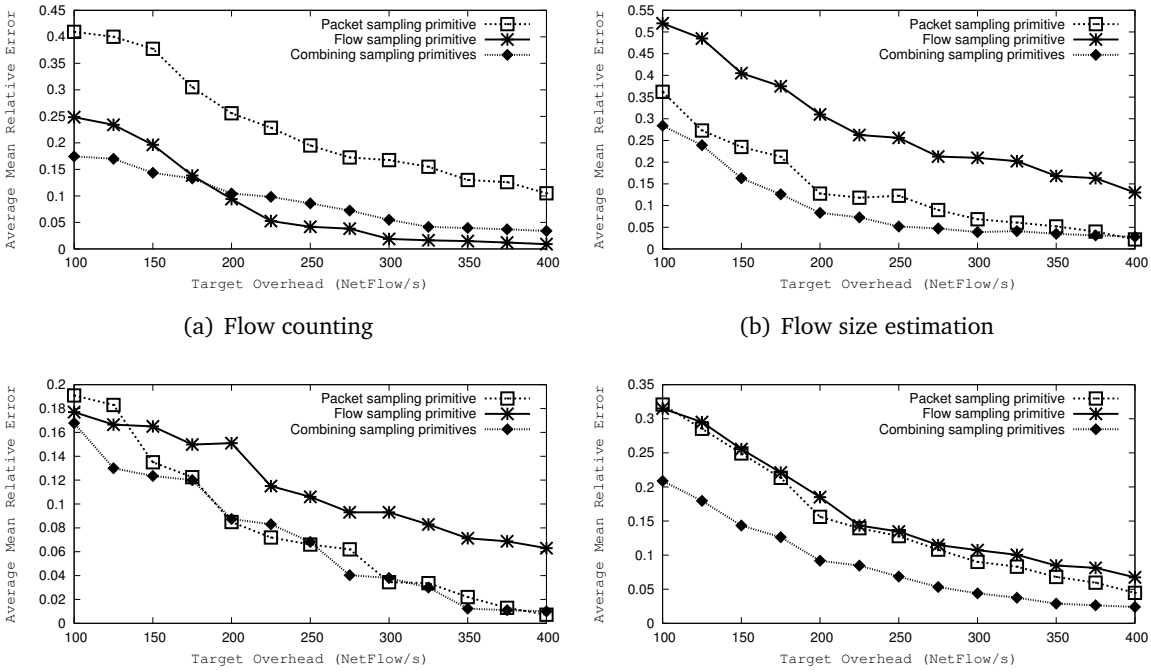
In this section we aim to address the performance of our system using real experiments over our platform. For this experiment, we set the timer d for updating sampling rates to 5 minutes, the time scale τ to 3600s, the minimum possible sampling rate $\mathrm{SR}_{\mathrm{min}}$ to 0.0005 and the maximum possible one $\mathrm{SR}_{\mathrm{max}}$ to 1. The $\mathcal{TO}$ is set to 200 NetFlow-records/s.

We plot in Figure 6.2 the average mean relative error for three specific monitoring applications:

- Flow counting application.
- Flow size estimation.
- Heavy hitter detection.

For these applications, we use our proposed combination method and we compare its performance with two known application-specific solutions i.e., packet sampling primitive and flow sampling primitive.

We plot in Figure 6.2(d) the global accuracy defined as the global weighted utility function described in Equation 6.19. This figure 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 order to have similar global accuracy using application-specific primitive to the one obtained by our method, we have to use a $\mathcal{TO}$ value larger than 150% of the value used by the adaptive solution. However, even in this case, results would be biased against some applications. 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. Hence, using our solution based on combining different sampling primitive measurements, allows not only optimizing resource consumption



(a) Flow counting

(b) Flow size estimation

(c) Heavy hitter detection

(d) Global accuracy

Figure 6.2: Average mean relative error vs. Target overhead ($\mathcal{TO}$) for three applications: Our approach vs. two application-specific approaches.

Table 6.1: 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	assigned weight	AMRE
SC1	0.5	0.0889	0.5	0.114	-	-	1	0.1014
SC2	0.5	0.0738	0.25	0.164	0.25	0.0865	1	0.0995
SC3	0.333	0.1045	0.333	0.083	0.333	0.0674	1	0.0851

but also improving the global accuracy as well as the accuracy of each specific monitoring application. These results are illustrated in Figures 6.2(a),6.2(b) and 6.2(c). We observe that each primitive provides accurate measurements for its specific application and less accurate results for the other applications. For instance, we can see that packet sapling primitive improves accuracy of flow size estimation and heavy hitter detection while it provides a large estimation error for flow counting. We can also notice how 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. These figures show the large impact of the monitoring constraint ($\mathcal{TO}$) on measurement accuracies, we observe the clear reduction of estimation errors when increasing the value of $\mathcal{TO}$.

6.3.2 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. Table 6.1 presents 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 that increasing the weight value assigned to a given monitoring application allows reducing the average mean relative error (AMRE) of its measurement. However this impact is small since the system can profit from measurements provided by other sampling primitives and combine them in order to improve the overall accuracy.

In order to illustrate these results, we present in Table 6.2 the average of sampling rate values as well as the percentage of reported NetFlow records by each sampling primitive. We observe that the value of sampling rates and the number of reported NetFlow records of each primitive depend on the assigned weights to the different applications. In fact, assigning a large weight to a given application increases its impact on the global accuracy. Hence, the system tries

Table 6.2: 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	0.03613	58.23%	0.023	41.77%
SC2	0.0287	62.84%	0.197	37.16%
SC3	0.016	39.65%	0.314	60.35%

to increase sampling rate values of the suitable sampling primitive which allows improving the accuracy of this application. For instance, by increasing the weight value assigned to the flow counting from 0.333 in scenario SC3 to 0.5 in scenario SC1, the system increases the average flow sampling rate from 0.016 to 0.03613 and reduces the AMRE of this task from 0.1045 to 0.0889. This introduces an increase in reported Netflow records of this sampling primitive from 39.65% to 58.23%.

6.3.3 Overhead prediction process validation

In order to evaluate the performance of the overhead prediction method, we plot in Figure 6.3 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 availale resources in order to provide the best possible accuracy. However, the use of a small time scale ($\tau = 600$s) leads to an oscillating behavior of the overhead since the system tracks more details and changes in the traffic. However, tracking changes on a large scale ($\tau = 5400$s) leads to a stable behavior of the overhead. This latter behavior is due to the avoidance of some details and variations in the traffic specific to individual observation period.

6.4 Conclusions

In this Chapter, 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 systems, 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

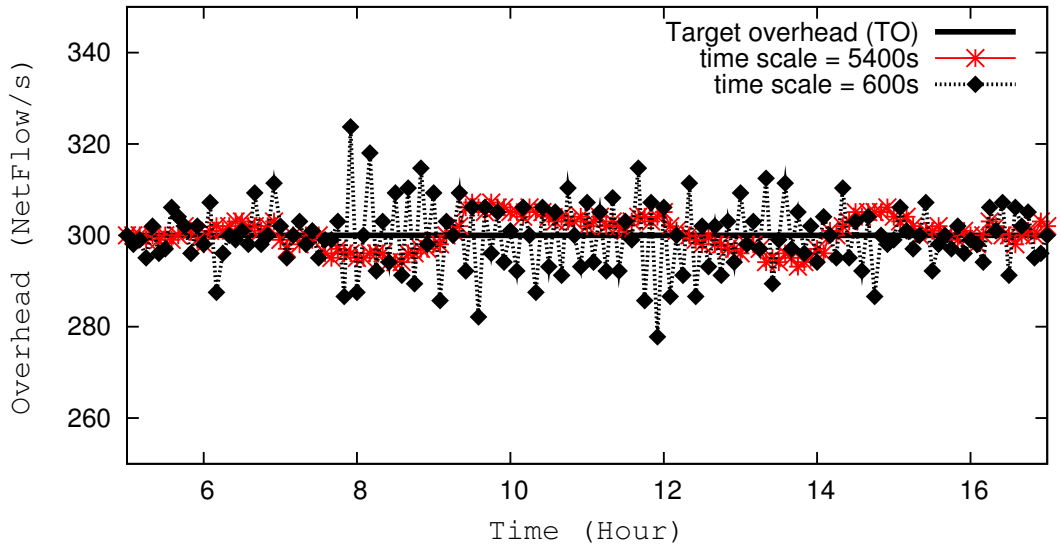


Figure 6.3: Resulting overhead vs. time using two time scales to track traffic variations

applications requirements and network conditions.

7

CONCLUSION

Contents

This chapter concludes our thesis. In Section 7.1 we will make a summary of our work and restate the main findings. Section 7.2 contains a number of proposals for a possible future work that will extend our network-wide monitoring system.

7.1 Overall closure comments

Traffic measurement allows network operators to achieve several purposes such as traffic engineering, network resource provisioning and management, accounting and anomaly detection. Although many monitoring solutions exist today, the number Network engineers and operators are faced with a number of challenges that arise in the context of network monitoring and measurement. In fact, existing solutions, either lack the necessary detail, do not scale to high enough speeds, fail to extract the meaningful information from raw data or are not flexible enough to keep up with the ever changing traffic mix. Our research work focused on an advanced network-wide monitoring system which adopts a centralized approach in order to address these shortcomings. Our system supports a central unit that deals with existing monitor-

ing tools in order to provide visibility over the entire network. It should adjust its configuration according to network conditions, measurements accuracy and monitoring constraints.

Nonetheless, we have started by presenting the projects related to our work and we have compared, from different points of view, all these various approaches. Thus, we were able to derive the following important remarks. The first underscores the fact that the vendors do not want to implement sophisticated sampling schemes that give good results under certain circumstances. They want to implement simple and robust solutions that are well described by some form of a standard (i.e. sFlow, NetFlow). Thus, we decided to design a new solution that deals with existing monitoring tools and tries to coordinate responsibilities between the different monitors in order to improve the overall accuracy. The second one requires that the monitoring system should adopt a centralized approach in order to provide visibility over the entire network and to investigate the local measurements and correlate their results in order to address the tradeoff between accuracy and monitoring constraint. And the last one indicates that the network configuration module should provide a responsive solution able to detect changes in network conditions and adapt the different sampling rates to network state. At the same time it should avoid unnecessary details and oscillation in the traffic in order to keep the resulting overhead within the desired bounds.

Based on this survey, we have designed a centralized network-wide monitoring architecture that deals with existing monitoring tools (NetFlow) and coordinates responsibilities among the different monitors in order to achieve multi-monitoring objectives. We have detailed each of the constituent modules and their interfaces, and we have presented the various kinds of decision algorithms.

We have pursued our work by introducing a new approach for the emulation of Internet traffic and for its monitoring across the different routers, through which, we put at the disposal of users a real traffic emulation service coupled to a set of libraries and tools capable of Cisco NetFlow data export and collection, which are meant to run advanced applications for network wide monitoring and optimization. Our platform offers a complete set of features towards the development and evaluation of solutions for network monitoring and management. Namely, it offers the possibility to reproduce real backbone network topology, to monitor and sample the packets being forwarded in a given router and finally the ability to analyze the collected flows. Our platform allows users to remotely tune the sampling rate of a given monitor. Users can also easily introduce traffic engineering methods within the monitors. Furthermore, we provide an exhaustive analytical sensitivity analysis in order to characterize, qualitatively or quantitatively, what impact a particular input parameter has on a system output. In other words, using Sensitivity Analysis, one can determine how changes in one or several parameters will impact the target output variable.

Then, we have concentrated our discussion on another important mechanism needed within

the future monitoring systems, which is the network reconfiguration method. We have proposed two different network reconfiguration methods in order to address the tradeoff between responsiveness and smoothness defined as avoidance of the introduction of unnecessary noise and details in the traffic. we have first designed a Reactive network reconfiguration method. It automatically reconfigures the sampling rates in the different monitors according to the monitoring application requirements and resource consumption constraints. It proceeds in optimizing the configuration and adjusting sampling rates in small steps based on dynamics inspired from the one used by TCP for the adjustment of its congestion window and using the gradient Projection Method (GPM) to identify the monitors that we should reconfigure until the optimal configuration is reached. For the case of estimating the full traffic matrix, experimental results proved the ability of the reactive optimization method to continuously improve the monitoring accuracy while limiting the overhead to its target value. Moreover, it provides a fair allocation of sampling rates over monitors so that measurement errors are homogenously distributed among flows independently of their volumes. Compared to static edge configuration, our network-wide adaptive system has shown its advantages in better capturing network flows especially for small flows. Next, we have presented a Proactive network reconfiguration method. This optimization method consists of: an overhead prediction based on an Exponential Weighted Moving Average filter to track long-term and short-term variations in the traffic, a global weighted utility function to deal with multiple monitoring tasks at the same time, and an optimization algorithm that configures monitors to address the tradeoff between resource consumption and accuracy of different tasks. Results proved the ability of the Proactive optimization method to keep the resulting overhead around a target value. We also demonstrated that the proposed method is practical: it provides an efficient method to achieve multiple monitoring objectives using a weighted utility function and it relies on a flexible method to track variations in the traffic according to an adaptable time scale. Furthermore, we have compared the performance of the two different network reconfiguration method (i.e., Reactive and Proactive optimization methods) described previously. We study the performance of these optimization methods in terms of measurement accuracy, resulting overhead and responsiveness to changes in the network traffic. Experimental results showed the ability of the reactive optimization method to track changes in network traffic conditions. It is suited for monitoring applications that depend on the communication structure e.g., security applications and anomaly detection application that analyze volume structure. These monitoring applications need a continuous analyze of the network traffic. In order to detect and react to network changes, the optimization should track all the details in network traffic. However, this leads to an oscillating behavior and can degrade the accuracy of measurements. The Proactive optimization method provides more smoothness and reacts less aggressively to changes in network traffic changes. This solution is suited to monitoring applications that require an understanding of volume structure; e.g., heavy-hitter

detection and traffic engineering that require an understanding of the number of packets/bytes per-port or per-src. These monitoring applications don't require tracking all changes and details in the network traffic. We need just to fix a time scale to track changes in network traffic. Once done the optimization method reconfigure monitors accordingly.

Furthermore, we have introduced 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. 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 $\mathcal{TO}$.

7.2 Future work

In the previous section we have summarized the contributions of this dissertation and having highlighted their efficiency and wide applicability. Next, we will discuss future directions this work can lead to.

We argue that using our adaptive network-wide monitoring system, monitoring applications will be easier to achieve. In fact, adding more monitoring applications is very easy in the context of our approach; one has to correctly define these monitoring applications. Hence, it is exciting to validate our system with more monitoring applications. For instance, the flow size distribution, the tracking of some user-specific flows and the detection of anomalies are among the applications we want to cover.

The distribution of the control and adaptive overhead tuning are other interesting objectives to realize. In this dissertation we have focused on building a centralized network-wide monitoring system that collects measurements from the different monitors deployed in network routers to build global more accurate results and to provide visibility over the entire network. We believe that distributing the control can clearly reduce the amount of exported data and optimize the resource consumption cost. It would be interesting to see how our methods could be extended to a distributed system. There are many interesting ways to achieve this objective. For instance, we can apply the game theory to identify which monitors we should reconfigure.

Finally, the current experimentations have been done on emulated topology networks with virtual routers and virtual links. However, we can deploy our experimental platform Monlab on real network nodes. Hence, running more realistic experimentations on real network topologies with real routers is considered future work.

BIBLIOGRAPHY

[1] Abilene or internet2: The us research and acamdemic backbone. http://www.internet2.edu/network/. 84

[2] Cisco netflow. http://www.cisco.com/. 47

[3] Flowd small fast and secure netflow collector. http://www.mindrot.org/projects/flowd/. 53

[4] . Geant: The european research and academic backbone. http://www.geant.net/. 84, 100

[5] Google apps. http://www.google.com/apps. 6

[6] Is the skype outage really a big deal? http://news.cnet.com/8301-107843-9761673-7.html/. 6

[7] Mawi working group traffic archive. http://tracer.csl.sony.co.jp/mawi/. 48, 55, 84

[8] . Monlab: Emulation platform for network wide traffic sampling and monitoring. http://planete.inria.fr/MonLab. 9, 11, 47, 48, 82, 106

[9] Random sampled netfow. http://www.cisco.com/. 28

[10] Salesforce: Crm software solutions and enterprise cloud computing. http://www.salesforce.com. 6

[11] sflow home page. http://sflow.org. 28

[12] . Softflowd flow-based network traffic analyser. http://www.mindrot.org/projects/softflowd/. 51

[13] Tcpdpriv: A program for eliminating confidential information from packets. http://ita.ee.lbl.gov/html/contrib/tcpdpriv.html/. 55, 84

[14] Tcpdump/libpcap. `http://www.tcpdump.org/`. 50, 52

[15] . Xml description of the geant topology. `http://planete.inria.fr/GEANT`. 54

[16] . Yaf: Yet another flowmeter. `http://tools.netsa.cert.org/yaf`. 48

[17] Greenberg A., Hjalmtysson G., Maltz D. A., Meyers A., Rexford J., Xie G., Yan H., Zhan J., and Zhang H. A clean slate 4d approach to network control and management. In *ACM SIGCOMM CCR*, 2005. 8

[18] Lakhina A., Crovella M., and Diot C. Characterization of network-wide anomalies in traffic flows. In *In ACM Internet Measurement Conference*, 2004. 17

[19] Lakhina A., Crovella M., and Diot C. Diagnosing network-wide traffic anomalies. In *In ACM SIGCOMM*, 2004. 4, 8, 17, 18

[20] Lakhina A., Crovella M., and Diot C. Mining anomalies using traffic feature distributions. In *In ACM SIGCOMM*, 2005. 17, 18, 48

[21] M. Allman, V. Paxson, and W. Stevens. Tcp congestion control. internet engineering task force. In *RFC 2581*, 1999. 5

[22] N. Alon, N.G. Duffield, C. Lund, and M. Thorup. Estimating sums of arbitrary selections with few probes. In *PODS*, 2005. 37

[23] Paul D. Amer and Lillian N. Cassel. Management of sampled real-time network measurements. In *14th Conference on Local Computer Networks*, 1989. 28

[24] S. Bellovin. Security aspects of napster and gnutella. In *Invited Talk at USENIX Annual Technical Conference*, 2001. 6

[25] L. Bernaille, R. Teixeira, and K. Salamatian. Early application identification. In *in proceedings of the 2nd CoNEXT Conference*, 2006. 48

[26] S. Boyd and L. Vandenberghe. Convex optimization. In *Cambridge University Press*, 2004. 100, 120

[27] D. Brauckhoff, A. Wagner, and M. May. Flame: A flow-level anomaly modeling engine. In *in proceedings of CSET*, 2008. 47

[28] G.R. Cantieni, G. Iannaccone, C. Barakat, C. Diot, and P. Thiran. Reformulating the monitor placement problem: Optimal networkwide sampling. In *Proc. of CoNeXT*, 2006. 7, 14, 41, 42, 96, 112, 113

[29] Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. On random sampling over joins. In *In ACM SIGMOD*, 1999. 41

[30] Cisco. Netflow services and applications. *White Paper*, 2000. 6, 13, 21, 30, 47

[31] K.C. Claffy, George C. Polyzos, and Hans-Werner Braun. Application of sampling methodologies to network traffic characterization. In *Proceedings of ACM SIGCOMM*, 1993. 28

[32] B. CLAISE. Cisco systems netflow services export version 9. *RFC3954*. 6, 13, 21, 30

[33] M. P. COLLINS and M. K. REITER. Finding peer-to-peer file-sharing using coarse network behaviors. In *in proceedings of ESORICS*, 2006. 4

[34] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. Introduction to algorithms. In *MIT Press and McGraw-Hill*, 2001. 51

[35] C. Cranor, T. Johnson, O. Spataschek, and V. Shkapenyuk. Gigascope: a stream database for network applications. In *Proc. of the ACM SIGMOD*, 2003. 13, 21

[36] R.I. Cukier, C.M. Fortuin, K.E. Shuler, A.G. Petshek, and J.H. Schaibly. Study of the sensitivity of coupled reaction systems to uncertainties in rate coefficients. *The Journal of Chemical Physics*, 1973. 58, 94

[37] R.I. Cukier, J.H. Schaibly, and K.E. Shuler. Study of the sensitivity of coupled reaction systems to uncertainties in rate coefficients. iii analysis of the approximations. *The Journal of Chemical Physics*, 63, 1975. 58, 61, 94

[38] N. Duffield and M. Grossglauser. Trajectory sampling. *White Paper*, 2003. 38

[39] N. Duffield and C. Lund. Predicting resource usage and estimation accuracy in an ip flow measurement collection infrastructure. *ACM SIGCOMM Internet Measurement Conference*, 2003. 36, 37

[40] N. Duffield, C. Lund, and M. Thorup. Charging from sampled network usage. *ACM SIGCOMM Internet Measurement Workshop*, 2001. 4, 36

[41] N. Duffield, C. Lund, and M. Thorup. Properties and prediction of flow statistics from sampled packet streams. In *Proc. of IMC*, 2002. 21, 36, 74

[42] N. Duffield, C. Lund, and M. Thorup. Estimating flow distributions from sampled flow statistics. In *Proc. of the ACM SIGCOMM Conference on Applications, Technologies, Architectures*, 2003. 21

[43] N. Duffield, C. Lund, and Mikkel Thorup. Flow sampling under hard resource constraints. *SIGMETRICS*, 2004. 6, 36, 37, 41

[44] N. G. Duffield, A. Gerber, and M. Grossglauser. Trajectory engine: A backend for trajectory sampling. In *In Proc. Network Operations and Management Symposium (NOMS)*, 2002. 38

[45] N. G. Duffield and M. Grossglauser. Trajectory sampling for direct traffic observation. In *IEEE/ACM Transactions on Networking*, 2001. 38, 39

[46] N. G. Duffield and M. Grossglauser. Trajectory sampling with unreliable reporting. In *IEEE Infocom*, 2004. 40

[47] N.G. Duffield, C. Lund, and M. Thorup. Learn more, sample less: control of volume and variance in network measurement. *IEEE Transactions in Information Theory*, 51:1756–1775, 2005. 21, 22, 25, 36, 41

[48] N.G. Duffield, C. Lund, and M. Thorup. Optimal combination of sampled network measurements. In *IMC 2005*, 2005. 6, 7, 14, 21, 41, 48, 73

[49] C. Estan, K. Keys, D. Moore, and G. Varghese. Building a better netflow. In *Proc. of ACM SIGCOMM*, 2004. 21, 22, 23, 24, 25

[50] C. Estan and G. Varghese. New directions in traffic measurement and accounting. In *In Proc. ACM SIGCOMM*, 2002. 35

[51] C. Estan and G. Varghese. New directions in traffic measurement and accounting. In *Proc. of ACM SIGCOMM*, 2002. 112

[52] Cristian Estan. Internet traffic measurement: What's going on in my network? In *Ph. D. thesis at University of California San Diego*, 2003. 4

[53] Cristian Estan and George Varghese. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. In *in journal of ACM Transactions on Computer Systems (TOCS)*, 2003. 34

[54] A. Feldmann, A. Gilbert, P. Huang, and W. Willinger. Dynamics of ip traffic: A study of the role of variability and the impact of control. In *in proceedings of ACM SIGCOMM*, 1999. 48

[55] A. FELDMANN, A¿ G. GREENBERG, C. LUND, N. REINGOLD, J. REXFORD, and F. TRUE. Deriving traffic demands for operational ip networks: Methodology and experience. In *in proceedings of ACM SIGCOMM*, 2000. 4

[56] A. Feldmann, J. Rexford, and R. Caceres. Efficient policies for carrying web traffic over flow-switched networks. *IEEE/ACM Transactions on Networking*, 1998. 36

[57] Ballani H. and Francis P. Conman: A step towards network manageability. In *In Proc. of ACM SIGCOMM*, 2007. 8

[58] Edwin A. Hernandez, Matthew C. Chidester, and Alan D. George. Adaptive sampling for network management. In *Journal of Network and Systems Management*, 2001. 41

[59] N. Hohn and D. Veitch. Inverting sampled traffic. In *In Proc. IMC*, 2003. 33, 48

[60] N. Hohn and D. Veitch. Inverting sampled traffic. In *Proc. of IMC*, 2003. 112

[61] Chadi Barakat Imed Lassoued, Amir Krifa and Konstantin Avrachenkov. Network-wide monitoring through self-configuring adaptive system. In *in proceedings of IEEE INFOCOM*, 2011. 9, 10, 73

[62] Leydon John. P2p swamps broadband networks. In *whitepaper*, 2002. 6

[63] A. Johnson and J. Quittek. Packet sampling (psamp) protocol specifications. *RFC5476*, 2009. 29, 30

[64] Ryszard Erazm Jurga and Mi losz Marian Hulboj. Technical report packet sampling for network monitoring. In *CERN-HP Procurve openlab project*, 2007. 32

[65] K. Keys, D. Moore, and C. Estan. A robust system for accurate real-time summaries of internet traffic. In *In Proc. SIGMETRICS*, 2005. 34, 35, 36

[66] K. Keys, D. Moore, and C. Estan. A robust system for accurate real-time summaries of internet traffic. In *Proc. of SIGMETRICS*, 2005. 113

[67] Ramana Rao Kompella and Cristian Estan. The power of slicing in internet flow measurement. In *in proc. IMC '05 Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, 2005. 36

[68] A. Krifa, I. Lassoued, and C. Barakat. Emulation platform for network wide traffic sampling and monitoring. *TRAC*, 2010. 9, 47, 48, 82, 106

[69] Imed Lassoued and Chadi Barakat. Adaptive multi-task monitoring system based on overhead prediction. In *in proceedings of the ACM CoNext PRESTO workshop on Programmable Routers for Extensible Services of Tomorrow*, 2010. 9, 10, 73

[70] Imed Lassoued and Chadi Barakat. A multi-task adaptive monitoring system combining different sampling primitives. In *in proceedings of the 23 International Teletraffic Congress (ITC)*, 2011. 9, 10, 73

[71] Myungjin Lee, Mohammad Y. Hajjat, Ramana Rao Kompella, and Sanjay G. Rao. Relsamp: Preserving application structure in sampled flow measurements. In *Proc. of IEEE Infocom*, 2011. 6

[72] Y. Lu and S. Mohanty. Sensitivity analysis of a complex, proposed geologic waste disposal system using the fourier amplitude sensitivity test method. *Reliab. Eng. syst. Safe.*, 72, 2001. 59

[73] Keith McCloghrie and Marshall T. Rose. Management information base for network management of tcp/ip-based internets. In *RFC 1213*, 1991. 13, 16, 20

[74] A. Medina, N. Taft, K. Salamatian, S. Bhattacharyya, and C. Diot. Traffic matrix estimation: Existing techniques and new directions. In *In Proceedings of the ACM SIGCOMM*, 2002. 17

[75] M. Molina, S. Niccolini, and N.G. Duffield. A comparative experimental study of hash functions applied to packet sampling. In *ITC-19*, 2005. 38

[76] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. The spread of the sapphire/slammer worm. In *CAIDA Technical Report*, 2003. 6

[77] D. Moore and C. Shannon. The spread of the witty worm. In *In IEEE Security and Privacy*, 2004. 6

[78] N. Naoumov and K. Ross. Exploiting p2p systems for ddos attacks. In *In International Workshop on Peer-to-Peer Information Management*, 2006. 6

[79] P. Phaal, S. Panchen, and N. McKee. Inmon corporation's sflow: A method for monitoring traffic in switched and routed networks. *RFC3176*, 2001. 6, 13, 21, 30

[80] J. Quittek, T. Zseby, B. Claise, and S. Zander. Requirements for ip flow information export (ipfix). *RFC3917*, 2004. 29

[81] G. Sadasivan, N. Brownlee, B. Claise, and J. Quittek. Architecture for ip flow information export. *RFC5470*, 2009. 29, 30

[82] A. Saltelli, K. Chan, and M. Scott. Sensitivity analysis. *Wiley*, 2000. 58, 94

[83] A. Saltelli, M. Ratto, T. Andres, F. Campolongo, J. Cariboni, D. Gatelli, M. Saisana, and S. Tarantola. Global sensitivity analysis: The primer. *John Wiley and Sons*, 2008. 58

[84] A. Saltelli, S. Tarantola, F. Campolongo, and M. Ratto. Sensitivity analysis in practice: A guide to assessing scientific models. *Wiley*, 2004. 58, 94

[85] A. Saltelli, S. Tarantola, and K. P.-S. Chan. A quantitative model-independent method for global sensitivity analysis of model output. *Technometrics*, 41:39–56, 1999. 59

[86] V. Sekar, M.K. Reiter, W. Willinger, H. Zhang, R.R. Kompella, and D.G. Andersen. cSamp: A system for network-wide flow monitoring. In *Proc. 5th USENIX NSDI*, 2008. 7, 14, 41, 48, 96, 113

[87] I.M. Sobol. Sensitivity estimates for nonlinear mathematical models. *Math. Model. Comput. Exp.*, 1:407–414, 1993. 60

[88] S. Staniford, D. Moore, V. Paxson, and N. Weaver. The top speed of flash worms. In *In Proc. ACM Workshop on Rapid Malcode (WORM).*, 2004. 6

[89] K. Suh, Y. Guo, J. Kurose, and D. Towsley. Locating network monitors: Complexity heuristics and coverage. In *Proc. of IEEE Infocom*, 2005. 7, 8, 42, 48

[90] M. Thottan and C. Ji. Anomaly detection in ip networks. In *IEEE Trans. Signal Processing*, 2003. 48

[91] Tobias Oetiker and Dave Rand. Mrtg: Multi router traffic grapher. http://people.ee.ethz.ch/oetiker/webtools/mrtg/. 16

[92] R. Torres, M. Hajjat, S. Rao, M. Mellia, and M. Munafo. Inferring undesirable behavior from p2p traffic analysis. In *In ACM SIGMETRICS*, 2009. 6

[93] Hui Zhang Vyas Sekar, Michael K Reiter. Revisiting the case for a minimalist approach for network flow monitoring. In *Proc. of IMC*, 2010. 32, 33, 35

[94] Hui Zhang Vyas Sekar, Michael K Reiter. Revisiting the case for a minimalist approach for network flow monitoring. In *Proc. of IMC*, 2010. 113, 114

[95] Carey L. Williamson. Internet traffic measurement. In *Tutorial article at the Department of Computer Science University of Calgary. IEEE Internet Computing*, 2001. 4, 6

[96] Li X., Bian F., Zang H., Diot C., Govindan R., Hong W., and Iannaccone G. Mind: A distributed multi-dimensional indexing system for network diagnosis. In *In IEEE INFOCOM*, 2006. 8, 17, 20

[97] Li X., Bian F., Crovella M., Diot C., Govindan R., Iannaccone G., and Lakhina A. Detection and identification of network anomalies using sketch subspaces. In *In ACM Internet Measurement Conference*, 2006. 17, 20

[98] Y. XIE, V. SEKAR, D. A. MALTZ, M. K. REITER, and H. ZHANG. Worm origin identification using random moonwalks. In *in proceedings of IEEE Symposium on Security and Privacy*, 2005. 4

[99] Zhang Y., Roughan M., Duffield N., and Greenberg A. Fast accurate computation of large-scale ip traffic matrices from link loads. In *In Proc. of ACM SIGMETRICS*, 2003. 8

[100] T. Zseby, T. Hirsch, and B. Claise. Packet sampling for flow accounting: Challenges and limitations. In *PAM*, 2008. 21

[101] T. Zseby, M. Molina, N. Duffield, S. Niccolini, and F. Raspall. Sampling and filtering techniques for ip packet selection. In *RFC 5475*, 2009. 27, 28, 29