# On Improving DREAM Framework with Estimations and ProgME

by

Rene Hernandez Remedios

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements
For the M.C.S. degree in
Computer Science

Ottawa - Carleton Institute for Computer Science
School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

# Abstract

Software Defined Networking (SDN) is an emerging architecture that is dynamic, manageable, cost-effective and adaptable, making it ideal for the high-bandwidth, dynamic nature of today's applications. Using SDN, networks can enable a variety of concurrent, dynamically instantiated measurement tasks, that provide fine-grain visibility into network traffic by configuring Ternary Content Address Memory (TCAM) counters in hardware switches. However, TCAM memory is limited, thus the accuracy of measurement tasks depends on the number of resources devoted to them on each switch.

In this thesis, we propose a solution that improves Dynamic Resource Allocation for Software-defined Measurements (DREAM), a framework with an adaptive step size search that achieves a desired level of accuracy for measurement tasks. We have enabled prediction capabilities in the framework to generate better counters configurations using previous network traffic information. We implement four estimation techniques (EWMA-based Prediction, Polynomial Curve Fitting, KMeans++ Cluster and Pseudo-Linear Extrapolation) that have been tested with simulations running three types of measurement tasks (heavy hitters, hierarchical heavy hitters and traffic change detection) that show the proposed techniques improve task accuracy and tasks concurrency in DREAM.

Existing traffic measurements tools usually rely on some predetermined concept of *flows* to collect traffic statistics. Thus, they usually have issues in adapting to changes in traffic condition and present scalability issues with respect to the number of flows and the heterogeneity of the monitoring applications.

We propose an integration of the Programmable MEasurements (ProgME) paradigm, which defines a novel approach to defined measurement tasks in a programmable way using the concept of flowsets, on top of the DREAM framework. This enables better scalability for measurement tasks that deal with large amounts of traffic flows on DREAM while reducing the required number of counters allocations for the tasks.

**Acknowledgements**

This thesis has been an amazing journey from the beginning until the very end. I take this opportunity to express my gratitude to the Faculty of Graduate and Postdoctoral Studies, University of Ottawa, for giving me the chance to develop and fulfill this research venture.

First, I want to express my gratitude to my sister, Ingrid and my brother-in-law Marcel, for helping me during these two years. Without them, this thesis would have not been possible.

I am also truly thankful to Dr. Amiya Nayak, my thesis supervisor, for his invaluable guidance and persistent support through all the project steps. His critical evaluations have helped keeping me on the right track towards the completion of this thesis.

I would like to thank my wife, Mairelys, for her emotional comfort and dedication. I also convey my recognition to my family, specially my parents, Rene and Gilda, and my cousin Elio for their assist and best wishes, even from far away.

Finally, I just want to say thank you to my roommates, Raudel and Gabi, for sharing this journey with me.

# Contents

# List of Figures

# List of Tables

# List of Algorithms

# Chapter 1

# Introduction

Computer networks are composed of a large number of components such as routers, switches and diverse types of middleboxes (i.e equipment that manipulate traffic for purposes other than packet forwarding, such as firewall and network address translators) with many complex protocols implemented on them.

The traditional structure of data networks has been largely hardware-centric. Because of its huge deployment base and the fact that it is considered part of our society's critical infrastructure, networks and the Internet in general, face great challenges to evolve both in terms of its physical framework as well as its protocols and software performance [51]. However, current trends and patterns in Internet applications and services are becoming increasingly complex and demanding, thus making compulsory to address these existing challenges.

Several approaches have been suggested to address these limitations, such as Network Virtualization (NV), Network Function Virtualization (NFV) and SDN. By leveraging software-based solutions, these approaches aim to facilitate innovation in network environments and to enable different concepts [41]:

- **Sharing** A resource big enough can be divided into multiple virtual pieces. Virtual Machines (VMs) can be run in different processors and used by separated users.

- **Isolation** Multiple users sharing a resource may not trust each other; by enabling isolation among users, systems can monitor activities and avoid user interferences.
- **Aggregation** The combination of small resource makes it possible to build large reliable virtual resources that behave as a single unit to the user.
- **Dynamics** Resource requirements can change fast due to user mobility. When resources are virtual, allocation is usually faster and easier to manage.
- **Ease of management** Based on the facts that virtual resources are software-based and expose a uniform interface through standard abstractions.

## 1.1 Network Virtualization (NV)

NV is defined by the decoupling the roles of the traditional Internet Service Provider (ISP) into the following two independent entities:

- **Infrastructure providers (InPs)** that manage the physical infrastructure.
- **Service providers (SPs)** that create virtual networks (VNs) by aggregating capabilities from different InPs and offer end-to-end services.

In such environments, it is possible to proliferate heterogeneous network architectures free of the inherent limitations of the existing Internet [29].

## 1.2 Network Function Virtualization (NFV)

NFV transforms how the operators architect their infrastructure and services by using virtualization to separate software instances from hardware platforms and by decoupling functionality from location for faster services provisioning [39]. In essence, networks functions are implemented through virtualization techniques and can be instantiated on

demand without requiring the installation of new equipments. An example of this could be running a software-based firewall in a VM on a x86 platform [60].

## 1.3 Software Defined Networking (SDN)

SDN is the latest revolution in networking innovations. All components of the networking industry, including equipment vendors, Internet service providers, cloud service providers and users, are working in various aspects of SDN [41].

The Open Networking Foundation (ONF), an organization dedicated to the promotion of SDN, defines SDN as the physical separation of the control plane from the forwarding plane (data plane) in traditional networks [38]. This paradigm offers an open interface layer between the packet forwarding hardware and the network operating system that runs on the hardware, as shown in Figure 1.1. Therefore, it enables the decoupling of the network into a programmable network control layer and an abstract underlying infrastructure for applications and network devices.

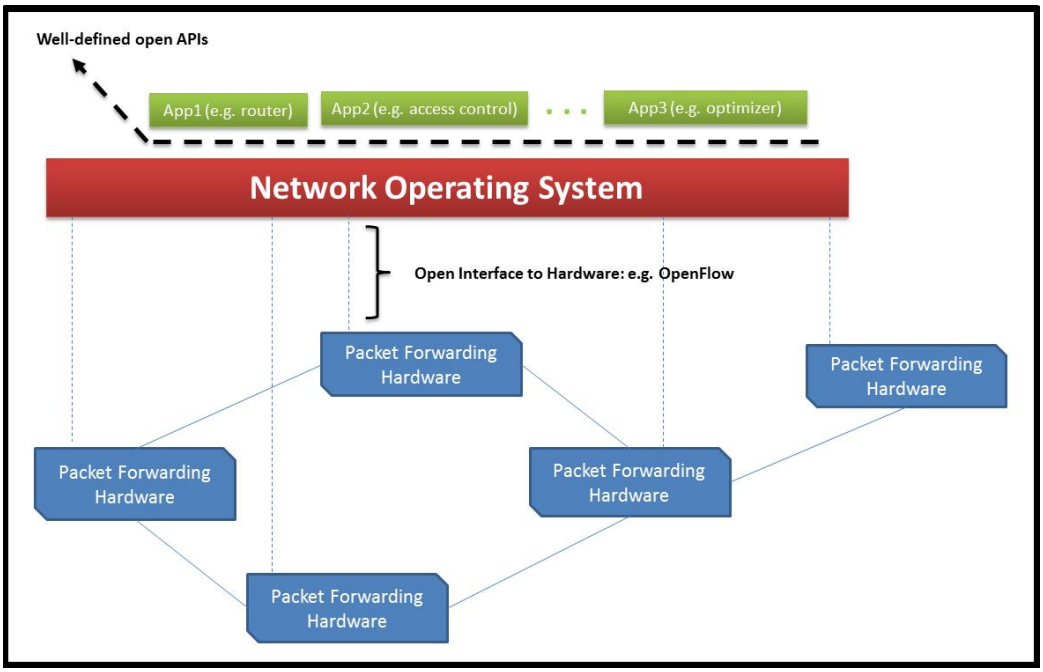



Figure 1.1: Concept of software-defined network [62]

The network control plane in SDN is centralized as a piece of software called *controller* or the *network operating system*, as shown in Figure 1.1. The controller keeps global information of the network and this data can be accessed by some well-defined open APIs to be used by different applications. Thus, customization of the network can be dynamically and automatically enforced by different policies to improve its usability and performance [62].

## 1.4 Traffic Measurements

Traffic measurement plays an essential role for the performance of a wide range of network management tasks, including traffic engineering, traffic accounting, load balancing and performance diagnosis [31]. A measurement tool, be it hardware-based or software-based, collects statistics of network traffic. Application use these information to make network control decisions, such as traffic re-routing.

Traffic measurement refers to the process of counting the number of packets (or bytes) that matches some criteria over certain period of time. The traditional measurement architecture focuses on per-flow statistics, by finding a matching flow for every packet analyzed. These statistics can be used to extract insights of the network behavior by a post-processing mechanism applied by a management application.

The ability to measure different types of network traffic at different time-scales is very critical for tasks such as traffic engineering and congestion detection to guarantee application performance. Usually, network devices, such as switches and routers, are inflexible to deal with different types of traffic due to the implementations of routing rules [44]. The development of SDN and its utilization for traffic measurements promises to overcome these deficiencies.

## 1.5 Motivation

Traffic measurements based on per-flow analysis are well-known and implemented in many systems networks. Although this traditional approach has some success in offering insights about network traffic, the scalability of this architecture is limited in practice. First, it is based on the inflexible definition of flow[1]. Nowadays, in any high speed network, the number of flows can easily reach millions; and this makes keeping track of per-flow traffic a stressful situation from a memory and processor viewpoints. Furthermore, this architecture takes a post-processing approach, which means that measurement tools have little-to-no knowledge of the metrics requirements. It is up to the management application to aggregate the traffic statistics to provide meaningful information. Another concern is how well an implementation of the architecture can respond to changes in the network traffic conditions, e.g., going from measuring large flows (*elephants*) to small flows (*mice*) [37].

Newer approaches to software-defined measurements focus on more adaptive and flexible techniques, in search for better scaling and accuracy of the measurement tasks. Several works, such as DREAM [54], focus on dynamic resource allocation (where resources mean counters present in network switches). Other ideas, such as ProgME [37], propose new alternatives for traffic pattern definitions which improve measurement scalability with respect to traffic and allow more heterogeneity of monitoring applications.

By combining both ideas, it could be possible to develop a more general solution, where tasks can ask for resource allocation in a dynamic way and at the same time provide a flexible specification for the set of IPs that should be consider. Moreover, it could be useful to provide ways to enhance measurement tasks accuracy with the usage of previous traffic data.

For such scenario, the following questions drive the development of this research:

- How can previous traffic information be used to enhance the accuracy of the tasks

[1] A flow can be defined in several ways. But once it is chosen for a measurement task, it is normally fixed.

on DREAM?

- How to provide a more flexible filter specification for tasks, such as in the Programmable MEasurements (ProgME) proposal?

## 1.6 Contributions

In this thesis, we study different frameworks and approaches to develop better solutions for traffic measurements in SDN. The major contributions of this research to improve traffic measurements are listed below:

- We have proposed an estimation-based approach to use existent traffic data to improve the accuracy of measurement tasks on DREAM. Several estimation techniques are implemented to validate the feasibility of the proposal.

- We have proposed an implementation of the ProgME architecture on top of the DREAM framework. Therefore, it makes DREAM a more extensible and complete solution suitable to perform a wider range of traffic measurement analysis.

## 1.7 Organization

The rest of the thesis is organized as follows:

- Chapter 2 presents an analysis of measurements in SDN. The current state of the art is provided, with a focus on DREAM framework, the ProgME paradigm and related ideas. The chapter also identifies the key challenges that remain at the forefront of this field of research.

- Chapter 3 introduces a discussion on how to use existing traffic data to enhance the accuracy results of tasks running on DREAM. Several approaches are discussed, and the key elements of the implemented solution are analyzed.

- Chapter 4 provides a discussion on the ProgME implementation within the DREAM framework. It gives a flexible IP's filters definition to keep track of network traffic and the fundamental solution details are also highlighted.

- Chapter 5 discusses the results of this work. It includes comparison against previous results as well as tests that show the validity of the proposal.

- Chapter 6 gives the concluding comments of our work. Future extension of the research are also mentioned.

# Chapter 2

# Background and Related Work

This chapter provides a background of traffic measurements and its application in the context of the Software Defined Networking (SDN) paradigm. It also reviews existing studies of software-aware measurement analysis to establish the core principles for this thesis. The chapter is divided in seven sections. Section 2.1 illustrates the general background of traffic measurements. Section 2.2 discusses the characteristics of the traffic analysis. Section 2.3 examines the SDN paradigm and how traffic networks can be implemented on top of it. Section 2.4 provides an state of the art analysis for SDN-enabled traffic analysis. Section 2.5 reviews relevant studies that relate to resource allocation and programmable architectures. Section 2.6 outlines the main challenges currently open for the field of traffic measurement in SDN-enabled networks. Finally, Section 2.7 provides a summary of the main ideas discussed in this chapter.

## 2.1 Background

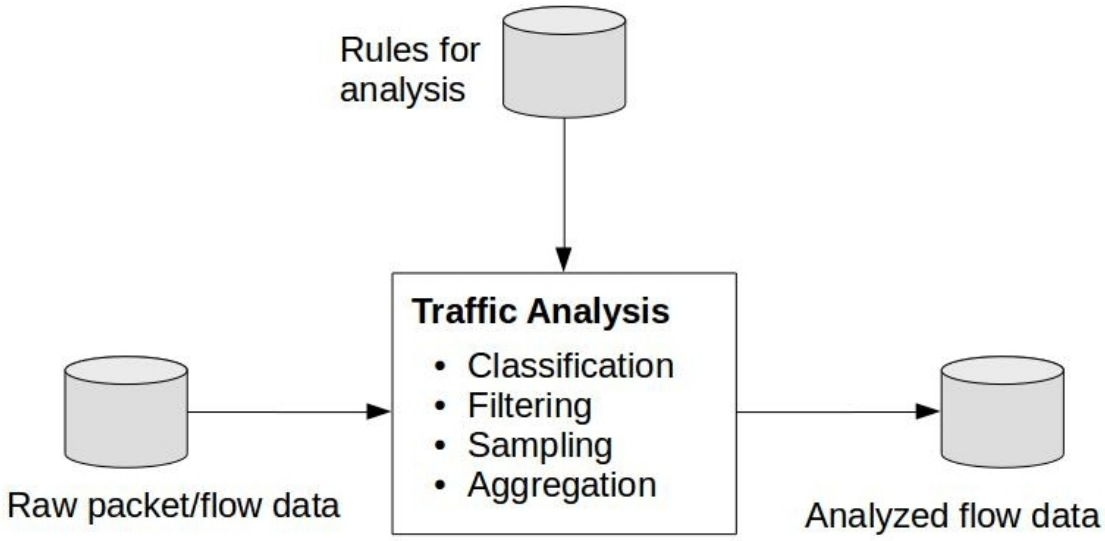



Figure 2.1: Overview of a traffic measurement system

Characterization and measurement of Internet traffic has become over the past few years one of the major challenging issues in network operations [9]. It relies on an in-depth understanding of the composition and the dynamics of the traffic, which is essential in the management and supervision of the networks. Furthermore, the increasing capacity and availability of Internet connected devices has led to the apparition of more complex behavior for typical users. [28].

In computer networks, traffic measurement is the process of measuring the amount and type of traffic on a particular network, as shown in Figure 2.1. It allows network managers and analysts to make day-to-day decisions about operations and to plan for long-term developments.

It has gained interest over the years as an important network-engineering tool for networks of multiple sizes. Different applications (traditional ones such as web-based, malicious others such as worms and viruses or peer-to-peer) can affect the underlying infrastructure of the network [28].

Measurement strategies can be seen as an essential tool for identifying anomalous behavior (e.g., unexpectedly high traffic volumes, Denial of Service (DoS) and Distributed Denial of Service (DDoS) attacks, routing problems, among others), for the design and validation of new traffic models, for offering highly demanded services, as well as for activities such as upgrading network capacities. It should be noticed the difference between network measurement and identification: the former is about data gathering and processing while the latter is focused in the recognition and classification of some traffic characteristics (which could vary depending on the technique used) [28].

## 2.2 Analysis of Traffic Measurements

Traffic measurements can be classified in different ways depending on the approach taken, the type of content to be measured and the measurement tools used to perform the monitoring operations. Different techniques are more suitable to some tasks than others. Below, we show several categorizations for measurements and the related tools and a review of their use cases.

### 2.2.1 Hardware Measurements Tools

Hardware measurements are performed by hardware tools deployed in the networks. These tools are often referred as network traffic analyzers: they are special purpose equipment designed expressly for the collection and analysis of network data. Usually these equipments are expensive, depending on the number of network interfaces, types of network cards, storage capacity and analysis capabilities. Such products are often geared towards network operators for managing and troubleshooting networking issues.

In large scale networks, there is the possibility of bottleneck due to disk I/O intensive operations and memory bandwidth or operating systems call. Thus, hardware tools and their components, such as network adapter, memory/disk bandwidth and buffer management, might be required to overcome these limitations because they are highly

optimized to do only network monitoring and analysis jobs [30].

However, since the performance of personal computer and their peripherals such as CPU, memory and disk have been improving over the years, the convenience of hardware-based measurements tools have decreased in time [30].

### 2.2.2 Software Measurements Tools

Software-based measurements typically depend on tools that apply kernel-level modifications to network interfaces of commodity network cards to enhances them with packet capture capabilities. This approach is mostly inexpensive and provides greater functionality for measurement customizations and analysis.

Tools such as tcpdump [75], Wireshark [76] and ntop [71] are widely used in networking research. Using these tools, huge volumes of data can be analyzed in efficient and scalable ways. This offer possibilities to perform off-line traffic analysis, which can be useful to manage privacy requirements and for traffic characterization. Conversely, the capability to monitoring online traffic is more suited for time-sensitive applications such as intrusion detection or application identification [7].

### 2.2.3 Active Measurements

Active measurements are mainly used for fault and vulnerability detection and network or application performance tests. These techniques are not implemented everywhere due to several drawbacks listed below [28]:

- Cannot reveal network characteristics influenced by users due to the fact that active measurements send user behavior independent packets.
- May face scalability issues related to the size of the monitored network. A large network could make prohibitive to perform tests for end systems and to conduct experiments in order to gain insights about the behavior of the given network.

- Usually, it requires the generation of extra traffic to be able to execute the measurements.

### 2.2.4 Passive Measurements

Passive measurement techniques are performed by observing network traffic packets and flows. For an in-depth characterization of network traffic, passive measurements can be classified in two levels [28]:

- Packet-based measurements: Consist of capturing packets headers and analyzing them.
- Flow-based measurements: Deals with a summary of unidirectional streams of packets passing through a given router.

Passive measurements are particularly suitable for traffic engineering and planning because they show traffic dynamics and distribution. The main concern regarding the implementation of passive techniques is dealing with the massive amount of data, since it scales with link capacity. In other words, the volume of processed data can become very large on high-capacity link networks.

#### 2.2.4.1 Packet-based Measurements

Measurements are performed on each packet traveling across the measurement point. Thus, collection of the information can be very fine-grained, including source and destination IP address, source and destination port numbers, packet sizes, protocol numbers and specific application data.

There are several packet capture tools (sniffers) available that can be used to collect packet information:

- **TCPdump [75]** is a command-line packet analyzer that can print out the description of the contents of packets and make some statistical analysis out of the trace files.

- **Wireshark [76]** is a network protocol analyzer combined with a user-friendly GUI and includes many traffic signatures. It can be used for accurate, payload-based application identification.

- **SNORT [74]** A too for real-time traffic analysis and packet logging. It is capable of performing content search and match and of detecting many types of network attacks.

Packet-based measurements presents several issues for its implementation in a network system [28]:

- **Data scalability** The amount of require space for storing packet traces can be prohibitive. Usually, Database Management Systems (DBMS) are used for data storage.

- **Transmission speed** Suboptimal hardware (e.g., low-end network interface, low CPU power) will not be able to capture packet at full speed in the network. Therefore, some packets can be missed from the analysis.

#### 2.2.4.2 Flow-based Measurements

At a macroscopic level, measurements are performed on flows, which are unidirectional series of packets that are matched by existing aggregation rules in the system. Different types of data can be collected, such as number of flows per unit of time, flow bitrate, flow size and flow duration [28].

Common parameters that can be used to group the packets into flows are: source and destination IP address, source and destination port, protocol number. Different tools use different subsets of these parameters to provide their flow definition. Cisco NetFlow [68], a tool created to process flows in the network, provides a set of services for IP application, such as network traffic accounting, network planning, security, DoS monitoring capabilities and network monitoring, among others.

Flow measurement provides better scalability since it does not require specific information for each packet, just the accumulated information associated with the flow. In combination with wildcard rules, which enables a further generalization of the flow definition, flow-based techniques can perform better in high-speed network with large link capacity.

### 2.2.5 Data Reduction Techniques

For both packet and flow monitoring tools, the large amount of data generated by traffic in the network is a serious concern. As link capacity and number of flows passing through a router or switch grows, it becomes computationally intense to keep up the state information and the counters associated. Therefore, data reduction techniques are crucial for scalable network traffic measurements [28].

These techniques are usually carried out online in a single pass through the traffic stream to avoid buffering and reprocessing. Common methods in use are [32]:

- **Aggregation** Combination of several data into a unite, single component. It is commonly additive and it is used to provide compact summaries when the individual components can be discarded from the measurement.

- **Filtering** Selection of data based on specific values. Filtering is useful to reduce to a traffic subset of interest, once this subset has been identified.

- **Sampling** Involves the random or pseudo-random selection of data from the traffic stream.

The main difference between the methods listed above is that filtering and aggregation require knowledge of the features of interest in advance, whereas sampling allows for the retention of arbitrary and unbiased information while reducing data volume at the same time [32].

### 2.2.6 Goals on Traffic Analysis and Classification

Accurate traffic classification is essential to a variety of network activities [19] [20], including:

- **Identification of application usage and trends** Identifying correctly the user application and its popularity may generate valuable information for network operators to optimize traffic engineering and for providers to offer services based on user demand.

- **Identification of emerging applications** Accurate identification of new network applications can provide insights on the emergence of new trends that could alter the dynamics and performance of network traffic.

- **Anomaly detection** Anomaly diagnosis is fundamental to provide high security and service availability. Anomalies are unusual and may cause significant changes in network traffic, such as those produced by DoS attacks.

- **Accounting** For service providers, knowing the behavior and applications usage of their subscribers may be of interest for application-based accounting or for offering new products.

### 2.2.7 Traffic Measurement Errors

Measurement errors are caused by faults in equipment, constraints on equipment design or user-dependent errors. The list below provides a common classification of these types of errors:

- **Pre-processing errors** Are generated by computers when compiling data for operators.

- **Post-processing errors** Are generated by operators when analyzing data that has been measured.

- **Statistical errors** Are caused by the averages in traffic measurements and by the fact that measurements are made from discrete samples.
- **Database errors** Exist when errors are generated by faults in the storage of information.
- **Interpretation errors** Are the result of misinterpretation of data by analysts.

Thus, conducting a sound Internet measurement study is a difficult undertaking [22]. Analysis usually suffer from known limitations due to some of the errors stated above, including limited measurement duration or coverage, loss of information during the measure process and failure to identify the application correctly [28].

## 2.3 SDN

Software Defined Networking (SDN) provides a clear separation between the hardware infrastructure and the software capabilities in the network, as mentioned in Chapter 1.

### 2.3.1 Innovations

The separation of concern between hardware and software in SDN can be better analyzed around four main innovations:

#### 2.3.1.1 Separation of Control and Data Planes

Protocols in a network usually fall in one of the following three categories or planes: data, control and management. The data plane refers to all the messages generated by the users. In order to transmit correctly all this information, networks use messages to perform tasks, such as find a valid route and check connectivity. These kind of messages are called control messages and are essential for network operations. Traffic statistics and equipment state in the network belongs to the management plane.

One of the key additions of SDN is the separation between the control and data planes in a network. The data plane forwards the packets by using the forwarding tables generated by the control panel. The control logic is implemented in a separated controller that prepares these forwarding tables. Thus, the switches implement data plane logic that is greatly simplified [41].

#### 2.3.1.2 Centralization of the Control Panel

Data and control planes are usually distributed in the Internet. To prepare routing tables, multiple routers interact with each other and exchange reachability information with their neighbors and their neighbors' neighbors.

Centralization, out of the question just a few years ago, is used today to optimize the state of networks based on dynamic changes and allows faster propagation of the adjustments than with distributed protocols [41].

#### 2.3.1.3 Programmability of the Control Panel

With a central controller, it is easy to implement control changes on the network by modifying the programs that run in the control. With a suitable API, a variety of policies can be implemented and modified as the network system changes.

A programmable control plane is the most important contribution of SDN. It allows to build several virtual layers in the network that have different policies and goals and yet share the same hardware infrastructure. Dynamically changing policies would be, otherwise, very difficult and slow for a fully distributed system [41].

#### 2.3.1.4 Standard API

There are tree main APIs or interfaces in a SDN centralized controller, each one with specific requirements. Figure 2.2 shows how these APIs work together in a network.

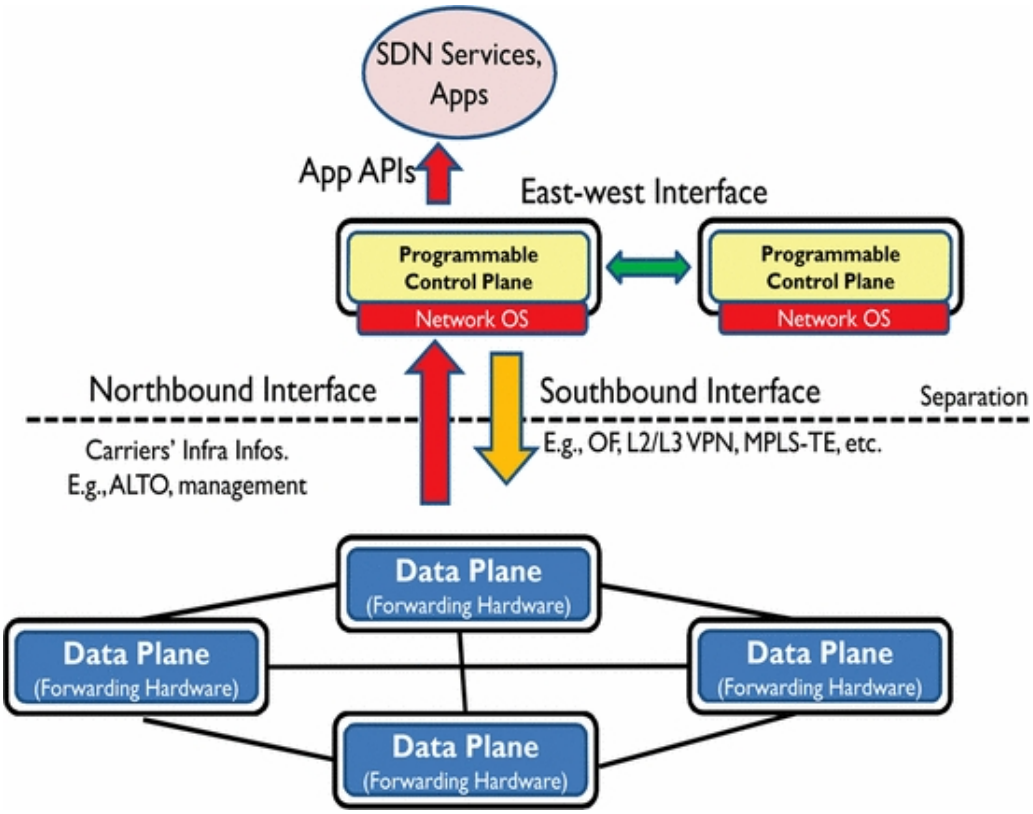



Figure 2.2: SDN architecture and APIs [40]

- **Southbound API** Provides a way to communicate between the programmable control plane and the data plane. Its main purposes are [40]:

  - Programmability and re-configurability: Should support flexibility in the control plane and make it possible to adopt new control schemes on networks. It should allow for easy and rapid creation of virtual networks and their dynamic reconfigurations.

  - Resource sharing: Should allow the abstraction of physical resources' characteristics so that controllers and/or applications can access those capabilities by using the APIs.

  - Traffic isolation: Should provide secure isolations among multiple virtual network, without lost of performance and security.

- Network abstraction: Abstractions for the information of physical network resources should be provided by the Southbound API. Support of lower and higher level interfaces for resource control allows to disengage the virtual networks from the complex characteristic of the network infrastructure.

- **Northbound API** An API between the network infrastructure and applications/services. Objectives of northbound API are mentioned as follows [40]:

  - Routing-related requirements: Should provide well-defining routing-related information from network infrastructure or controllers to services/applications (e.g., topology, discovery, traffic patterns, delay, Quality of Service (QoS), among others).

  - Management-related requirements: Should provide well-defining management-related information from network infrastructure or controllers to services/applications (e.g., energy use, monitoring, maintenance, among others).

  - Policy-related requirements: Should provide well-defined policy-related information from network infrastructure or controllers to services/applications (e.g., access control, security, among others).

- **East-west API** Provides a communication interface between SDN controllers. Amongst its objectives are: intra-domain and inter-domain communication, scalability, interoperability and deployability [40].

### 2.3.2 Use cases for SDN

Software defined networking has several use cases that can help improve the throughput and behavior of networks and applications in the following domains:

- **Cloud Computing** Cloud services have developed at a rapid pace over the last decade. However, these innovations have focused in server and data technologies as well as distribute applications. This has led to networks to become a major

hindrance for cloud operations due to the separation between the management for networks and servers. SDN is a viable way to achieve integration between the network and cloud framework, by providing standardized interfaces that can be used by SDN controllers and cloud orchestration frameworks [49].

- **Routing** The API between data plane forwarding and centralized control plane provides ample opportunities for routing protocol adaptations. This is very difficult to achieve in current decentralized routing schemes implemented on closed network element. Routing services can be realized by the SDN concept, by using programming modules on SDN controllers. Implementations for these services could provide traffic optimization, secure routing and path protection, among others [49].

- **Network Management** Today's network management policies tend to rely on configurations and decisions made by the network operators. This makes difficult to modify established policies, which lead to inefficient network operations. In order to change this, networks need to be able to adapt policies in a dynamic way based on a range of information. This calls for a more general specification of network policies that can be translated into specific rules using a policy engine. SDN controllers offer a very suitable way to provide this capacity to the network, as they have all the network information available in a central location [49].

### 2.3.3 Traffic Measurements on SDN

Traditional measurement techniques, such as active and passive methods can be implemented in SDN. Below, we discuss some of their main limitations:

- **Active measurements** Active techniques requires planning to handle the requirements of the SDN centralized architecture. The increment of the data throughput by several orders of magnitude can possibly lead to the saturation of the control mechanism [46]. In consequence, the data cannot provide the SDN controller with

the information in the required timeframes to minimize the impact of traffic disturbance [46].

- **Passive measurements** Passive techniques often depend on data reduction techniques, such as sampling and filtering, and rely on statistical approaches to infer the current state of the network traffic. Although, they are non-intrusive and with small-to-none additional traffic in the SDN, measurement inaccuracies can be introduced due to small flows being missed from the analysis or multiple monitoring nodes along a given SDN flow path sampling the same packet [42].

#### 2.3.3.1 Areas of Research and Development

Currently, newer approaches try to overcome the previous mentioned issues by leveraging the programmable interfaces offered by SDN to obtain fine-grained measurement of network flows. There are three main areas of research and development in this direction, as shown in Figure 2.3.

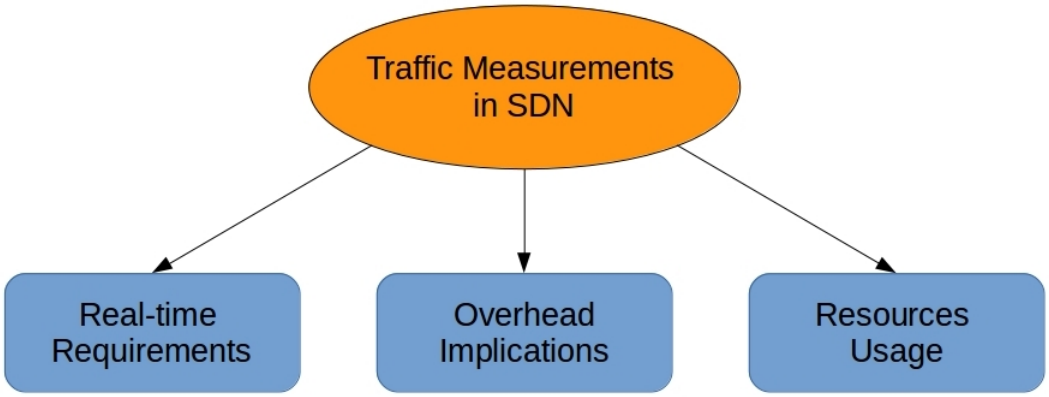



Figure 2.3: Traffic measurement areas in SDN, adapted from [62]

- **Real-time requirements** Traffic measurement in SDN relies heavily in the recollection of statistical data about flows in the network. These large amounts of detailed data may generate scalability issues for real-time analysis, especially if the mea-

sured information is for time sensitive applications [62].

- **Overhead implications** Monitoring of networks usually introduces some overhead, which needs to be taken into account as trade-off with traffic measurement accuracy [62].

- **Resources usage** Measurement techniques depends on specific resources to achieve high accuracy. Enabling dynamic changes in the distribution of these resources on the network can improve the general performance of these measurements.

## 2.4 Current State of the Art

Traffic measurements implemented in SDN-enabled networks play an essential role in the development and overall usage of such systems. Current networking systems are characterized by their large scale and the heterogeneity in the traffic they produce, which makes the process of predicting and measuring traffic a complex task. Current approaches try to address these complexities by relying in fast analytical models (e.g. machine learning techniques, Monte Carlo statistical methods and automation tools) to help converting data into informed decisions [62].

### 2.4.1 Real-time Requirements

Traffic measurements in SDN need to cope with large number of data available to be processed. Using different techniques and processes, several studies propose different alternatives to improve this situation.

PLANCK, proposed by [55], is a software-defined measurement architecture that uses the capabilities of port mirroring presents in many commodity switches. For 1 Gbps commodity switch, the time it takes to process data for measurement is in average from 280 microseconds to 7 milliseconds, while is in the range of 275 microseconds to 4 milliseconds for a 10 Gbps commodity switch connection. Although it is several times

faster that current schemes in traditional networks, traffic volume may exceed the port capacity, leading the switch to drop packets.

Using sampling-based SDN measurement methods, OpenSample [56] leverages sFlow [73] packets to support quasi real-time measurements of both the total network load and individual flows. OpenSample is a low-latency platform implemented on top of Floodlight OpenFlow controller that relies in TCP sequence numbers from packet header to accurately measure flow statistics. Due to its sampling approach, OpenSample manage to reduce to the control loop time to an average of 100 milliseconds rather than 1-5 seconds usually found in traditional polling-based approaches.

### 2.4.2 Overhead Implications

As traffic overhead growths due to increasing usage of measurements in network systems, distinct investigations have spent effort trying to find a balance between the accuracy of the measurements and overhead they generate.

In [35], it is analyzed the possibility of measuring large scale traffic aggregation in commodity switches and it is proposed a measurement framework that enables switches to match packets directly by using a small collection of wildcard rules in TCAM. This proposition reduces the overhead in the controller significantly because the switches are capable of making the decision of whether a processed packet match their own local rules without the need to forward it to the SDN controller. The framework is evaluated against HHH rules to measure and understand the trade-off between accuracy and overhead. Nevertheless, a major issue with this approach is the ability to add or update rules in the system.

Another proposal, OpenNetMon [50], focuses to determine whether end-to-end QoS parameters are met and delivers the input for Traffic Engineering (TE) to compute appropriate paths in OpenFlow networks. It is a pull-based measurement system where network traffic is monitored in a per-flow basis to produce data metrics between endpoints such as throughput, delay and packet loss. It fetches data from the switches by using an

adaptive strategy that increases when flow rates differ between samples and decreases when flows stabilize to minimize the number of queries it has to perform. This adaptive rate helps reduce the network and switch CPU overhead while optimizing measurements accuracy.

Similar to OpenNetMon, intelligent Traffic (de)Aggregation and Measurement Paradigm (iSTAMP) [53] leverages the means of OpenFlow to measure traffic parameters. It proposes the partition of TCAM entries of switches/routers into two parts as:

1. Aggregate part of incoming flows to generate aggregated measurements
2. De-aggregate and directly measure the most informative flow for per-flow measurements.

Among their propositions, there are the implementation of an optimal aggregation matrix to minimize the flow-size estimation error, an efficient-compressive flow aggregation matrix under resources constraints of limited TCAM sizes and an intelligent Multi-Armed Bandit based algorithm to adaptively sample the most significant flows. Then, iSTAMP jointly processes these measurements to estimate the size of all network flows using different optimization techniques.

Work done in [64] proposes several mechanisms to modify and augment the network representation used by the SDN controller to perform measurement analyses. These mechanisms extract traffic characteristics from network observations which are then used to derive performance metrics. Due to the bursty nature of network traffic and the adverse impact of this property on network performance, the authors propose an approach for extracting flow autocorrelations from switches counters and a random sampling approach that helps reduce the monitoring overhead in the network while enabling fine grained characterization of the flow autocorrelation structure.

The trade-off between measurement accuracy and traffic overhead is also studied in [45] and [48]. The research in [45] proposes a framework for collecting traffic information in hash-based switches, along with a HHH algorithm to select important traffic, to

support different measurement tasks. However, monitoring rules have to be deployed in the switches across the network. The work in [48] introduces a novel method that performs adaptive zooming in the aggregation of flows to be measured. To improve the balance between monitoring overhead and anomaly detection accuracy, it is proposed a prediction based algorithm that can dynamically change the measurement granularity along both the spatial and temporal dimensions.

### 2.4.3 Resources Usage

The implementation of traffic measurements in SDN environments usually needs to balance the effects between the usage of resources to process the measurements and their accuracy. Several studies have addressed this issue by leveraging the characteristics of the traffic and taking reactive approaches to changes in the traffic.

The work in [54] proposes Dynamic Resource Allocation for Software-defined Measurements (DREAM), a software that leverages the relationship between the user-specified level of accuracy and resource usage for measurement tasks in the system. In DREAM, resources are dynamically allocated for the measurement task to reach the desired level of accuracy based on traffic characteristics. The implementation has been tested with both a HH and a HHH metrics to show that DREAM can support more concurrent tasks with higher accuracy.

The research in [52] argues that current SDN applications, due to their proactivity, may require larger number of flow table entries, thus exceeding current TCAMs capabilities in the network. The work focuses on the implications of reactive installation of flow entries in the switches and shows the existence of a trade-off between the size of the flow table and the rate of dynamic installation of a missing or expired rule. Similar to DREAM, it is proposed a resource allocation strategy based on the current network load, the effective behavior of the flows, their granularity and their inter-packet arrival time. The evaluation of their approach shows that is a promising mechanism for improving TE flexibility with no additional requirements in terms of flow table size.

FlowMon, proposed by [65], combines the advantages of sampling and flow counting to propose a sample and fetch-based two stage large flow detection mechanism. It works by capturing first the suspicious large flows through coarse-grained sampling methods and then notifies the SDN controller to differentiate the true large flows by using measurement rules installed in selected switches. In order to optimize the associated TCAM resource allocation, it provides a dynamic flow entry assignment model. Experiments show that this proposal can help improve the large flow detection accuracy, decrease the TCAM resource consumption and balance the measurement load among switches in the network.

## 2.5 Related Work

Several studies have worked to improve some of the issues discussed in previous sections. Below, we provide an analysis of the studies we consider are the most important ones for this research, including DREAM and ProgME, which are used as the basis for the development of this research thesis.

### 2.5.1 Dynamic Resource Allocation for Software-defined Measurements (DREAM)

DREAM framework is a system designed to work with software-defined measurement for TCAM counters. Users of DREAM can dynamically instantiate multiple concurrent measurement tasks (such as heavy hitter or change detection) at an SDN controller, while specifying flow filters for the tasks at the same time. Since traffic for each tasks is very likely to be measured at multiple switches, DREAM needs to allocate resources at each required switch to maintain high levels of accuracy for its tasks [54].

In order to achieve this, DREAM leverages two observations:

- With more TCAM counters, tasks measurement becomes more accurate, but there is a point of diminishing returns: beyond certain accuracy threshold, more resources

account for less improvement in the level of accuracy.

- TCAM counters are only required at switches where there is traffic that matches the specified flow filter. Also, the number of resources depends on the traffic volume and its distribution.

Using these observations, DREAM supports more concurrent measurement tasks by matching the varying needs of the measurement tasks with TCAM resources. The framework can be extended to support different metrics than those originally implemented (Heavy Hitters (HH), Hierarchical Heavy Hitters (HHH) and Change Detection (CD)) and can support different primitive resources (e.g sketches) [54].

### 2.5.2 Programmable MEasurements (ProgME)

ProgME is an architecture design that can adapt to application requirements and traffic conditions in real time. It proposes a definition of flowset (arbitrary set of flows) as the basis for the traffic collection process. Instead of the per-flow counter approach, it uses a counter per flowset, which enables better scaling and resolution within a traffic profile. Furthermore, it includes a flowset composition language (FCL) by applying set theory operations (union, intersection and negation) over existing flowset. This means it can process broader measurement tasks than the management systems deployed in existing networks [37].

The main propositions of ProgME are outlined as follow:

- A versatile flowset definition as the base of network measurement. It includes the flowset composition language (FCL) for defining arbitrary set of flows and Binary Decision Diagram (BDD) as the underlying data structure for efficient operations (set and packet matching operations) between the flowsets.

- A Flowset Query Answering Engine (FQAE) to support user queries from the collected statistics.

- A multi-resolution tiling (MRT) algorithm, which can dynamically re-program the flowset measurement to zoom in on heavy hitters.

### 2.5.3 SCREAM: Sketch Resource Allocation for Software-defined Measurement

SCREAM is a system designed to work with hash-based counters, or sketches [47]. Sketches are summaries of data to provide an approximated answer to a specific set of queries. They can be implemented with SRAM memory, which is more cheaper and more power-efficient than alternatives such as TCAMs. Sketches can use sub-linear memory space to answer different measurement tasks such as heavy hitter [17], super-spreaders [18], large changes [13], flow-size distribution [15] and flow-size entropy [36].

The main challenges for sketch-based measurements are [61]:

- **Management of multiple instances of measurement tasks** Measurement tasks of different types and defined on different aggregates may be executed concurrently in the network or virtual network.

- **Efficient usage of memory counters** To achieve high accuracy, measurement tasks could required a large number of counters and the availability of these counters is bounded by resources such as the SRAM memory needed to save sketch counters, the control datapath inside switches and the control network bandwidth.

In order to solve the aforementioned challenges, SCREAM makes two main contributions [61]:

- **Sketch-based task implementation across multiple switches** Each task must gather sketch counters from multiple switches and prepare the measurement results to the end user. As sketches may need different sizes to provide accurate measurement, SCREAM implements novel techniques to merge sketches with different sizes from different switches.

- **Accuracy estimator** SCREAM introduces accuracy estimation without ground-truth or *knowledge a priori* of the traffic model and provides low estimation errors. It feeds these instantaneous accuracy estimates into a dynamic resource allocator [54] to support more accurate tasks by leveraging temporal and spatial statistical multiplexing.

### 2.5.4 Programmable Architecture for Scalable and Real-time Network Traffic Measurements

The work done in [27] presents a novel hardware-software co-designed solution that is programmable and adaptable to runtime situations offering high-throughputs that can match current link-speeds. The essential contribution in this design is the orthogonalization of memory lookups from traffic measurements through the query-driven measurement scheme.

To evaluate it, a prototype has been implemented that shows the scalability of this scheme against per-flow sampling-based solutions and a heavy hitter identification algorithm using the aforementioned query-driven scheme. The evaluation shows, as a result, better performance in both speed and throughput, even when used as an off-line solution.

## 2.6 Challenges

SDN has been a breakthrough in the networking industry in the recent years. Traffic measurements are a key enabler for potential benefits of the SDN paradigm through all the network systems. Nevertheless, there are several major challenges and research issues to be addressed. In Figure 2.4, we show an overview of some of the main challenges in the field.

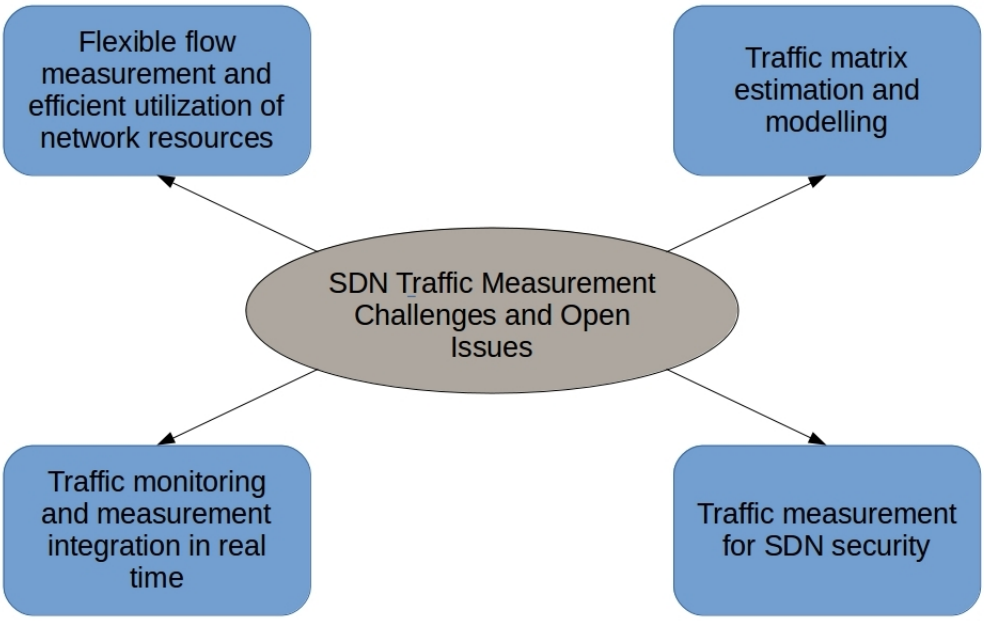



Figure 2.4: Challenges and Open Issues in SDN Traffic Measurements, adapted from [62]

### 2.6.1 Flexible Flow Measurement and Efficient Utilization of Network Resources

The main goal of traffic measurement in SDN is to yield a flexible flow measurement framework that can support analyses for different granularities to be capable of satisfying a variety of applications. Yet this effort is not trivial due to the fact that it requires the estimation of fine-grained volume of network flows with flexible configurations in interconnected and heterogeneous large scale systems [62].

Another target is to develop efficient solutions to handle big data in these large scale systems, in the contexts of user behavior, locality and time-dependent statistics, especially from mobile applications [63]. In such scenarios, different mechanisms are need to create flexible and dynamic measurement schemes that would help network operators gain insights of the system status and help maximize their resource utilization [62].

To address these requirements, research must be performed in sampling techniques

that allow pattern recognition in flows and their size estimation, efficient storage solutions and algorithmic approaches for flow accountability. In addition, dedicated APIs should be developed to measure and manage bandwidth from controllers [62].

### 2.6.2 Traffic Matrix Estimation and Modelling

Traffic Matrices (TMs) reflect the amount of traffic that runs between pair of sources and destination in a network. They play an important role in many network tasks, such as network design [16], traffic engineering [16], traffic accounting [8] and performance diagnosis [34].

Current network infrastructures are complex, large scale systems that provide connection among multiple domains. Therefore, it is very challenging to make direct measurements of TMs due to the hard constraints of network measurement resources [59].

In SDN contexts, although in their beginning stages, TMs may be used in ubiquitous fashion for optimal traffic engineering in SDN. Thus, it is a pressing need to find mechanisms to estimate and model a TM through mathematical and statistical methods, as well as to produce scalable algorithms to process high number of information from TMs [62].

### 2.6.3 Traffic Monitoring and Measurement Integration in Real Time

Real-time application and services are sensitive to the occurrences of delays and demand stringent QoS requirements to be able to adapt to network changes and dynamic resource allocation. For example, mobile applications require real-time traffic monitoring and measurement to cope with changes in network channels due to users' mobility [62].

To provide real time processing in a SDN environment, there are several issues that need to be addressed [62]:

- Scalability issues to integrate a large number of fine-grained measurement statistics

to a centralized controller in order to provide fast decision mechanisms for QoS policies.

- Synchronization of real-time traffic monitoring and flow analysis from different controllers to facilitate decision making capabilities.

- Better mechanisms for inference and statistical prediction methods in real time to compensate for limitations in the capacities of the back-end database located in the controller.

Thus, special focus would be required on provide APIs that may run on SDN controllers to do automatic inference of traffic data, context dependent traffic analysis, real time QoS, among others, for delay-sensitive applications.

### 2.6.4 Traffic Measurement for SDN Security

As the SDN paradigm becomes widely available in network infrastructures such as cloud computing centers, data centers, carrier network and other highly sensitive systems, potential vulnerabilities and security issues are a serious concern due to the fact that several security attacks can be conducted against SDN through different network components [58].

Table 2.1: Major security threats in SDN, adapted from [57]

| SDN Plane | Threat Type | Description |
|---|---|---|
| Application Plane | Lack of authentication & authorization | No compelling authentication & authorization mechanisms both for applications and third-party integrations tools. |
| | Fraudulent flow rules insertion | Malicious or compromised applications can generate false flow rules and it is difficult to check if an application is compromised. |
| | Lack of access control & accountability | Difficult to implement access control & accountability on third-party application and nested applications that consume network resources. |
| Control Plane | DoS attacks | Visible nature, centralized intelligence and limited available resources makes the control plane a target for DoS attacks. |
| | Unauthorized controller access | No compelling mechanisms for enforcing access control on applications |
| | Scalability & availability | Logic centralization in one place will face scalability and availability challenges as usage grows. |
| Data Plane | Fraudulent flow rules | Data plane is not logic-aware and hence more susceptible to fraudulent flow rules |
| | Flooding attacks | Flow tables of OpenFlow switches can only store a finite number of flow rules |
| | Controller hijacking or compromise | Data plane depends entirely on the control plane, which makes it dependent in the controller security |
| | TCP-Level attacks | Transport Layer Security (TLS) protocol can be affected by TCP-level attacks |
| | Man-in-the middle attacks | In the absence of TLS, man-in-the middle attacks can be carried out. |

SDN offers the possibility for security enhancements by means of a global visibility of the network state from the logical centralized control plane. Hence, the SDN architecture empowers networks to actively monitor traffic and diagnose threats to facilitate network forensics, security policy alteration and security service insertion [57]. However, the SDN separation in different planes and the centralization logic in the controller make this networking paradigm an appealing target for several types of attacks, as listed in Table 2.1 above, that would take control of operations and carry out malicious activities. By using traffic measurements mechanisms, there are several steps that can be implemented to alleviate these attacks occurrences [62]:

- Diagnose specific sources of events, security violations and attacks.
- Harvest data from nodes and analyze the information to match it with security policies in order to minimize the possibilities of misconfiguration that could grant access for attackers.
- Accurately identify and categorize traffic anomalies as well as isolate and trace back atypical signals within the data.

## 2.7 Summary

In this chapter, we have analyzed traffic measurements and their applications, mainly related to the SDN architecture. We provided a breakdown of different classifications for traffic monitoring, as well as a revision of the current state of the art and the main challenges that lie at the forefront of research.

Below, the following tables sum up the essential topics discussed in this chapter. First, Table 2.2 condenses the details of the different classifications of traffic measurements, mentioned in Section 2.2. Next, Table 2.3 gives a summary of the areas where research is in progress for traffic measurements in SDN, as seen previously in Section 2.3.3.1. Finally, Table 2.4 outlines the challenges in the field, shown in Section 2.6, and lists their main points.

Table 2.2: Analysis of traffic measurements

| Topic | Classification | Comments |
|---|---|---|
| Type of tools | Hardware-based | • Special equipment designed for collection and analysis of data<br>• Expensive and dependent on network interfaces, storage capacity and analysis capabilities<br>• May help avoid bottleneck due to I/O operations or bandwidth consumption |
| | Software-based | • Modifications on network interfaces to provide packet capture capabilities<br>• May implement custom analysis on top of existent tools<br>• Suitable for both online and off-line traffic analysis |
| Measurement approach | Active | • Applications in fault and vulnerability detection and performance tests<br>• Scalability issues for large networks<br>• Increase in traffic overhead |
| | Passive | • Focus on TE and planning<br>• Large volume of processed data to analyze<br>• Two main approaches: Packet-based and flow-based |
| Data reduction | Aggregation | Combining several data into a single component. Used in measurements to provide compact representation |
| | Filtering | Selection of traffic data based on specific values such as protocol, IP address among others |
| | Sampling | Random or pseudo-random selection of data from the stream. Usually used in scalability solutions |

Table 2.3: Development lines for traffic measurements in SDN

| Line | Description | Research |
|---|---|---|
| Real-time requirements | Recollect statistical data about network flows and combine it with scalable real-time analysis and time sensitive applications. | • PLANCK [55]<br>• OpenSample [56] |
| Overhead | Analyze overhead of measurement techniques and minimize its impact in networks. | • Jose et al. [35]<br>• OpenNetMon [50]<br>• iSTAMP [53]<br>• Z. Bozakov et al. [64]<br>• Moshref M. et al [45]<br>• Zhang, Ying [48] |
| Resource usage | Provide dynamic adaptability between measurement tasks demands and resources supplies to improve their performance. | • DREAM [54]<br>• M. Dusi et al. [52]<br>• FlowMon [65] |

Table 2.4: Challenges and Open Issues

| **Challenge** | **Comments** |
|---|---|
| Flexible flow measurement and efficient utilization of network resources | • Better sampling techniques<br>• Efficient storage and algorithms for flow accounting<br>• Specialized applications to control bandwidth |
| Traffic Matrix estimation and modelling | • Estimation and modelling of TM using mathematical and statistical approaches<br>• Scalable algorithms to process large number of TM data |
| Traffic monitoring and measurement integration in real time | • Synchronization of real-time traffic monitoring and traffic analysis from multiple controllers<br>• Inference-based and prediction-based statistical methods in real time<br>• Process automation for inference, measuring traffic data and real-time QoS |
| Traffic measurements for SDN security | • Analysis of traffic measurements to identify and diagnose anomalies<br>• Real-time anomalies detection, classification and analysis at the flow level<br>• Collection of traffic data and analysis of non-compliant packets with allowed protocols |

# Chapter 3

# Improving DREAM: Tasks Prediction

This chapter focuses on the analysis of the essential steps taken as part of this research to generate a solution for how to improve measurement accuracy for tasks in DREAM by using previous traffic data, which was stated in the introduction chapter. It breaks down the analysis as follows: first, it introduces the key components of the original DREAM proposal in Section 3.1, such as the task measurement accuracy, the counter allocation strategy, an adaptive step size search, the task object and the counter configuration. Section 3.2 comments on an optimization idea to improve the tasks accuracy and its main issues. The proposed solution is then discussed in Section 3.3 and Section 3.4 makes the final comments on DREAM and the proposed ideas.

## 3.1 Original Proposition

As it has been mentioned before, DREAM is an adaptive system that aims to improve the performance for concurrent traffic measurement tasks by dynamically allocating/deallocating resources (i.e TCAM counters) depending on the accuracy needs of the existing tasks, while ensuring better resources usage in the switches of the network.

The framework implements a collection of algorithms running on an SDN controller. Users can import tasks with a specified accuracy into the system and receive periodic reports with measurement results for their tasks. Using these results, users can then reconfigure network parameters, install defenses or increase network capacity [54]. A user can be either a network operator, or another software component that instantiates tasks and processes results.

The workflow of DREAM is shown in Figure 3.1 below, which illustrates external interface communications to DREAM and noticeable features of its inner-workings. The user creates a task with its correspondent parameters (step 1). Then DREAM accepts or rejects the task based on the availability of resources (step 2). Each accepted task receives from DREAM a default number of counters at one or more switches (step 3) and is associated to a *task object* that references the resource allocation algorithms run by DREAM for each task [54].

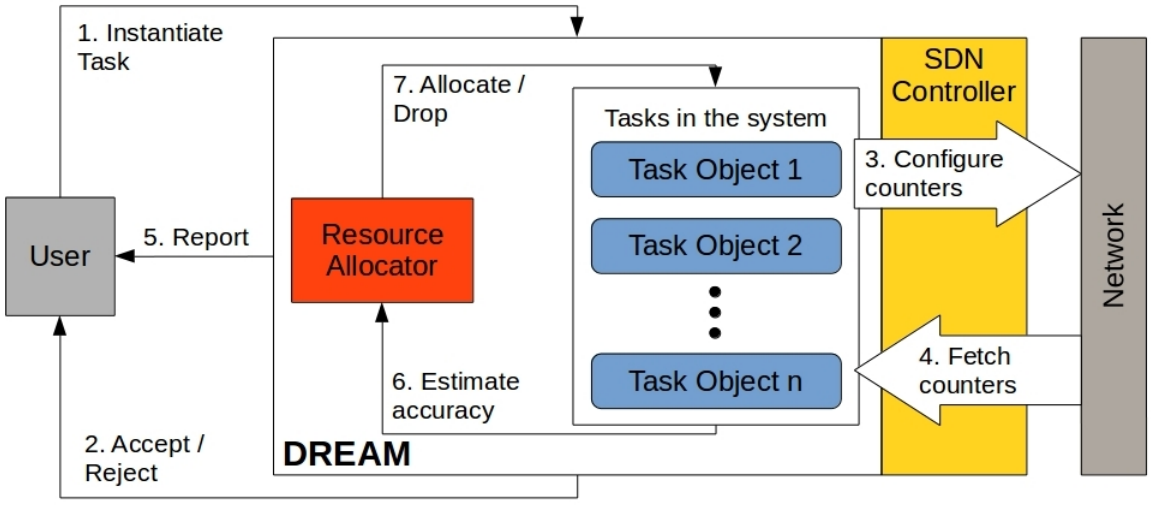



Figure 3.1: DREAM overview, adapted from [54]

Within the DREAM system, counters are periodically fetched from switches and passed on to task objects (step 4). These task objects then compute measurement results and report the results to users (step 5). At the same time, each task object measures the current task accuracy by using a *accuracy estimator*. These estimations are used by

the DREAM *resource allocator* to decide the amount of TCAM counters to allocate to each task and forward this information to the corresponding task object (step 7), who determines how to reconfigure these counters among the associated switches (step 5). Finally, if there is not enough available resources for a task, then DREAM drops the task, removes its associated task object and releases all the task's TCAM counters.

The DREAM implementation includes three types of measurement tasks:

- **Heavy Hitters (HH)** It is a traffic aggregate, which can be identified by a packet header field, that exceeds a specified volume. A common example is HH detection of source IPs that contributes to large amount of traffic in the network.

- **Hierarchical Heavy Hitters (HHH)** It is an extension of HH that detects longest prefixes that exceed a certain threshold even after discarding any HHH descendants already detected in the prefix trie. This technique is useful when dealing with hierarchical aggregates computed at different levels, such as the ones found in DDoS attacks [11].

- **Change Detection (CD)** Traffic anomalies in a network usually are related to important changes in traffic behavior. For example, large changes in traffic volume from source IPs are used to test for anomaly detection [54].

These *network-wide* measurements tasks have many applications in data centers and ISP networks. For example, they are used in multi-path routing analysis [31], network provisioning [6], threshold-based accounting [8], anomaly detection [48] and DDoS detection [24].

### 3.1.1 Task Measurement Accuracy

DREAM allocates additional resources to a task if its current accuracy is below the specified accuracy bound. However, due to the fact that tasks can see traffic in multiple

switches in the network, it is not straightforward to decide what measure of accuracy to use in a per switch-basis. The two possible measures for a task $i$ are:

- **Global accuracy ($g_i$)** The accuracy of the task $i$ in the network as a whole.
- **Local accuracy at switch $s$ ($l_{i,s}$)** The accuracy of the task $i$ in the local switch $s$.

Since both measures can be misleading, DREAM proposes the use of an *overall accuracy* $a_{i,s} = max(g_i, l_{i,s})$ to decide if it is necessary to make a resource allocation decision. To minimize the effect of the fluctuations of traffic change and estimation errors over the accuracy, DREAM applies a Exponentially Weighted Moving Average (EWMA) filter to smooth this overall accuracy [54].

### 3.1.2 Counter Allocation

DREAM uses a resource allocation algorithm to assign counters to each task and an admission control algorithm to decide whether to accept a new task into the system.

Counter allocation decisions are made when a task accuracy is below its target accuracy expectation. DREAM uses a step-wise search at the switch level that eventually converges to the desired target accuracy. This operation, as shown in Figure 3.1, redistributes counters from *rich tasks* (whose overall accuracy are above the accuracy bound) to *poor tasks* (whose overall accuracy are below the accuracy bound). DREAM makes this allocation decisions in *allocation epoch*, a time unit that spans multiple measurement epochs [54].

#### 3.1.2.1 Adaptive Step Size Search

The allocator does not known in advance the required number of TCAM counters for a task, denoted as $R_{i,s}(t)$, to reach its target accuracy. Since this $R_{i,s}(t)$ is unknown and varies over time, the allocator algorithm iteratively increases or decreases the associated number of counters for the task in steps depending on the overall accuracy $a_{i,s}$, until

the right amount of resources is allocated. DREAM estimates the $R_{i,s}$ by analyzing the changes in the accuracy status for the task (from *poor* to *rich* or the other way around) [54].

In order to ensure fast convergence, the system performs the allocation using an Adaptive Step Size with a multiplicative policy for both the increase and decrease steps (MM) [54].

#### 3.1.2.2 Resources Headroom

To avoid low task satisfaction, the system does not perform to the TCAM capacity limit. DREAM keeps a *headroom* of available TCAM counters (5% in the implementation) and will reject immediately a task if this spare capacity is below this target value on any switch for the task. This permits to avoid possible repercussions in tasks accuracy by fluctuations in resources usage and to ensure high task accuracy satisfaction.

Nevertheless, due to the variability of network traffic, DREAM may sometimes drop tasks when the headroom is insufficient. The system implements a *drop priority* for tasks such that poor tasks with low drop priority (i.e., tasks that should be dropped last) can get resources from those tasks with high drop priority (i.e., those that can be dropped first). When tasks with high drop priority remain poor for several consecutive epochs, DREAM terminates them from the system and releases its resources back into the network [54].

### 3.1.3 Task Objects

DREAM task object, as shown in Figure 3.1, runs a generic algorithm (Algorithm 1) on the SDN controller. Its fundamental component is the task-independent algorithm for configuring TCAM counters across multiple switches, and it does not depend on the specifications of each task type. This careful separation of responsibilities between task-independent and task-dependents parts enables an easier evolution of DREAM as a platform to perform measurement analysis.

**Algorithm 1** Generic task object implementation, cited from [54]

```
1: for measurement iteration do
2:     counters ← fetchCounters(task, switches)
3:     report ← createReport(counters)
4:     (global, locals) ← task.estimateAccuracy(report, counters)
5:     allocations ← allocator.getAllocations(global, locals)
6:     counters ← configureCounters(counters, allocations)
7:     saveCounters(counters, switches)
8: end for
```

In Algorithm 1, the task object performs six steps at each measurement interval. It fetches counters from switches (line 2), creates the current report for the task (line 3), estimates the task accuracy (line 4) and then calls the resource allocator (line 5). Using this information, the task updates its counters to match the new allocations and to improve its accuracy (line 6). Lastly, the task object saves the new counters configuration (line 7).

### 3.1.4 Counters Configuration

After the resource allocator has assigned TCAM counters to each task object on each switch, the tasks must decide how to configure these counter, which means they have to decide which traffic aggregates to monitor on which switches using these counters. Since the task objects cannot monitor every possible flow in the network due to the TCAM capabilities constraints, they have to resort to monitor traffic aggregates, thus provoking some trade-offs in the accuracy.

The main question becomes then choosing the best set of prefixes to monitor to have a sufficient accuracy measurement while complying with the limited resources.

DREAM proposes a *divide and merge* strategy to associate counters with prefixes to be monitored. It starts by measuring an initial prefix set as a prefix trie. Figure 3.2 shows an example prefix trie of a task and the application of the strategy after allocating more counters for the task. If a monitored prefix is deemed as *interesting* (gray nodes in the figure) from the perspective of the specific task (revealing a possible heavy hitter or

a change in traffic direction), then DREAM proceeds to divide the prefix to monitor is children and use more counters to monitor both children now. On the contrary, if some prefixes are *uninteresting*, DREAM merges them to free counters for more convenient measurements.

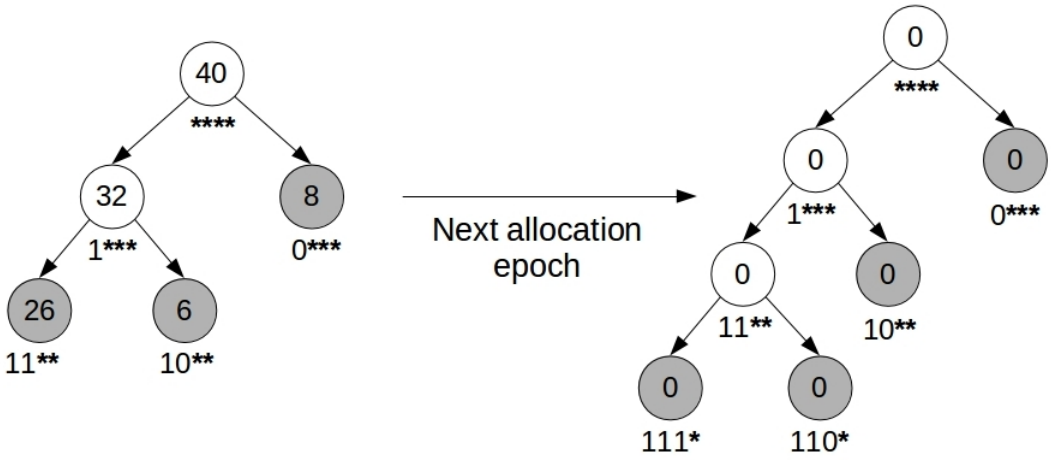



Figure 3.2: Result of *divide and merge* strategy for a prefix set with **threshold** 10. Gray nodes are the ones with corresponding TCAM counters, adapted from [54]

After each measurement epoch, DREAM updates the information for parents nodes in the prefix trie based on the measurements gathered during this epoch by the children with counters, and then proceeds to repeat the process of updating the prefix trie in the next allocation epoch, deciding whether to split prefixes and allocates counters or merges prefixes into their parent and free resources.

## 3.2 Convex Optimization for Resource Allocation

As we have shown in Section 3.1.2, DREAM allocates resources to tasks using an adaptive step-wise strategy to ensure high satisfaction by balancing the task accuracy status to converge to the required number of counter for the task.

One alternative approach would be to model this problem as a convex optimization that is applied periodically to maximize the number of satisfied tasks, while minimizing

the number of task that are dropped. To generate this optimization model, it is necessary to characterize first the relationship between counters and tasks accuracy *a priori* and generate ground truth measures to build upon the optimization model.

The main issue with characterizing the trade-off between counters and tasks accuracy is that it is hard to do in a real-time scenario, which would be beneficial for any optimization technique in terms of performance execution. Thus, the system would have to rely on some variation of off-line processing that can extract meaningful information from the traffic data to generate a mapping function between counters usage and tasks accuracy (that would serve as starting point for the optimization algorithm). To track these information in a compact way, techniques such as Traffic Matrix (TM) could be used to provide efficient estimation and ground truth of the tasks accuracy. Nevertheless, the delay created by dependency on an off-line processing strategy and the fact that improvements such as TM are still in the research stage diminish the possible gains to implement any optimization model.

Also, if there is an occurrence of traffic change, which could imply that tasks do not see traffic in the same switches anymore, the current configuration may no longer work, so it must be included in a way to reset the process and build again the relationship for the accuracy of the tasks and their dedicated resources.

### 3.2.1 Optimization Model

We model the allocation problem as an optimization problem as follows:

$$max \sum w_i x_i \tag{3.1}$$

$$w_i \geq \alpha_i$$
$$\sum rt_i = R$$
$$\sum r_{i,j} \leq rs_i$$
$$r_{i,j} \geq 0$$
$$rt_i \geq 0$$

where:

- $\vec{rt}$ is the vector of counters associated with each task.
- $\vec{rs}$ is the vector of counters capacity at each switch.
- $R$ is the total number of counters
- $r_{i,j}$ is the number of counters associated with task $i$ at switch $j$.
- $\vec{x}$ is the vector of the current tasks to be measured.
- $\vec{w}$ is the vector of the accuracy of each tasks and each $w_i$ could be defined as a function in the following domains:

$$w_i : (rt_i\ ,\ x_i) \rightarrow \mathbb{R}^+$$

  using the accuracy estimator already defined for each type of tasks in DREAM.

This optimization would rely on the use of actual accuracy estimation from DREAM and not ground truth accuracy, which is difficult to obtain in real-time traffic.

### 3.2.2 Variations

There are several variations that could be developed for this optimization [54]:

- Perform the optimization iteratively: jointly for all tasks across all switches optimize the increase/decrease of TCAM counters resources, measuring the results of these updates and repeat the process for as long there are tasks that have not been satisfied.

- Perform the optimization using a simulated annealing or neural net approach: The key would be to design an efficient way to predict and measure the *goodness* or distance between any two valid configurations.

For all these optimization approaches, common issues are [54]:

- The difficulty to scale to large numbers of switches and tasks because the complexity of the problem is in the order of the product of the number of switches and tasks.

- It may not even be possible to find a valid solution because these strategies could end in an infeasible optimization or an optimization that does not converge at all if the number of required resources for tasks exceeds the system capacity. Then, these techniques would have to resort to drop tasks after having admitted them.

- Even if a valid solution is found, at the moment of reconfiguring the counters on the system, it may not work anymore due to changes in traffic behavior and the optimization would need to start again.

## 3.3 Design and Implementation of the Proposed Solution

The proposed solution adopts a simpler idea and tries to reuse the existing features provided by the DREAM as much as possible. Using estimation algorithms, the implementation will predict the most likely packets that would match with the task filters in order to reach an acceptable accuracy faster than the original implementation.

In Section 3.3.1 below, we discuss the main components of the proposed solution and the main modifications made on top of DREAM framework.

### 3.3.1 Algorithm Overview

The solution performs several override steps to adapt DREAM framework for using the new functionalities and introduces new components in the workflow for calculating task accuracy. These components, mentioned below, are part of the life cycle of the updated task object once it is created during DREAM execution:

- **Preprocessing** To build the required data for the estimation algorithm by processing traffic information and generating the training data to feed the chosen estimation algorithm.

- **Estimation** It performs the analysis of processed traffic data in the measurement epoch to update the estimation algorithm and determines the next set of IP filters that should be measured.

- **Counters configuration** Using a user-defined percentage, the task object, aided by an *estimator adaptor* object, will divide the counters in two groups: the counters that DREAM configures following the original proposal and the ones that are going to be configured by the estimation algorithm.

All these components are connected to the estimation adaptor object, which it is initialized by the task object to perform estimation-related processes.

### 3.3.2 Estimator Adaptor

The estimator adaptor[1] acts as the bridge between the DREAM task objects and the estimation strategies implemented as part of the proposal. It is involved in each of

[1]In what follows, it is used the term adaptor to refer to the estimator adaptor object, unless specified otherwise.

the stages of the estimation process, and following the same design thinking proposed originally by DREAM, facilitates the decoupling of the estimation analysis, data update for the estimation algorithm and the counters configuration. Therefore, it makes easier to change estimation strategies and counter configuration for estimated filters which helps improve DREAM usability.

#### 3.3.2.1 Preprocessing

The preprocessing step happens at the moment of creation of a task object. Figure 3.3 below shows the workflow for this stage of the code execution.

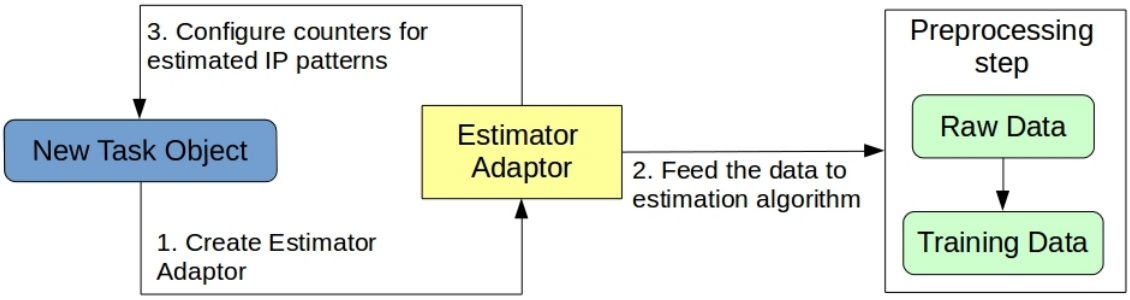



Figure 3.3: Task object creation process

This stage performs three essential steps during the task object creation. It builds an estimator adaptor object with specific information from the task[2] (step 1). The adaptor processes the training data (previous packets traces, e.g. CAIDA dataset) according to the specificities of the associated task (step 2). Finally, the adaptor sets for the task, the initial IP counters configuration according with a first run from the estimation algorithm[3] (step 3).

The following information is analyzed and kept to be used in the later steps for the estimation algorithm:

[2]namely the estimation strategy, the associated training data access, the task threshold and the task filter pattern

[3]usually only one counter at this first stage

- Packet information (2-tuples, 5-tuples or 6-tuples[4]) of those that match with the task filters.
- Number of occurrence of the same packet information over the entire training set.
- Specific information can also be stored depending on the type of the task (e.g. heavy hitter detection tasks can keep the average size of the packets).

#### 3.3.2.2 Estimation

Task estimation is the process of predicting which filter patterns can be of importance for the task (which in turn will make the task more accurate by improving the traffic aggregates measured by the counters during the measurement epochs).

The estimation workflow is split into different phases within the DREAM execution, as shown in Figure 3.4 below.

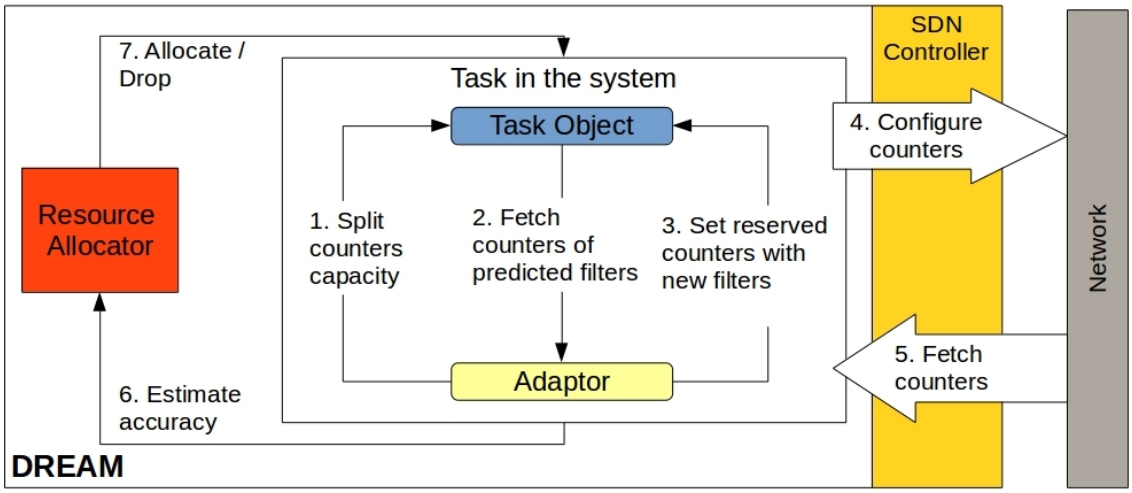



Figure 3.4: Estimation step in DREAM workflow

The main differences compared to the original DREAM workflow in Figure 3.1 can be summarized as follows:

[4]Tuples with more elements provide more information for the algorithm to work with

- **Step 1** At the end of the current measurement epoch, when a new counters allocation is performed by DREAM, the task object fetches counters from the switches and the adaptor, using the *estimation share* parameter, reserves a certain numbers of counter from the total available to be associated with estimated filters.

- **Step 2** Afterwards, the adaptor fetches the associated counters for the estimated filters and updates the algorithm and the task object with the score information from these counters. After the update, it removes these counters from the set of counters in use by the task object.

- **Step 3** Once the task object has finished to set up the counter configuration for its IP filters, the adaptor, using now the updated data from the previous iteration, proceeds to set the reserved counters with the corresponding predicted IP filters obtained from the calculations of the estimation algorithm.

#### 3.3.2.3 Counters Configuration

During the resource allocation stage, the adaptor reserves a certain number of counters to map them later to estimated IP address generated by the estimation algorithm. Once DREAM finishes to set up its counters configuration using its *divide and merge* strategy mentioned before in Section 3.1.4, the adaptor then uses the estimation algorithm to generate suitable filters to associate for these counters. An example of the combine procedure of DREAM and the adaptor counter mapping is shown in Figure 3.5.

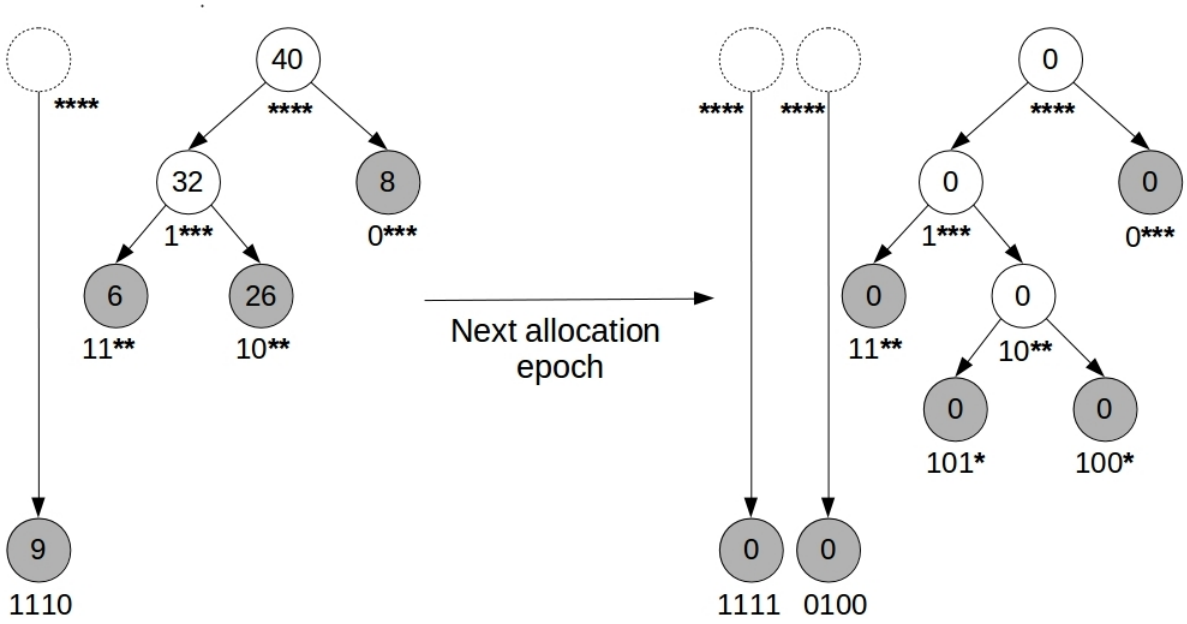



Figure 3.5: Combined counter configuration with **threshold** 10

At each new allocation epoch, the adaptor first updates the corresponding parent prefixes in the trie with the information gathered from its associated counters and then proceeds to remove these estimated counters from mapping. Afterwards, it asks the estimation algorithm to generate the appropriate number of filters to associate with its available counters for the next epoch.

As seen in Figure 3.5 above, from one allocation epoch to the next, there is an increment of one in the counters capacity for the adaptor and the DREAM strategy respectively. As a result, DREAM strategy splits down in the node most interesting in the prefix trie, as explained for the original strategy in Section 3.1.4, while the adaptor sets the counter to a new IP to be measured based on the calculations of the estimation algorithm.

### 3.3.3 Estimation Algorithm

The estimation algorithm is used by the adaptor to predict the best IP filters to associate with counters. The adaptor initially trains the algorithm with a predefined training data

that serves as the base for the initial estimations. At each epoch, the algorithm is updated by the adaptor with new information from the current traffic, which in turn allows to tune the estimation to make better predictions.

Below, we provide an overview of the estimation algorithms as part of the proposal.

#### 3.3.3.1 EWMA

This strategy works by correlating the average weight of a given IP within the data to the likelihood that it is going to appear again in the traffic. To avoid oscillations due to high fluctuations in packets weights the average is smoothed using an EWMA filter [33] with history weight *ewmaAlpha* in the form of:

$$mean = ewmaAlpha * mean + (1 - ewmaAlpha) * weight \tag{3.2}$$

with $ewmaAlpha = 0.4$

#### 3.3.3.2 Polynomial Curve Fitting

Curve fitting [3] constructs a function (*curve*) that has the best fit for a series of data points. To apply this strategy, we model the fitting function for each of the known IPs to the algorithm.

$$f : (w_i, s_i) \rightarrow w \tag{3.3}$$

where $ip_i$ represents the current IP to be fitted, $s_i$ the epoch in which it was analyzed and $w_i$ the measured weight for IP in this epoch.

For each known IP, the algorithm builds a corresponding curve fitting function by using the pair (weight, step) as data points. Then at each iteration, it generates a fitting weight for the IP based on the previous information, and the larger ones are returned by the algorithm.

#### 3.3.3.3 K-means++ Cluster

K-means is a technique, part of the family of cluster analysis, that aims to partition $n$ observations into $k$, with $k <= n$ clusters where each observation belongs to a specific cluster with the nearest mean, which serves as the cluster representation. K-mean++ [25] is an improvement over K-means that provides a lower bound for the quality of the cluster solution with respect the optimal k-means solution.

For the algorithm, the tuple $(ip, step, weight)$ is defined as an observation point. To maintain the algorithm execution in a reasonable time, the number of iterations to find the clusters is set to 10 and Euclidean distance is used as the distance measure.

At each iteration, the algorithm builds the cluster and then returns as predicted filters, the IP field from each cluster head. As opposite to the two previous estimation strategies, this technique allows to generate estimated IP filters that have not been seen in the network.

#### 3.3.3.4 Pseudo-Linear Extrapolation

Extrapolation [1] is the process of estimating, beyond the available observation range, the value of a certain variable given its relationship with other ones. For this specific application, it is desired to predict a valid IP using previous data (namely IP, step occurrence and weight).

The strategy implements a variation of linear extrapolation using three data points and averaging the slope of the interpolated line to provide the IP guess. Thus, it assumes it can generate good estimations by predicting the relationship between the tuples as an approximately linear function. This pseudo-linear approach can be expressed as follows:

$$(\sum_{k=1}^{n} y_k + (t - x_{k-1}) * (y_k - y_{k-1})/(x_k - x_{k-1}))/n \tag{3.4}$$

where

- $y$ represents the IP field,

- $x$ represents the weight for the corresponding IP,
- $t$ is the threshold value for the task, and
- $n$ is a configurable value representing the neighbors number to be used for the extrapolation (by default set to 3)

Similar to the cluster estimation technique, the pseudo-linear implementation allows for the generation of IP filters without previous data evidence in the traffic.

### 3.3.4 Modified Generic Task Object Algorithm

With the proposed solution, DREAM task objects, supported by its associated adaptor object, run a modified version of Algorithm 1. First, a build stage (only run at the moment of the task object creation) is added to start the estimator adaptor, where it processes the training data in accordance with the specificities of the estimation strategy and the task and set the initial counters configuration, as shown in Algorithm 2. Then, several modifications are applied to Algorithm 1 that processes the incoming packets, to update the estimation data and improve the prediction capabilities, which in return should improve the task accuracy, as shown in Algorithm 3.

**Algorithm 2** Building stage for task object

```
1: for each task do
2:     adaptor.processTrainingData(estimationAlg, task)
3:     counters ← fetchCounters(task, switches)
4:     allocations ← allocator.getInitialAllocations
5:     (dreamAlloc, adaptorAlloc) ← adaptor.splitsAllocations(taskObject, allocations)
6:     dreamCounters ← configureCounters(counters, dreamAlloc)
7:     adaptorCounters ← adaptor.configureCounters(counters, adaptorAlloc)
8:     saveCounters(switches, dreamCounters ⋃ adaptorCounters)
9: end for
```

For each new admitted task in Algorithm 2 above, the adaptor first processes the training data to generate suitable aggregates for the estimation algorithm (line 2). Then

it gets the mapping of counters for the task in the switches (line 3). In the next step, the task object obtains the counter allocations for task[5] (line 4), followed by the splitting step performed by the adaptor to associate some counters with estimated IP filters and the remaining ones with the usual DREAM strategy (line 5). (Line 6) and (line 7) perform the counters configuration according to each object strategy. Finally, the task object installs the new counters in the system (line 8).

**Algorithm 3** Estimation and update

```
1: for measurement iteration do
2:     counters ← fetchCounters(task, switches)
3:     report ← createReport(counters)
4:     (global, locals) ← task.estimateAccuracy(report, counters)
5:     allocations ← allocator.getAlloc(global, locals)
6:     adaptorCounters ← adaptor.getCounters(counters)
7:     (dAlloc, aAlloc) ← adaptor.splitAlloc(taskObject, allocations)
8:     adaptor.updateStats(report, adaptorCounters)
9:     adaptor.updateStats(global, locals, report, counters)
10:    adaptor.removeCounters(taskObject, adaptorCounters)
11:    adaptor.updateTask(taskObject, adaptorCounters)
12:    dreamCounters ← configCounters(counters, dAlloc)
13:    adaptorCounters ← adaptor.configCounters(counters, aAlloc)
14:    saveCounters(switches, dreamCounters ⋃ adaptorCounters)
15: end for
```

Algorithm 3 shows the new generic implementation derived from Algorithm 1. The first four steps are the same as in the original algorithm: the task object fetches the counters (line 2), creates the report for the counter (line 3), calculates the accuracy for task (line 4) and with the accuracy results, it gets the counters allocation for the next iteration (line 5). Next, the adaptor selects the counters that were used for estimation configuration (line 6). Then, the adaptor object proceeds to split the counters mapping to reserve a subset to be used for its own estimated configuration (line 7). In (line 8) and (line 9), the adaptor updates the estimation algorithm with the results from its specific counters as well as the general results from the task object accuracy measurement.

[5]originally in DREAM it was one counter, but for tasks that use estimations is two counters

Afterwards, it removes its counters from the mapping (line 10) and updates the task object with the score information from them (line 11). Then, the task object and the adaptor proceed to generate the counter configurations for the next iteration (line 12, 13). Finally, the task object saves the new counters configurations in the system (line 14).

## 3.4 Summary

In this chapter, we have discussed the main components of DREAM framework, possible alternatives to improve resource allocation for tasks in the system and how to implement estimation techniques on it. From the original framework, we have analyzed the task measurement accuracy processes, the counters allocations and its associated adaptive step size search and the generic task object implementation, including how the counters configurations works. We analyzed a possible optimization alternative for resource allocation and accuracy measurement, including several variations to consider and their fundamental issues. For the designed solution, we described key elements such as the supporting estimator adaptor, the implemented estimation strategies and the modified task object algorithm. The estimator adaptor bridges the estimation techniques usage with the DREAM system, processes the modified counters configuration and builds the appropriate training data to be used by the estimation techniques. We implemented four different estimation techniques: EWMA Estimation, Polynomial Curve Fitting, KMeans++ Cluster and Pseudo-Linear Extrapolation. KMeans++ Cluster and Pseudo-Linear Extrapolation are capable of predicting unseen IPs to be measured while EWMA Smoothed Average and Polynomial Curve Fitting are restricted to estimate IPs already analyzed from the training data or from the processed data during the task execution in DREAM.

# Chapter 4

# ProgME Integration on DREAM

As seen in Chapter 3, DREAM framework provides a way to dynamically assign resources in the network for tasks while achieving a high accuracy rate.

For the tasks definitions, the system provides a way to define filters to match the IP address of interest for the tasks. Using IP prefixes in a prefix trie, as shown in Figure 4.1, DREAM defines the set of interesting IPs that each task should patrol over time. These sets configuration can change dynamically by splitting down nodes in the trie (balancing the trade-off between resources availability and accuracy) to search for narrower IP subsets, as mentioned previously in Section 3.1.

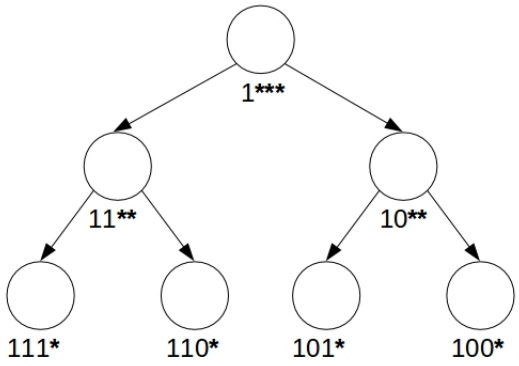



Figure 4.1: Prefix trie of source IPs with initial prefix **1*****, adapted from [54]

Although this technique provides some flexibility to specify IP address of interest,

it has the limitation that all monitored IPs must share the same prefix in order to be matched by the task. One possible alternative could be, within the current implementation constraints, to generate multiple tasks each with different IP prefixes that represent interesting sets to be measured.

Below, we discuss another approach that will allow for a more generic and flexible specification of pattern filters in DREAM, using the results from the ProgME framework [37]. With ProgME, it will be feasible to define different IP addresses to be filtered, without relying on the creation of multiple tasks at once or the common prefix set constraint either. The proposed implementation tries to follow closely the guidelines set by the work done in [37], while ensuring a successful integration with the components of DREAM framework.

The analysis is structured as follows. First, Section 4.1 presents the Flowset Composition Language (FCL), including its associated main concepts, flowset examples and the grammar. Section 4.2 clarifies how to provide a canonical flowset representation using BDD. Then, Section 4.3 introduces a flowset engine that enables the collection and reporting of statistics. The main ideas of the implementation on top of DREAM are detailed in Section 4.4, and Section 4.5 summarizes the analysis of ProgME and the implemented solution.

## 4.1 Flowset Composition Language (FCL)

FCL is the proposition of ProgME to enable the specification of arbitrary flowsets as a single entity. This allows the users to specify their requirements on aggregated traffic statistics and makes it easy for measurement tools to pre-process user requirements [37].

### 4.1.1 Concepts

To facilitate the discussion of FCL, below we provide a highlight of the main concepts that serve as the basis of ProgME architecture and are used in this project.

- **Flow** Refers to a set of traffic packets that share the same n-tuple definition in the header field. The most common tuple definitions are:

  - 5-tuple: <prt, sip, spt, dip, dpt>
  - 2-tuple: <sip, dip>

  *prt* is the protocol field, *sip* and *dip* are the source and destination IP address in Classless Inter-Domain Routing (CIDR) format and *spt* and *dpt* are the source and destination port, respectively. Other header fields, such as Type of Service (ToS), could be used as well.

  A flow is often used as the unit for traffic measurement and with a n-tuple definition, it can be considered as a point in the n-dimensional space, using each header field as a particular dimension.

- **Flowset** A set of arbitrary flows. It is not limited by structure and can take any shape, even being segmented in the space (n-dimensional space mentioned above). A single flow can be considered a special case of flowset containing only one member (the flow itself).

- **Flowset Grammar** Since a flow is a set itself, a grammar of set operations can be defined that enables the generation of arbitrary new flowsets by the combination of previous existent flowsets.

  The grammar defined in Table 4.1 provides several standards set operators such as *intersection* ($\bigcap$), *union* ($\bigcup$), *absolute complement* ($\neg$) and *relative complement* ($\setminus$). These operations are sufficient to build a flowset with arbitrary set of flows. Furthermore, more complex operation, such as NAND or NOR can be built using the provided operators. All the laws associated with set algebra, including *commutative*, *associative*, *distributive*, *identity* and *complement* laws, apply to flowsets as well [37].

Table 4.1: Grammar of Flowset Composition Language [37]

$$\begin{array}{lcc} e & = & e\ op\ e \mid (e) \mid \neg(e) \mid pr \\ op & := & \bigcap \mid \bigcup \mid \setminus \\ pr & := & < prt, sip, spt, dip, dpt > \end{array}$$

### 4.1.2 Flowset Examples

Table 4.2 presents two examples of flowset definitions that could be of interest for network administrators. Flowset $F_1$ represents all flows from private IP address and flowset $F_2$ matches with flows from incoming FTP traffic (port 21/22).

**Note**: In this example, * stands for do not care about the values in those tuple fields.

Table 4.2: Flowsets Examples [37]

| $F_1$ : Traffic from private IP | | |
|---|---|---|
| $F_1$ | = | $r_1 \bigcup r_2 \bigcup r_3$, where |
| $r_1$ | = | $< *, 10./8, *, *, * >$ |
| $r_2$ | = | $< *, 172.16./12, *, *, * >$ |
| $r_3$ | = | $< *, 192.168./16, *, *, * >$ |

| $F_2$ : FTP not from 10.1./16 | | |
|---|---|---|
| $F_2$ | = | $(r_4 \bigcup r_5) \bigcap r_6$, where |
| $r_4$ | = | $< *, *, *, *, 20 >$ |
| $r_5$ | = | $< *, *, *, *, 21 >$ |
| $r_6$ | = | $< *, 10.1./16, *, *, * >$ |

## 4.2 Underlying Data Structure: Binary Decision Diagram (BDD)

The string representation of flowset is not an optimal form to perform complex set operations. Following the idea suggested in [5] to encode firewall rules and access lists, ProgME proposes to use BDD [2] as the underlying data structure for flowsets [37].

BDD is an efficient data structure that is widely used in formal verification and simplification of digital circuits. A BDD is a directed acyclic graph that can provide a canonical representation of a set of boolean expressions. Every bit of the binary representation of the IPs, ports and protocol in the packet header is corresponded with a variable within the BDD structure.

Performing set operations such as *intersection*, *union*, *not* and *implication* is done by leveraging the structure and functions defined by BDD implementations.

The main details for using BDD as the underlying data structure for flowsets are mentioned below:

- The number of BDD variables for the implementation (using a 5-tuple flow) is 104 (8 bits protocol, 2x32 bits source and destination IPs and 2x16 bits source and destination port).
- To determine if a packet matches a flowset, the relevant bits are extracted from the header and used to generate a BDD. Afterwards, the logical implication ( $\Longrightarrow$ ) is used to determine if the packet belongs to the flowset.
- The number of nodes used to describe a 5-tuple flow is bounded by the number of variables (104). It can be less than that since BDD ignores the unused variables (masked bits in IPs address).
- The upper-bound of any flowset depends on the total number of nodes used to define each flowset, although it can be smaller due to the fact that BDDs tend to keep canonical forms for its boolean expressions.

## 4.3 Flowset Query Answering Engine (FQAE)

For any measurement task is essential to facilitate answering user queries about the characteristics of traffic aggregates. These aggregation can convey information of different granularities. Queries can ask for specific details (e.g. FTP traffic to host) or more general information (e.g. to know the ingress-egress of data in the network).

As mentioned in Chapter 2, current network systems process large volumes of flows, which makes harder to keep per-flow traffic information and produces scalability issues. By making the observation that the potential number of user queries can be far smaller

than the number of flows, one alternative could be to keep aggregated state information that concern to the queries [37], thus avoiding expensive per-flow information. For this, ProgME proposes the FQAE as a tool capable of answering any user query on traffic aggregates while keeping a low number of required counters. It contains two fundamental blocks:

- Measurement engine that collects per-flowset statistics.
- Program engine that processes a list of user queries as input and decides what to measure. It assumes that the queries are written as flowsets, as the examples in Table 4.2.

### 4.3.1 Disentangle User Queries

Since a packet could match potentially several different user queries, it can be inefficient to match it against all queries one by one if the number of queries is large. To address the issue, a mechanism, shown below in Algorithm 4, to disentangle the user queries into disjoint sub-queries such that each packet needs to be matched with exactly one sub-query is provided. Therefore, it is only necessary to find the matching sub-query and increment its counter. To achieve this, ProgME proposes to use a hash table mechanism (called *HashReduce*) to reduce the number of comparisons, similar to previous work done in [10].

**Algorithm 4** Disentangle of User Queries (adapted from [37])

**Input:** List of Queries Q ($|Q| = n > 0$)
**Output:** List of disjoint flowsets D

```
1:  D.append(𝕌)
2:  for x in Q do
3:      for p in D do
4:          if x <> p then                                  // Identical
5:              break
6:          else if x ⊂ p then                              // Subset
7:              D.append(p\x)
8:              D.replace(p, x)
9:              break
10:         else if x ⊃ p then                              // Superset
11:             x ← x\p
12:         else if p⋂x ≠ ∅ then                            // Overlap
13:             D.append(p\x)
14:             D.replace(p, x⋂p)
15:             x ← x\p
16:         else                                            // Disjoint
17:             continue
18:         end if
19:     end for
20: end for
21: return D
```

Algorithm 4 above generates a disjoint sub-queries $D$ set from an input list of user queries $Q$ represented as flowsets. It adds each flowset in $Q$ to $D$ after performing a sequence of comparisons with the existing flowsets in $D$. First, it initializes the result $D$ set with the universal ($\mathbb{U}$) set (line 1), to ensure that the final flowset sequence is a partition of the universe. Since flowset are inherently a set representation, any pair of flowsets must satisfy one of the following relationships: identical (line 4), subset (line 6), superset (line 10), overlap (line 12) and disjoint (line 16). Therefore, applying set operations, the input flowsets $Q$ can be transformed into a disjoint set of flowsets.

### 4.3.2 Matching Candidates

After the disentangle process applied to the user queries, every incoming packet is guaranteed to match exactly with one flowset representing a sub-query. A naive approach would be to compare the packet against each flowset in sequence. This is not an efficient solution when the number of flowset in $D$ is large [37].

As part of the FQAE implementation, it is introduced a hash table mechanism (*HashReduce*) to lower the number of comparisons needed to find the matching flowset. Based on similar work from [10], *HashReduce* implements a hash function that extracts several bits from the packet header fields. For each possibly hash value H, it generates then a correspondent BDD $H_{bdd}$, that represents a flowset with all the flows that match with that particular value in the header. Finally, the Table of Matching Candidates (TMC) is built by finding all flowsets from $D$ that have non-empty intersection with the $H_{bdd}$ [37].

The key point to analyze with *HashReduce* is the trade-off between memory consumption and lookup speed. Using more bits from header fields in the hash function increase the memory overhead, but can reduce the number of comparison with candidates in the table entries.

### 4.3.3 Statistics and Reporting

Collecting traffic statistics is a two-steps process for this implementation [37].

1. At the moment of receiving a packet, FQAE lookups the TMC to find the list of matching candidates by applying the same hash function to extract the bits from the incoming packet header.

2. FQAE compares the packet sequentially with the list of candidates until a matching flowset is found and its counter incremented.

During the measurement process, FQAE performs a traffic optimization of the TMC

candidates by sorting them based on the number of matched packets seen before (*Traffic-Sort*) [37]. This optimization is only possible due to the fact that all sub-query flowsets are fully disjoint. On the contrary, if some flowsets overlap with each other, finding the best ordering becomes a NP-complete problem [21], [23], and the solution would have to be fetched with some heuristics approaches.

To answer user queries, it is required then to aggregate the collected statistics from each related sub-query. The fundamental advantage here, in contrast to per-flow statistics, is that the number of sub-queries generated from the original user queries is usually far smaller than the number of flows in the traffic [37].

## 4.4 Design and Implementation

The implementation adapts the ProgME architecture to the specificities of DREAM framework.

Below, we discuss the main components of the implementation and the essential modifications performed to enable the integration within DREAM of flowset tasks, i.e, tasks that use flowsets to define the IP filters that are going to be mapped to counters.

### 4.4.1 Algorithm Overview

The ProgME solution performs several override steps to facilitate the accommodation of the new functionalities.

Two main concerns need to be addressed to guarantee the successful utilization of flowset filters in DREAM:

- **Counter capacity** Due to the nature of DREAM behavior and its adaptive step size, at any given point the flowset task may find that does not have the required amount of allocated counters to deploy each of the disjoint flowsets, generated in the Disentangle Algorithm 4, in the switches. Thus, it becomes necessary to develop a strategy to overcome these situations.

- **Task accuracy** Since each counter associated with a flowset always matches corresponding packets in traffic data successfully, usual accuracy metrics as recall and precision return 1, which clash with the DREAM allocation strategy based on the task accuracy.

#### 4.4.1.1 Counter capacity and configuration

When facing with a deficit in the counters capacity, the flowset task must decide which flowsets to merge in order to comply with the new allocation restrictions. Using the FQAE, it performs a *TrafficSort* optimization and the least valuable flowsets are selected to be merged, as shown below in the Algorithm 5.

**Algorithm 5** Counter capacity resolution

```
1: sortedFlowsets ← FQAE.trafficSort()
2: (toAlloc, toMerge) ← FQAE.splitFlowsets(sortedFlowsets, allocations)
3: mergedFlowset ← FQAE.merge(toMerge)
4: return FQAE.map(toAlloc + mergedFlowset, counters)
```

Algorithm 5 works as follows. First, it sorts the disjoint flowsets (line 1) by applying a traffic-aware strategy based on previously number of observed packets[1]. (Line 2) splits the flowsets, using the ordering in the previous line, in those that are going to be deployed the counters (*toAlloc*, which amounts to **capacity - 1**) and the remaining ones to be merged together (*toMerge*). It proceeds to merge the flowsets into a new one, using the union as the merge operation (line 3). Finally, the FQAE maps the counters to the flowsets (line 4).

#### 4.4.1.2 Task accuracy

As stated above, common accuracy techniques such *recall* and *precision* are not suitable to be used with flowsets, due to the fact that every packet is going to be matched by

[1] for equal number of observed values, it compares the flowsets weights to decide

its corresponding flowsets. Thus, the values for the recall and precision would be always 100%.

Since the task accuracy, real or estimated, it is necessary for DREAM to be able to allocate the desired number of counter, the flowset task uses the following measurement as accuracy.

$$\frac{\#\ of\ allocated\ flowsets}{total\ \#\ of\ disjoint\ flowsets} \tag{4.1}$$

Thus, if the number of allocated counters is smaller than the number of disjoint flowsets the accuracy is going to be the result of the previous equation, otherwise is going to be 1.

### 4.4.2 Flowsets Task Object Algorithm

As part of the implementation, the flowset task object runs a modified version of the Algorithm 1, as it is shown below in Algorithm 6.

**Algorithm 6** Flowset task object implementation

```
1: for measurement iteration do
2:     counters ← fetchCounters(switches)
3:     report ← createReport(counters)
4:     (global, locals) ← FQAE.estimateAccuracy(report, counters)
5:     allocations ← allocator.getAllocations(global, locals)
6:     counters ← FQAE.configureCounters(counters, allocations)
7:     saveCounters(counters, switches)
8: end for
```

Algorithm 6 shows the implemented modifications. The first 2 steps remains the same from the original DREAM algorithm: the task object fetches the counters (line 2) and creates the report for the counters (line 3). Aided by the FQAE, the flowset task object estimate the task accuracy, using the previously discussed strategy (line 4). Using the accuracy results, it gets the counter allocation for the next iteration (line 5). Then,

FQAE configures the counters, as stated previously in Section 4.4.1.1 (line 6). Finally, the task object saves the counter in the network switches (line 7).

## 4.5 Summary

In this chapter, we have examined the main ideas behind the ProgME architecture and the proposed solution to implement it on top of the DREAM framework. From the original ProgME proposal, the key concepts are detailed such as Flowset definition and grammar, the use of BDDs as the underlying data structure to support set operations among flowsets and the utilization of the FQAE as the central mechanism to perform flowset operation and answer flowset statistics. For the proposed solution, we have analyzed the main issues to overcome to integrate the Flowset architecture in DREAM. Key elements of the implementation have also been detailed such as the counter configuration with capacity constraints, an accuracy measurement design and a modified version of the task object adapted to perform flowset tasks operations.

# Chapter 5

# Evaluation and Discussion

This chapter analyzes the implemented features on top of DREAM, namely the estimation architecture for tasks and the implementation of programmable metrics, from ProgME. We have performed a series of simulations to evaluate the feasibility and limitations of the different alternatives and measure their performance against the original DREAM approach, including a parameter sensitivity analysis for the implemented components.

The analysis is split in two main areas. First, Section 5.1 analyzes the evaluation of the DREAM original strategy against the new estimation strategies implemented. Then, Section 5.2 proceeds to analyze the implementation of the ProgME in DREAM and shows an application use case of flowsets in DREAM. Finally, Section 5.3 sums up the results of the evaluations of implemented solutions.

## 5.1 DREAM Evaluation

We have implemented a prototype solution of the estimation algorithms for allocation on top of DREAM framework and use it to compare with the original DREAM allocation strategy.

Section 5.1.1 states the methodology followed to perform the different tests with the

prototype, including the different estimation parameter settings, network settings and evaluation metrics. Then, Section 5.1.2 splits the analysis of the results for switches with smaller number of TCAM counters and switches with larger numbers, and Section 5.1.3 provides some considerations about the behavior of the estimation algorithms with different parameters values.

### 5.1.1 Methodology

We have followed similar methodology conventions to evaluate our solution as the ones used by the original work [54] to facilitate the results for comparison. We use the same task objects previously defined in [54] and similar network configurations and evaluation metrics.

#### 5.1.1.1 Implementation

Our solution has been implemented on top of the existing DREAM framework. The main contributions are centered around the estimator adaptor, which links the existing task objects with estimation algorithms. Since the estimator adaptor is a component within the task object, it can be used in Java Floodlight controller [70] with both OpenFlow switches and Open vSwitch [72]. The total implementation is nearly 2,000 lines of code.

#### 5.1.1.2 Estimation Parameter Settings

We configure the estimator adaptor with an estimation share of 5% of the task allocated capacity. The sensitivity of task objects using estimations in DREAM to this parameter is explored later in Section 5.1.3.

#### 5.1.1.3 Tasks

Our workload consists of the three tasks types implemented by the original DREAM framework, HH, HHH and CD, executed both individually and in combination. During

the evaluation, we use a similar parameters configuration as stated by the original proposal [54]: the default accuracy bound is set to 80%, the local and global accuracy are smoothed using EWMA with history weight of $\alpha = 0.4$, and the default threshold for the tasks is 10Mb. The default drop priority is to drop the most recent task first. In Section 5.1.3, we also explore the variation in performance with different accuracy bounds.

#### 5.1.1.4 Network Settings

Tasks run on average 5 minutes on the system. To evaluate the implementation, our tests include: middle-scale scenarios with 256 tasks that arrive following a Poisson process during 20 minutes and have traffic from 8 switches and large-scale scenarios, where 4096 tasks arrive in the system during 80 minutes having traffic from 8 out of 32 switches in the network. To simulate network traffic, we use a 5-hour CAIDA packet traces 2016 [66] from a 10Gbps backbone link of a Tier-1 ISP. We split the trace in 5-min chunks with 16 /4 prefixes, of which those with $> 1\%$ of total traffic are only used.

#### 5.1.1.5 Estimation Strategies

Tasks in the workload can be associated with an estimation strategy, either EWMA, Polynomial Curve Fitting, K-means++ Cluster or Pseudo-Linear Extrapolation. The estimation share for counters is set by default to 5% of the task object available counters, and the estimator adaptor defines a 100 measurement epochs interval window to limit the recorded training data for each of the estimation strategies.

#### 5.1.1.6 Evaluation Metrics

We evaluate the implemented estimation strategies and the DREAM original procedure following the analysis of three main metrics.

- **Satisfaction ratio** The percentage of time the task accuracy is above the accuracy bound.

- **Drop ratio** Measures the percentage of tasks that are dropped while being active in the system.

- **Rejection ratio** Measures the percentage of tasks that were never accepted into the system.

For the satisfaction metric, we also show the $5^{th}$ percentile, which captures the tail accuracy behavior: a $5^{th}$ percentile of 10% means that 95% of the tasks were satisfied for 10% or more of their lifetime. Related to the drop ratio measure, we also show the average duration of tasks before being dropped by the system.

### 5.1.2 Simulation Results

Our simulations allow us to study the performance of our solution and to compare the estimation strategies with the original DREAM solution. In the following sections, we analyze the results for the middle-scale and large-scale scenarios that run workloads with task of type HH, HH and CD separately, as well as combined workloads that run a mixture of these tasks. These analyses demonstrate that estimation techniques are a feasible and scalable alternative and, in general, have a superior performance than the original DREAM approach.

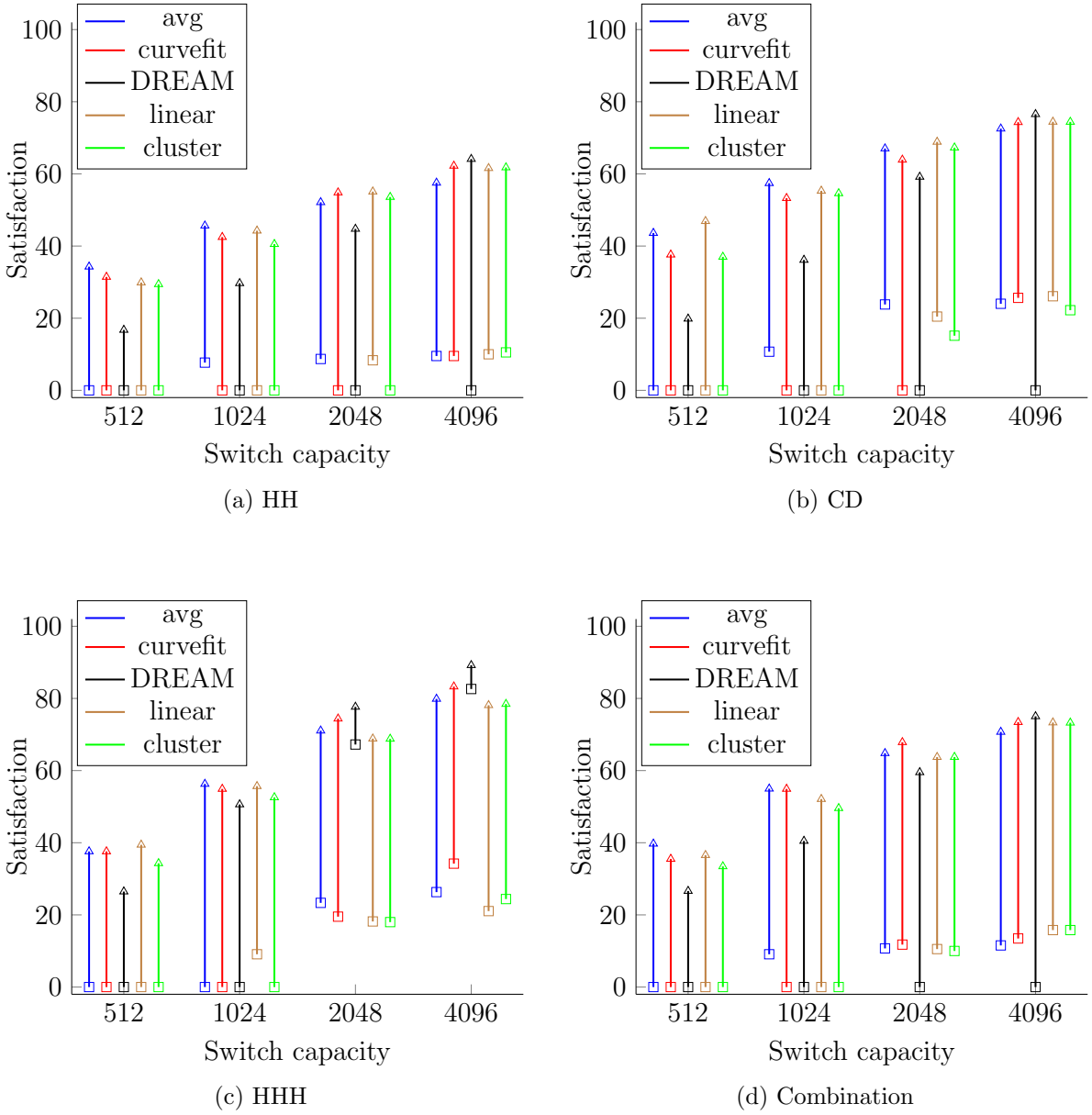
#### 5.1.2.1 Middle-scale Scenario

Figure 5.1 shows for different switch capacities the average satisfaction (*upper end*) and $5^{th}$ percentile (*lower end*) of tasks for HH, CD, HHH and Combination workloads, using either one of the estimation techniques, such as EWMA Smoothed Average (*avg*), Polynomial Curve Fitting (*curvefit*), K-means++ Cluster (*cluster*) or Pseudo-Linear Extrapolation (*linear*), or the default DREAM implementation. Figures 5.2, 5.3, 5.4 and 5.5 show the averages for drop, rejection and duration before drop ratios for those same workloads. We stacked in bar charts the drop (*lower darker color section*) and rejection (*higher lighter color section*) ratios for each of the strategies, and the numbers at the top

of each bar reflect the percentage of tasks that were either dropped or rejected by the system.

**Highly resource-constrained switches**. For smaller switches (those where the workload overloads the switches resources), our estimation strategies maintain higher average task satisfaction and $5^{th}$ percentile while at the same time rejecting fewer tasks from the systems. For example, in Figure 5.1a, for switches with 512 or 1024 counters, the *avg* strategy has better average task satisfaction (around 35% for the 512 and 45% for the 1024) than DREAM (around 17% for the 512 and 30% for the 1024), while it has a $5^{th}$ percentile close to 10% for the 1024 case. For the ratios analysis, in Figure 5.2, DREAM has the higher total of tasks either dropped or rejected for switches with 512 and 1024 counters (around 76% for 512 and 61% for 1024 while *avg* has around 69% and 55% respectively) and although it drops fewer tasks than the estimation techniques, DREAM tasks have a higher duration running in the system before being dropped, thus keeping resources in use for tasks that are going to be discarded.

These results are similar across techniques in different tasks workloads for the cases of switches with 512 or 1024 counters with the exception of the HHH workload. In the HHH workload, DREAM has better ratios for dropping and rejecting task and similar ratio for the running duration than the estimation techniques with switches with 512 and 1024 counters. Although the estimation techniques have better average satisfaction, the difference with DREAM is smaller. We believe that this is related to how the estimation techniques focus on predicting complete IP patterns in the prefix trie, see Section 3.3, which increases the possibility to lose track of some intermediate nodes that could be of interest as possible HHHs.




(a) HH (b) CD

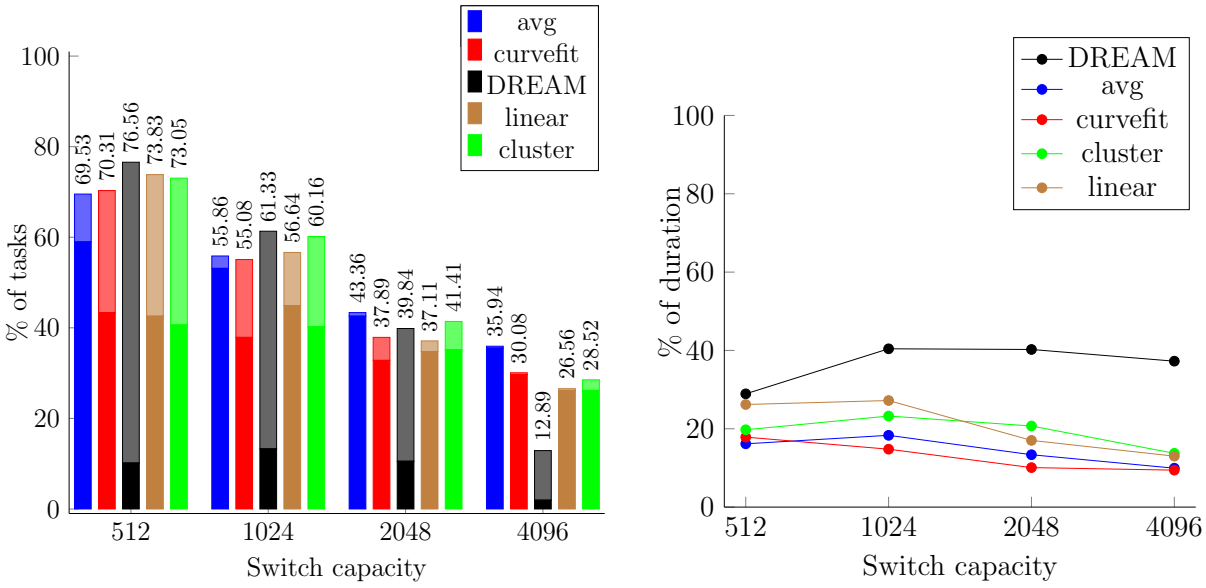
(c) HHH (d) Combination

Figure 5.1: Satisfaction and $5^{th}$ percentile metric for tasks workload

**Large capacity switches**. For large switches, the results are closer between DREAM and the estimation techniques. The estimation strategies have better $5^{th}$ percentile (between a 10% and 20% better on average) for switches with 2048 and 4096 counters. For example, in Figure 5.1a, the $5^{th}$ percentile for the *linear* strategy in the HH workload is around 10% while it goes over 25% in the CD workload, as shown in Figure 5.1b. For

these same scenarios, the $5^{th}$ percentile of DREAM remains at 0. Across both switch capacities, estimation techniques have smaller duration on average for tasks that are going to be dropped by the system, as shown in Figures 5.2f and 5.3b. For switches with 2048 counters, estimation strategies have also better average satisfaction (consistently between 5% and 10% better) than DREAM, see Figures 5.1a, 5.1b, but the percentage of tasks either dropped or rejected are very close between them with DREAM performing better in some cases and the estimation techniques in others. For switches with 4096 counters, DREAM has better average satisfaction and the lower number of tasks not finished across workloads, which we believe is influenced by its evenly analysis of the state of the counter prefix trie to find the best solutions and the fact that it has enough room to match those solutions, while the estimation techniques may be introducing random noise with a greater share of reserved counters that could diminish their contribution to the tasks accuracy.

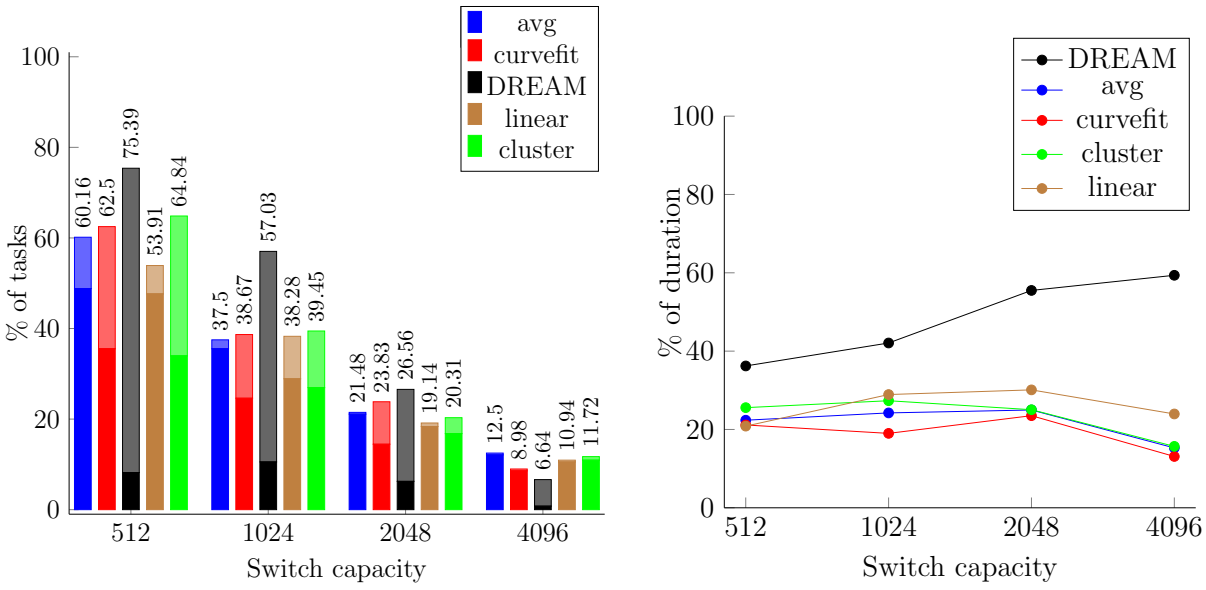
Once again, the HHH workload is the exception. In this scenario, DREAM outperforms estimation techniques in nearly every metric. It has better average satisfaction (around 89% versus 83% of *curvefit*, which is the second best) and a far higher $5^{th}$ percentile (around 82% against to 35% of *curvefit*). The rejection and drop numbers go to global lows of 3.91% and 0.39% for switches with 2048 and 4096 counters respectively, while the average duration drops to almost zero for switches with 4096 counters.




(e) Drop and Rejection Ratios

(f) Duration Before Drop Ratio

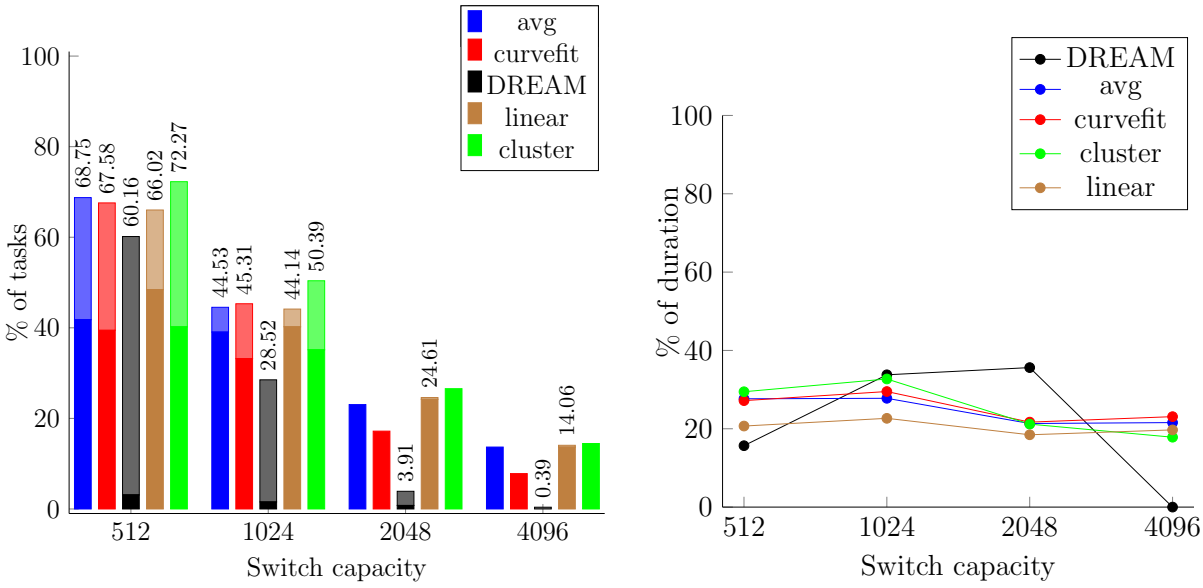
Figure 5.2: Drop ratio, Rejection ratio and Duration before Drop ratio for HH workload




(a) Drop and Rejection Ratios

(b) Duration Before Drop Ratio

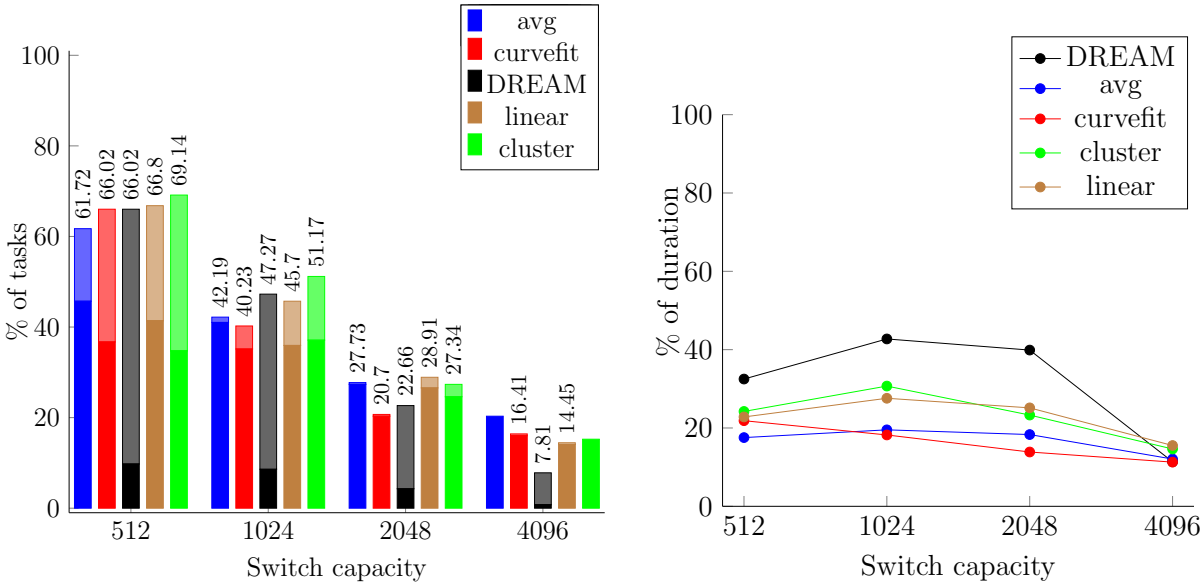
Figure 5.3: Drop ratio, Rejection ratio and Duration before Drop ratio for CD workload




(a) Drop and Rejection Ratios

(b) Duration Before Drop Ratio

Figure 5.4: Drop ratio, Rejection ratio and Duration before Drop ratio for HHH workload



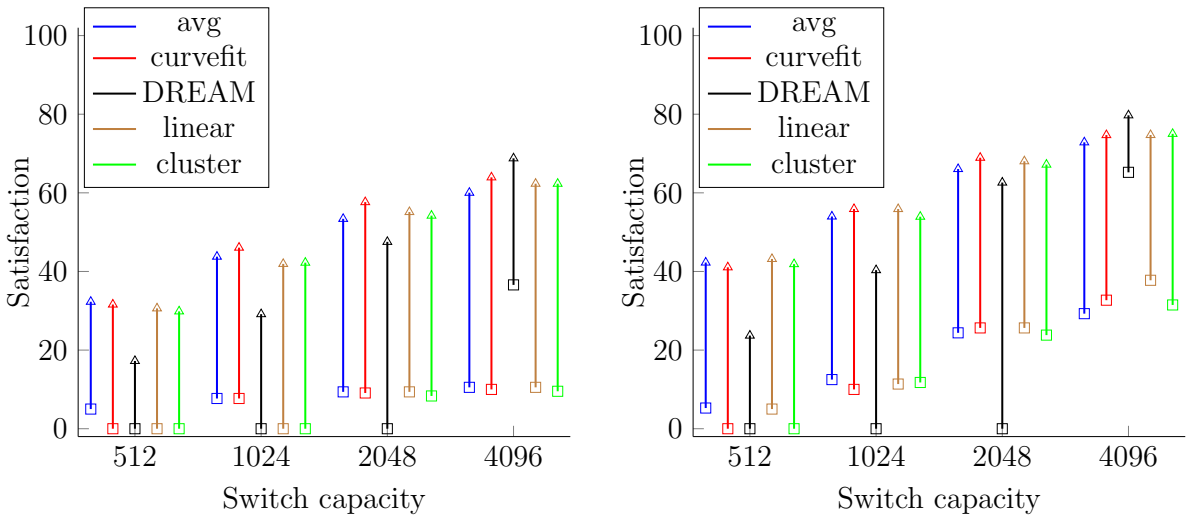

(a) Drop and Rejection Ratios

(b) Duration Before Drop Ratio

Figure 5.5: Drop ratio, Rejection ratio and Duration before Drop ratio for Combination workload

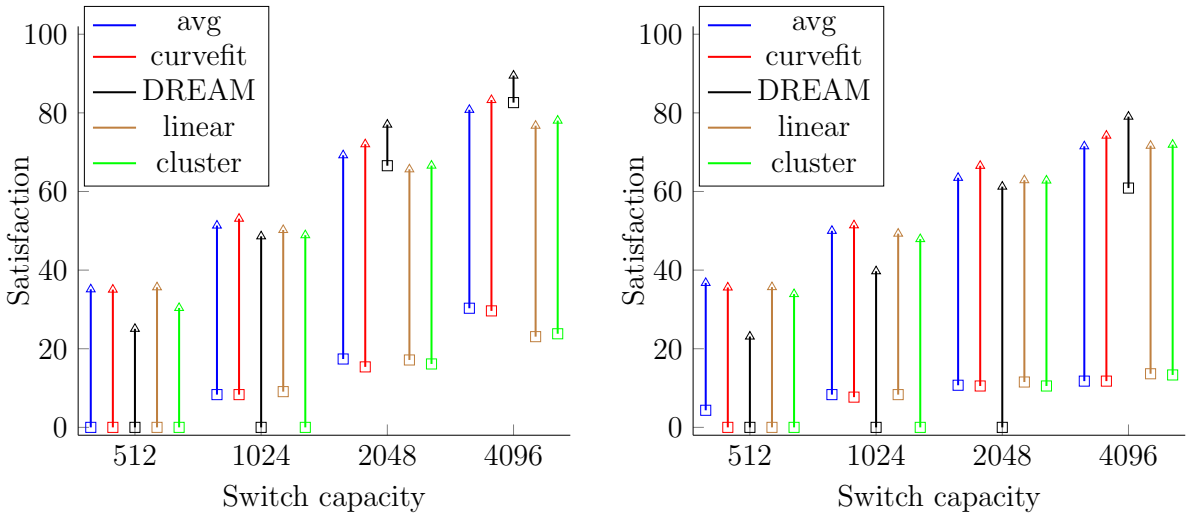
#### 5.1.2.2 Large-scale Scenario

The results in the large-scale simulations are similarly consistent with those previously obtained in the middle-scale scenarios. Figure 5.6 compares the average satisfaction of the different workloads, which show that estimation strategies perform better for resource-constrained switches while DREAM have better average satisfaction and $5^{th}$ percentile for switches with 4096 counters. Figures 5.7, 5.8, 5.9 and 5.10 show the behavior of drop, rejection and average duration ratios, which confirm the trends seen in the middle scenarios: estimation strategies outperform DREAM for highly resource-constrained switches while DREAM has the clear advantage for switches with capacity of 4096 counters and the average duration before drop is higher for DREAM than for estimation techniques. Once again, DREAM has the clear advantage in the HHH workload.



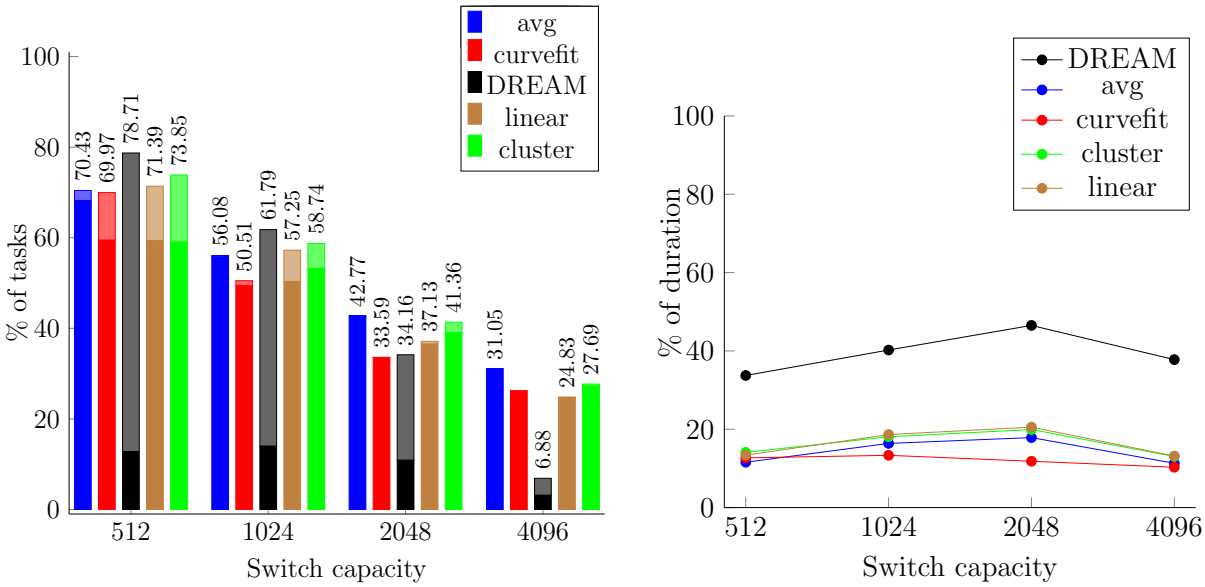

(a) HH

(b) CD



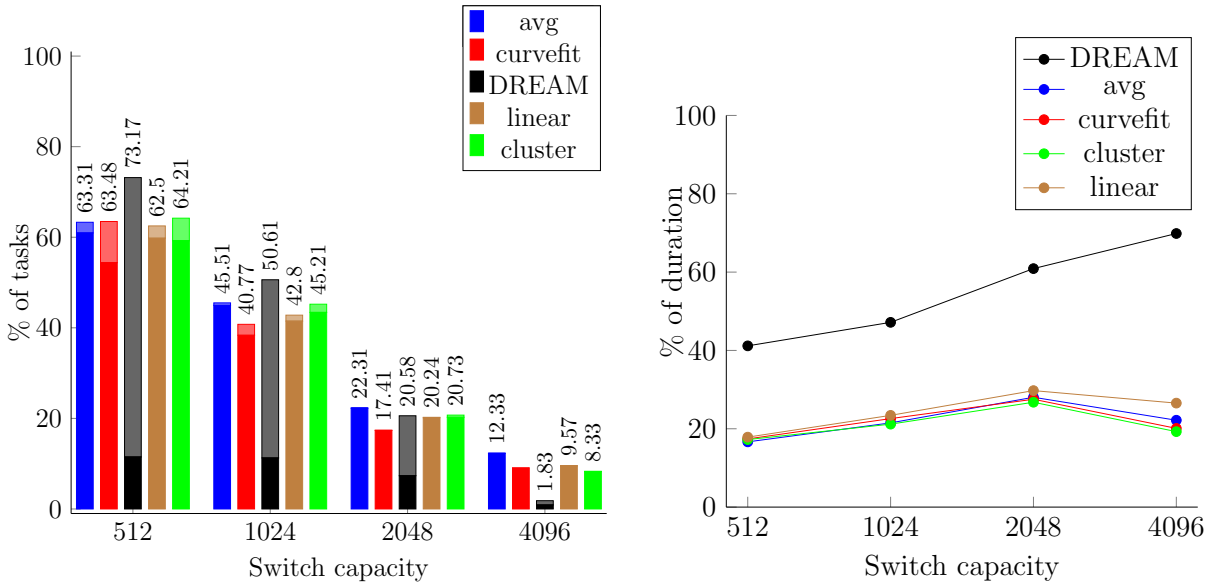

(c) HHH (d) Combination

Figure 5.6: Satisfaction and 5-th percentile metric for tasks workload




(e) Drop and Rejection Ratios (f) Duration Before Drop Ratio

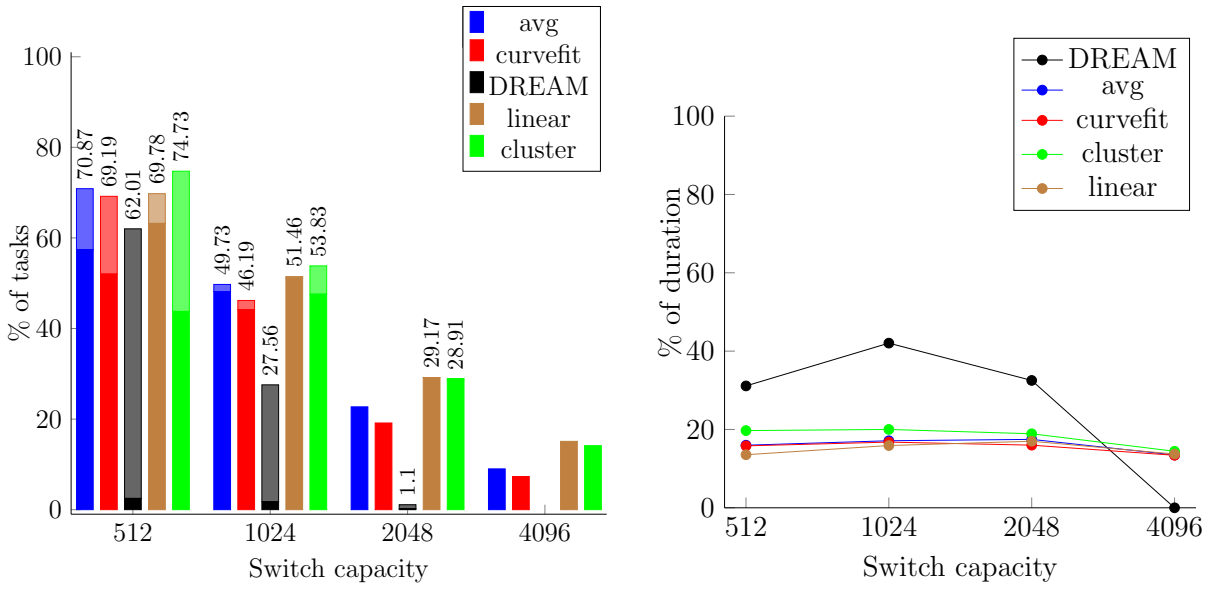
Figure 5.7: Drop ratio, Rejection ratio and Duration before Drop ratio for HH workload




(a) Drop and Rejection Ratios

(b) Duration Before Drop Ratio

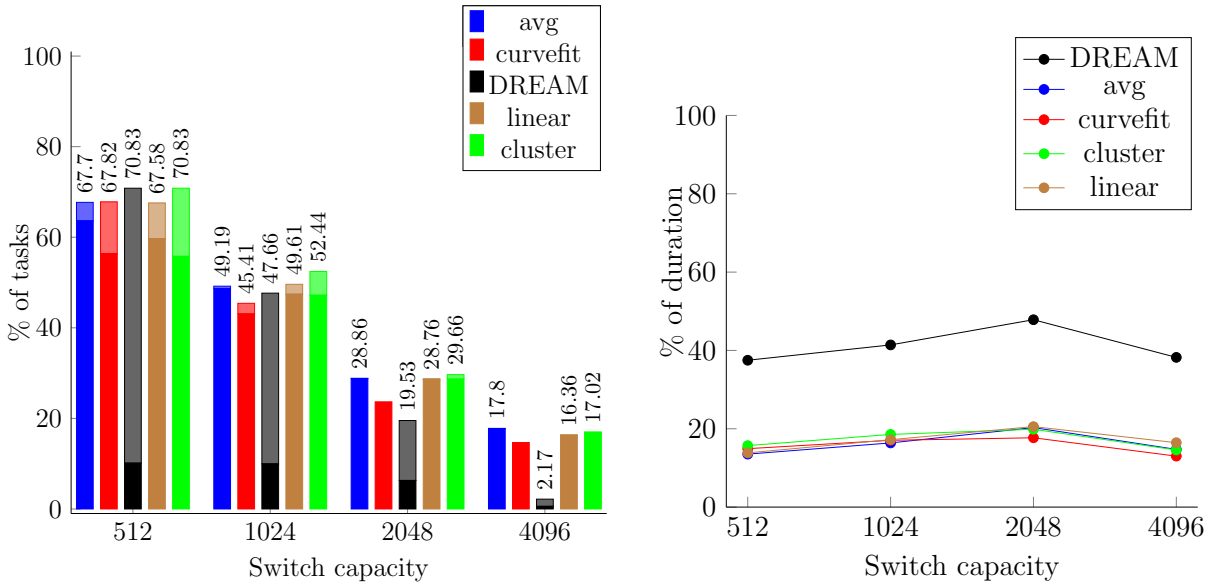
Figure 5.8: Drop ratio, Rejection ratio and Duration before Drop ratio for CD workload




(a) Drop and Rejection Ratios

(b) Duration Before Drop Ratio

Figure 5.9: Drop ratio, Rejection ratio and Duration before Drop ratio for HHH workload




(a) Drop and Rejection Ratios

(b) Duration Before Drop Ratio

Figure 5.10: Drop ratio, Rejection ratio and Duration before Drop ratio for Combination workload

### 5.1.3 Parameter Sensitivity Analysis

To understand how sensitive our results are to changes to parameters, we conduct several experiments using a fixed switch capacity of 1024 counters (a restrictive configuration) while varying other settings. For this set of results, we show the analysis for a specific type of task (HH) rather than using a combination workload, as this facilitates the interpretation of results (shown in Figures 5.11, 5.13 and 5.12). The qualitative behavior is similar for other types of tasks.

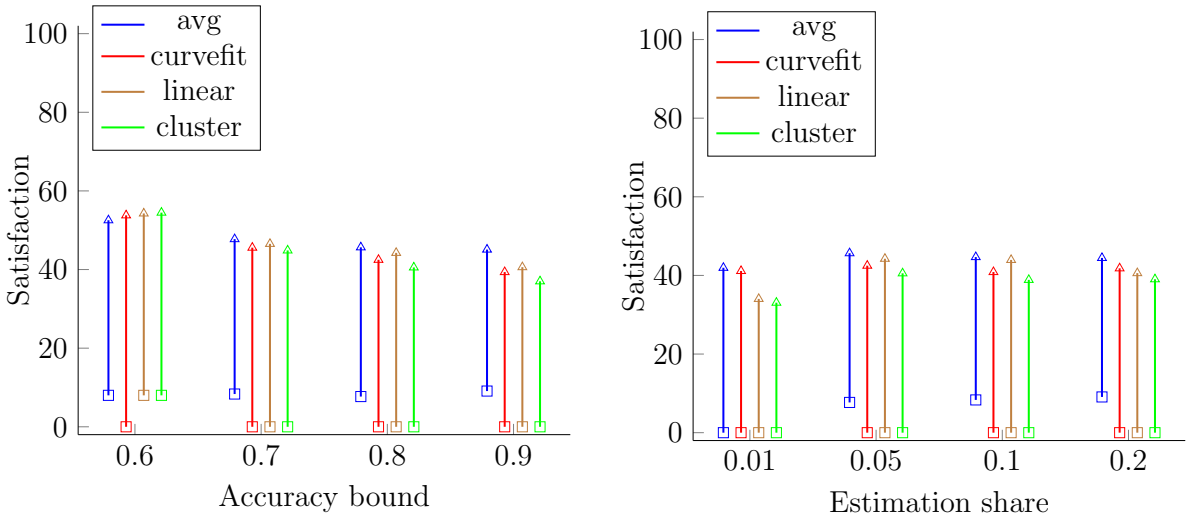
#### 5.1.3.1 Different Accuracy Bounds

The resources allocation becomes harder as the accuracy bounds increase because tasks, in general, need more resources to be satisfied. Among the estimation techniques the *avg* strategy brings the best results in term of average satisfaction and $5^{th}$ percentile for different accuracy bounds (Figure 5.11a), while its percentage of tasks that are either

dropped or rejected are no worse than 3% compared to the best for workloads with accuracy of 0.7 or below and is the best for accuracy bounds of 0.8 and higher (Figure 5.12a). For the average duration of tasks that are going to be dropped, the *curvefit* strategy ranks the best with a percentage between 10% and 14% (Figure 5.12b).

#### 5.1.3.2 Different Estimation Shares

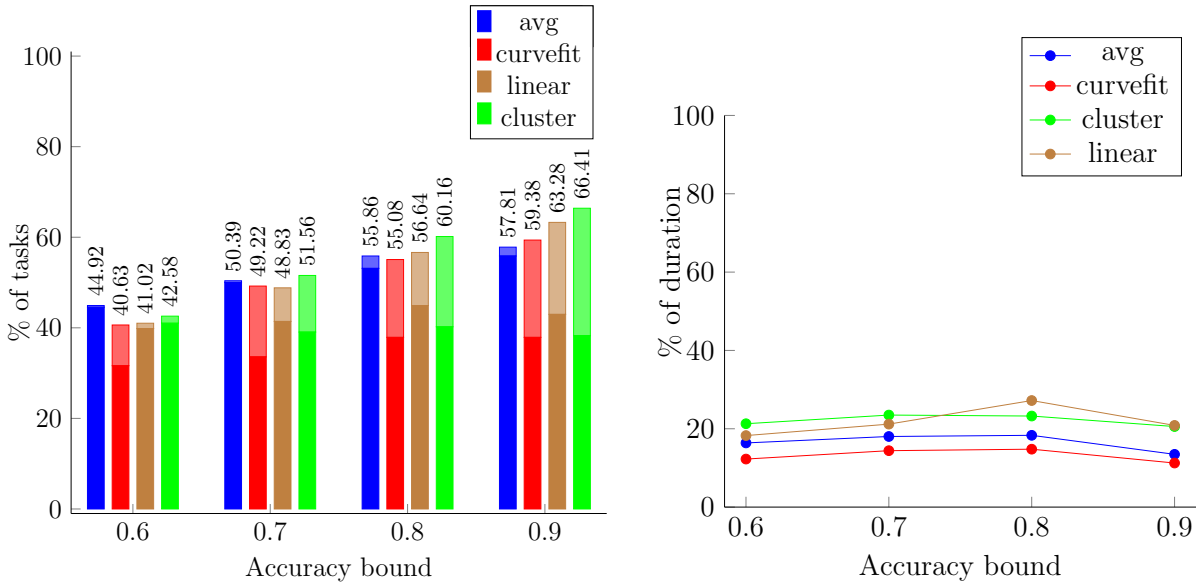
With increases in the estimation share parameter tasks would allocate more counter to predicted IPs pattern, thus having less amount of counter capacity to split down the IPs pattern using the original DREAM strategy. As it can be seen in Figures 5.11b and 5.13, tasks with estimation techniques have the best performance for satisfaction and ratios analysis when using a 5% counters allocation for estimation purposes with the average satisfaction ranking between 40% to 45% and the sum of the rejection and drop ratios ranking between 55% to 60% for all estimation strategies. This seems to confirm our assumptions that either a far too low or too high reserved resources allocation for estimation would hurt the overall tasks accuracy.




(a) Accuracy Sensitivity Analysis

(b) Estimation Share Sensitivity Analysis

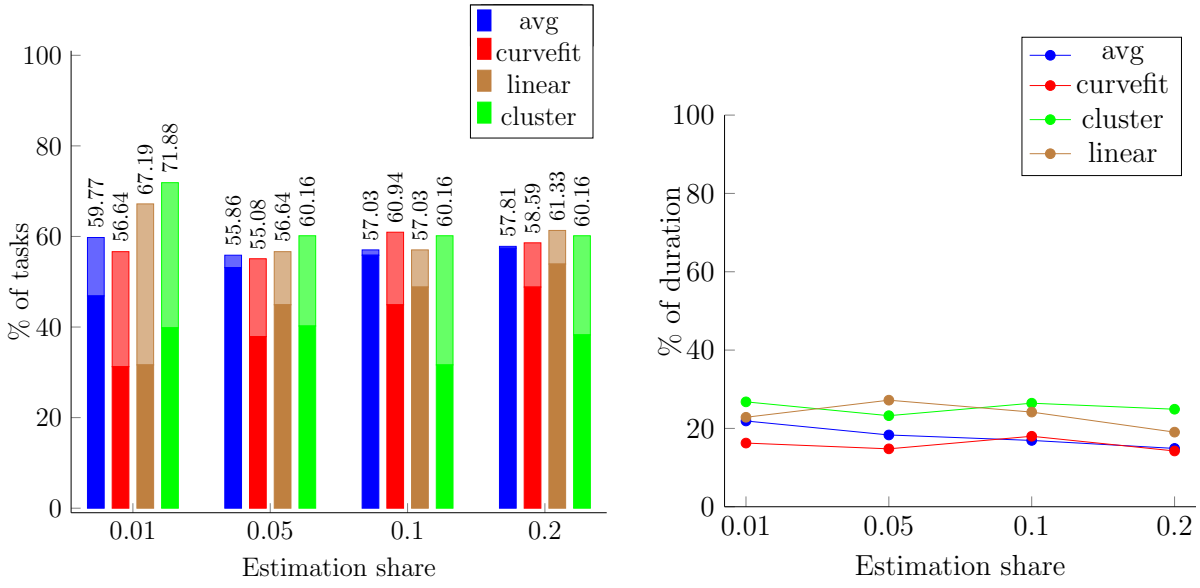
Figure 5.11: Satisfaction for Parameter Sensitivity Analysis




(a) Drop and Rejection Ratios

(b) Duration Before Drop Ratio

Figure 5.12: Drop ratio, Rejection ratio and Duration before Drop ratio for Accuracy sensitivity analysis




(a) Drop and Rejection Ratios

(b) Duration Before Drop Ratio

Figure 5.13: Drop ratio, Rejection ratio and Duration before Drop ratio for Estimation sensitivity analysis

## 5.2 ProgME Evaluation

In this section, we discuss the results from the implementation of the ProgME paradigm on DREAM. First, we analyze the Flowset Query Answering Engine (FQAE) scalability in Section 5.2.1 and the memory consumption of flowsets in Section 5.2.2. Then, we present an application scenario in Section 5.2.3 that can be used to debate the potentials of this proposal in domains such as traffic engineering and security monitoring, similar to what have been expressed by the ProgME authors [37].

### 5.2.1 Scalability of FQAE

FQAE provides an easy advantage in the resource-usage area. This is mainly due to the fact that its implementation uses per-flowset counters instead of the more common usage of per-flow counters.

By performing an empirical evaluation, we can show the scalability of FQAE by comparing the required number of counters for both approaches. Using CAIDA packets traces collected from the *equinix-chicago* monitor on 21/01/2016, we measure the average number of existing flows in 5-minutes traces splits from the original CAIDA data. Table 5.1 shows the results for 1-tuple and 2-tuple flows which occurs in large quantity (range $10^5 - 10^6$) for these tuple definitions.

Table 5.1: Average number of flows for 1-tuple and 2-tuple flows in a 5-minutes trace

| $< sip >$ | $< dip >$ | $< sip, dip >$ |
|---|---|---|
| $>= 50,000$ | $>= 150,000$ | $>= 300,000$ |

To simulate the use of FQAE, we have used several firewall configurations from a tier-1 ISP and Cisco firewalls, mapping firewall rules as flowsets. We consider each rule in the firewall as an initial user query for the FQAE. Table 5.2 shows the number of original queries and the number of disjoint queries based on the firewall configurations.

Table 5.2: Size of Queries

| Config | # flowsets (original) | # flowsets (disjoints) |
|---|---|---|
| #1 | 300 | 320 |
| #2 | 112 | 124 |
| #3 | 800 | 845 |

It can be observed that both the number of queries and the number of disjoint queries are significantly smaller than the number of observable flows from traffic traces, in accordance with the findings made in [37]. It is possible to argue that the potential number of independent flowsets generated by $n$ queries can grow as large as $m = 2^n$ in the worst-case scenario, if every flowset overlaps with all the others (line 12 in Algorithm 4). To avoid such situation, flowset without active flows can be merged into a large flowset.

This reduction in the number of used counters has two main implications for measurements architectures [37]:

- It makes possible to store counters in faster SDRAM registers, which is crucial for high speed networks.
- Networks can be monitored at a higher temporal resolution, thus enabling faster response rates to anomalous events.

### 5.2.2 Memory Consumption

FQAE can reduce the number of counters in use, thanks to its utilization of flowsets. However, a legitimate concern regarding its implementation is the memory footprint generated by the utilization of BDDs to represent flowsets as the underlying data structure. To better discern the memory cost associated to flowsets, we generate a scenario using flowsets to keep track of all bogon IP addresses, in CIDR blocks from the current bogon list [69], where each CIDR block is added in sequence one after the other into the flowset using the *union* operation.

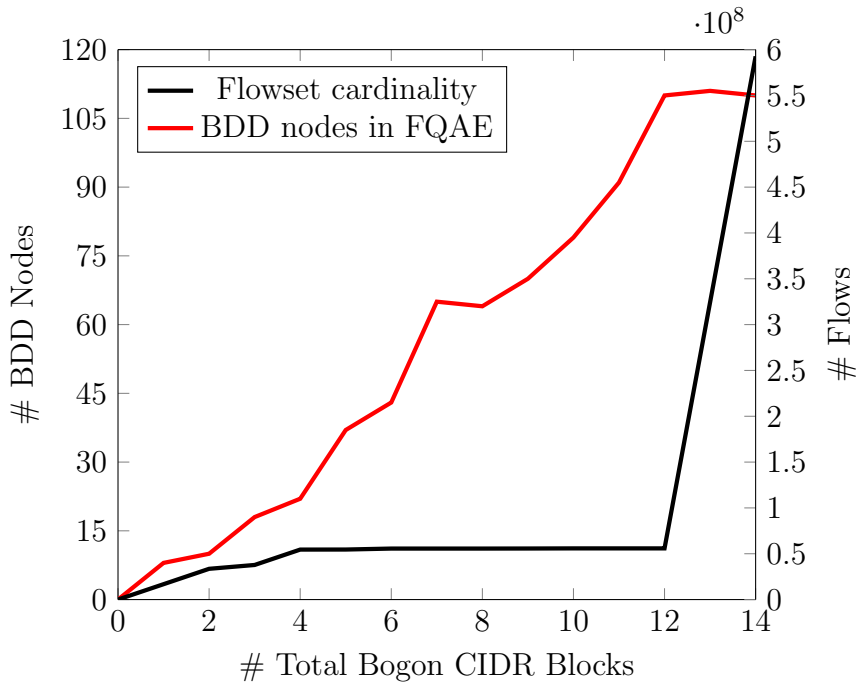



Figure 5.14: Bogons and flowset correlation

Figure 5.14 shows the resulting correspondence between each CIDR blocks and the number of required BDD nodes to represent the flowset *FQAE*. For further comparison, we also plot the flowset cardinality, which increases significantly faster than the number of used BDD nodes. The abrupt increment of the cardinality at the end of the list is a result of the addition of the last two bogon IP addresses: 224.0.0.0/4 and 240.0.0.0/4 in the CIDR list, each of them capable of matching $2^{28}$ unique IP addresses. However, the BDD representation does not use more than 120 nodes to generate the corresponding bogon flowset. These results match previous experimentations performed in the original ProgME research [37].

Note, that all the analysis has been made taking into account the number of BDD nodes instead of directing analyzing the *bytes* consumption. This is due to the existence of several available BDD packages with variable node size. This implementation is based on the JavaBDD package [26], which provides a Java native implementation, as well as interfaces to C++ implementations such as BuDDy and CuDDy.

### 5.2.3 Application Scenario: Tracking Bogons

To illustrate a possible usage for FQAE, we discuss how to track bogon packets, which are packets from IP addresses that are either reserved or unallocated. Since these IP addresses are spoofed, it is more meaningful to keep track of aggregated data such as the volume of bogon packets or the total number of occurrences.

Table 5.3: Comparison of tracking Bogons techniques

| Technique | # Counters | Accuracy | Computation |
|---|---|---|---|
| Per-flow | $6 \times 10^8$ | High | 1 Hash operation |
| FQAE | 1 | High | 1 BDD implication |

The current bogon list [69] has 14 CIDR block entries which amounts for close to $6 \times 10^8$ unique IP addresses, thus keeping per-flow statistics is not doable, although it has high accuracy and constant performance. Using FQAE, the bogons blocks can be precomputed into a single flowset representing the union of all the 14 different blocks. As a result, only one counter is necessary to keep track of every possible bogon packet.

## 5.3 Summary

In this chapter, we have examined how estimation strategies can help improve the accuracy of tasks running in DREAM and what the possible utilization of the ProgME implementation in DREAM is.

The estimation techniques predict the more likely patterns for the next measurement epoch based on previous processed data and those patterns are deployed in reserved subset of counters, thus maintaining the normal DREAM strategy while at the same time trying to gather better results from complete IPs patterns. The analysis results have shown that estimation techniques outperform DREAM in highly resource-constrained switches for almost every metric while for large-capacity switches the results are more even, with the exception of HHH workload, where DREAM have the advantage for these large-capacity switches.

For the ProgME paradigm implemented in DREAM, we provide an analysis of its main strengths and the advantages to use flowsets for coarse-grained measurements. The analysis shows that a flowset architecture reduces the number of counters required to use, which contrasts with other possible alternatives. We also illustrate a possible application scenario by tracking the bogon list and compare it with a per-flow implementation.

# Chapter 6

# Conclusion and Future Work

This chapter summarizes the thesis, discusses its main findings and contributions, and outlines areas for future research and development. The implemented features on top of the DREAM framework have proved useful to make it more appealing as a general framework for measurements analyses. However, still many extensions of this research deserve further consideration.

The chapter is divided into three main sections. Section 6.1 is a summary of the thesis. Section 6.2 discusses the future work, and finally Section 6.3 brings the thesis to a conclusion.

## 6.1 Summary of the thesis

This thesis has introduced the use of estimation techniques for measurement tasks in DREAM using a reserved amount of counters from the task counters mapping to install predicted IPs patterns, while the remaining counters are deployed using the original DREAM strategy. Using different estimation algorithms, our tests results showed that tasks produce better results than original DREAM tasks in nearly every analyzed metric, and that DREAM system can handle more tasks concurrently.

In addition, we have implemented a component to allow the creation of programmable

metrics using flowsets, in conformity with the guidelines from ProgME. We demonstrated the feasibility of the proposed implementation through an application scenario to track bogon packets. Using flowsets and programmable metrics, DREAM becomes a more scalable solution, by reducing the number of counters required to measure a set of IPs, and more complete as well, by facilitating the analysis of coarse-grained measurements.

## 6.2 Future Work

While this thesis has demonstrated the potential of improvements for traffic measurements using DREAM framework, many opportunities for extending the scope of this research remain. Below, we present those we think are the most important ones.

- **Pool of estimation algorithms** Currently, tasks can choose from four different estimation algorithms to work with: EWMA Smoothed Average, Polynomial Curve Fitting, KMeans++ Cluster and Pseudo-Linear Extrapolation. In order to enhance the flexibility and broaden the scope of DREAM framework, more choices of estimation techniques should be added into the system.

- **Fine-grained metrics using Flowsets** Flowset architecture has been implemented to enable coarse-grained measurements in DREAM. Thus, new algorithms could be designed to facilitate the extraction of specific metrics (such as HHH) from the flowset structure while being compatible with the resource constraint characteristics that are present in DREAM.

## 6.3 Conclusions

Measurement is fundamental for network management systems. This new version of the DREAM framework enhances its current capabilities with the additions of prediction-based techniques, which improve tasks accuracy and concurrency in the system, and

programmable metrics, which facilitate tasks scalability and coarse-grained measurement analysis.

# References


[1] J. S. Armstrong, "Forecasting by extrapolation: Conclusions from 25 years of research," *Interfaces*, vol. 14, no. 6, pp. 52–66, 1984.

[2] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Trans. Comput.*, vol. 35, no. 8, pp. 677–691, Aug. 1986.

[3] S. Arlinghaus, *Practical Handbook of Curve Fitting*. CRC Press, 1994.

[4] P. A., *Chapter 4 – traffic modelling and measurements*, 2000. [Online]. Available: `http://www.netlab.tkk.fi/opetus/s38145/s00/lectures/lect04.pdf`.

[5] S. Hazelhurst, A. Attar, and R. Sinnappan, "Algorithms for improving the dependability of firewall and filter rule lists," in *Dependable Systems and Networks, 2000. DSN 2000. Proceedings International Conference on*, 2000, pp. 576–585.

[6] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True, "Deriving traffic demands for operational ip networks: Methodology and experience," *IEEE/ACM Trans. Netw.*, vol. 9, no. 3, pp. 265–280, Jun. 2001.

[7] C. Williamson, "Internet traffic measurement," *IEEE Internet Computing*, vol. 5, no. 6, pp. 70–74, Nov. 2001.

[8] C. Estan and G. Varghese, "New directions in traffic measurement and accounting," *SIGCOMM Comput. Commun. Rev.*, vol. 32, no. 4, pp. 323–336, Aug. 2002.

[9] N. B. Azzouna and F. Guillemin, "Analysis of adsl traffic on an ip backbone link," in *Global Telecommunications Conference, 2003. GLOBECOM '03. IEEE*, vol. 7, Dec. 2003, 3742–3746 vol.7.

[10] F. Baboescu, S. Singh, and G. Varghese, "Packet classification for core routers: Is there an alternative to cams?" In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, vol. 1, Mar. 2003, 53–63 vol.1.

[11] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava, "Finding hierarchical heavy hitters in data streams," in *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29*, ser. VLDB '03, VLDB Endowment, 2003, pp. 464–475.

[12] G. Janssen, "A consumer report on bdd packages," in *Integrated Circuits and Systems Design, 2003. SBCCI 2003. Proceedings. 16th Symposium on*, IEEE, 2003, pp. 217–222.

[13] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen, "Sketch-based change detection: Methods, evaluation, and applications," in *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement*, 2003, pp. 234–247.

[14] C. Estan, K. Keys, D. Moore, and G. Varghese, "Building a better netflow," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 4, pp. 245–256, Aug. 2004.

[15] A. Kumar, M. Sung, J. ( Xu, and J. Wang, "Data streaming algorithms for efficient and accurate estimation of flow size distribution," *SIGMETRICS Perform. Eval. Rev.*, vol. 32, no. 1, pp. 177–188, Jun. 2004.

[16] M. Pióro and D. Medhi, *Routing, Flow, and Capacity Design in Communication and Computer Networks*. Morgan Kaufmann Publishers Inc., 2004.

[17] G. Cormode and S. Muthukrishnan, "An improved data stream summary: The count-min sketch and its applications," *J. Algorithms*, vol. 55, no. 1, pp. 58–75, Apr. 2005.

[18] G. Cormode and S. Muthukrishnan, "Space efficient mining of multigraph streams," in *Proceedings of the Twenty-fourth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, ser. PODS '05, 2005, pp. 271–282.

[19] T. Karagiannis, K. Papagiannaki, and M. Faloutsos, "Blinc: Multilevel traffic classification in the dark," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 4, pp. 229–240, Aug. 2005.

[20] A. W. Moore and D. Zuev, "Internet traffic classification using bayesian analysis techniques," *SIGMETRICS Perform. Eval. Rev.*, vol. 33, no. 1, pp. 50–60, Jun. 2005.

[21] S. Acharya, J. Wang, Z. Ge, T. F. Znati, and A. Greenberg, "Traffic-aware firewall optimization strategies," in *2006 IEEE International Conference on Communications*, vol. 5, Jun. 2006, pp. 2225–2230.

[22] M. Crovella and B. Krishnamurthy, *Internet Measurement: Infrastructure, Traffic and Applications*. John Wiley & Sons, Inc., 2006.

[23] H. Hamed and E. Al-Shaer, "Dynamic rule-ordering optimization for high-speed firewall filtering," in *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security*, ser. ASIACCS '06, 2006, pp. 332–342.

[24] V. Sekar, N. Duffield, O. Spatscheck, J. van der Merwe, and H. Zhang, "Lads: Large-scale automated ddos detection system," in *Proceedings of the Annual Conference on USENIX '06 Annual Technical Conference*, 2006, pp. 16–16.

[25] D. Arthur and S. Vassilvitskii, "K-means++: The advantages of careful seeding," in *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '07, 2007, pp. 1027–1035.

[26] J. Whaley. (2007), [Online]. Available: `http://javabdd.sourceforge.net/`.

[27] F. Khan, L. Yuan, C.-N. Chuah, and S. Ghiasi, "A programmable architecture for scalable and real-time network traffic measurements," in *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ser. ANCS '08, 2008, pp. 109–118.

[28] A. Callado, C. Kamienski, G. Szabo, B. Gero, J. Kelner, S. Fernandes, and D. Sadok, "A survey on internet traffic identification and classification," *Commun. Surveys Tuts.*, vol. 11, no. 3, pp. 37–52, Jul. 2009.

[29] N. M. M. K. Chowdhury and R. Boutaba, "Network virtualization: State of the art and research challenges," *Comm. Mag.*, vol. 47, no. 7, pp. 20–26, Jul. 2009.

[30] C. So-In, "A survey of network traffic monitoring and analysis tools," *Cse 576m computer system analysis project, Washington University in St. Louis*, 2009.

[31] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'10, 2010, pp. 19–19.

[32] W. John, S. Tafvelin, and T. Olovsson, "Review: Passive internet measurement: Overview and guidelines based on experiences," *Comput. Commun.*, vol. 33, no. 5, pp. 533–550, Mar. 2010.

[33] P. Čisar and S. M. Čisar, "Optimization methods of ewma statistics," *Acta Polytechnica Hungarica*, vol. 8, no. 5, pp. 73–87, 2011.

[34] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: Scaling flow management for high-performance networks," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 254–265, Aug. 2011.

[35] L. Jose, M. Yu, and J. Rexford, "Online measurement of large traffic aggregates on commodity switches," in *Proceedings of the 11th USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, ser. Hot-ICE'11, USENIX Association, 2011, pp. 13–13.

[36] P. Li and C.-h. Zhang, *A new algorithm for compressed counting with applications in shannon entropy estimation in dynamic data*, 2011.

[37] L. Yuan, C.-N. Chuah, and P. Mohapatra, "Progme: Towards programmable network measurement," *IEEE/ACM Trans. Netw.*, vol. 19, no. 1, pp. 115–128, Feb. 2011.

[38] ONF, "Software-defined networking: The new norm for networks," Open Networking Foundation, Tech. Rep., Apr. 2012.

[39] E. Portal, "Network functions virtualization: An introduction, benefits, enables, challenges and call for action," Oct. 2012. [Online]. Available: `http://portal.etsi.org/NFV/NFV_White_Paper.pdf`.

[40] M. K. Shin, K. H. Nam, and H. J. Kim, "Software-defined networking (sdn): A reference architecture and open apis," in *2012 International Conference on ICT Convergence (ICTC)*, Oct. 2012, pp. 360–361.

[41] R. Jain and S. Paul, "Network virtualization and software defined networking for cloud computing: A survey," *IEEE Communications Magazine*, vol. 51, no. 11, pp. 24–31, Nov. 2013.

[42] M. Jarschel, T. Zinner, T. Höhn, and P. Tran-Gia, "On the accuracy of leveraging sdn for passive network measurements," in *Telecommunication Networks and Applications Conference (ATNAC), 2013 Australasian*, Nov. 2013, pp. 41–46.

[43] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, Feb. 2013.

[44] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, 2013.

[45] M. Moshref, M. Yu, and R. Govindan, "Resource/accuracy tradeoffs in software-defined measurement," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '13, 2013, pp. 73–78.

[46] S. Sezer, S. Scott-Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, "Are we ready for sdn? implementation challenges for software-defined networks," *IEEE Communications Magazine*, vol. 51, no. 7, pp. 36–43, Jul. 2013.

[47] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with opensketch," in *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, ser. nsdi'13, 2013, pp. 29–42.

[48] Y. Zhang, "An adaptive flow counting method for anomaly detection in sdn," in *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '13, 2013, pp. 25–30.

[49] T. Zinner, M. Jarschel, T. Hossfeld, W. Kellerer, T. U. München, T. Zinner, M. Jarschel, T. Hossfeld, and P. Tran-gia, *A compass through sdn networks*, 2013.

[50] N. L. M. van Adrichem, C. Doerr, and F. A. Kuipers, "Opennetmon: Network monitoring in openflow software-defined networks," in *2014 IEEE Network Operations and Management Symposium (NOMS)*, May 2014, pp. 1–8.

[51] B. N. Astuto, M. Mendonça, X. N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: past, present, and future of programmable networks," *Communications Surveys and Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[52] M. Dusi, R. Bifulco, F. Gringoli, and F. Schneider, "Reactive logic in software-defined networking: Measuring flow-table requirements," in *2014 International Wireless Communications and Mobile Computing Conference (IWCMC)*, Aug. 2014, pp. 340–345.

[53] M. Malboubi, L. Wang, C. N. Chuah, and P. Sharma, "Intelligent sdn based traffic (de)aggregation and measurement paradigm (istamp)," in *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, Apr. 2014, pp. 934–942.

[54] M. Moshref, M. Yu, R. Govindan, and A. Vahdat, "Dream: Dynamic resource allocation for software-defined measurement," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, 2014, pp. 419–430.

[55] J. Rasley, B. Stephens, C. Dixon, E. Rozner, W. Felter, K. Agarwal, J. Carter, and R. Fonseca, "Planck: Millisecond-scale monitoring and control for commodity networks," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 407–418, Aug. 2014.

[56] J. Suh, T. T. Kwon, C. Dixon, W. Felter, and J. Carter, "Opensample: A low-latency, sampling-based measurement platform for commodity sdn," in *Proceedings of the 2014 IEEE 34th International Conference on Distributed Computing Systems*, ser. ICDCS '14, IEEE Computer Society, 2014, pp. 228–237.

[57] I. Ahmad, S. Namal, M. Ylianttila, and A. Gurtov, "Security in software defined networks: A survey," *IEEE Communications Surveys Tutorials*, vol. 17, no. 4, pp. 2317–2346, 2015.

[58] I. Alsmadi and D. Xu, "Security of software defined networks," *Comput. Secur.*, vol. 53, no. C, pp. 79–108, Sep. 2015.

[59] Y. Gong, X. Wang, M. Malboubi, S. Wang, S. Xu, and C.-N. Chuah, "Towards accurate online traffic matrix estimation in software-defined networks," in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, ser. SOSR '15, 2015, 26:1–26:7.

[60] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90–97, Feb. 2015.

[61] M. Moshref, M. Yu, R. Govindana, and A. Vahdat, "Scream: Sketch resource allocation for software-defined measurement," in *ACM International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, Dec. 2015.

[62] A. Yassine, H. Rahimi, and S. Shirmohammadi, "Software defined network traffic measurement: Current trends and challenges," *IEEE Instrumentation Measurement Magazine*, vol. 18, no. 2, pp. 42–50, Apr. 2015.

[63] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "Research challenges for traffic engineering in software defined networks," *IEEE Network*, vol. 30, no. 3, pp. 52–58, May 2016.

[64] Z. Bozakov, A. Rizk, D. Bhat, and M. Zink, "Measurement-based flow characterization in centrally controlled networks," in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, Apr. 2016, pp. 1–9.

[65] C. Xing, K. Ding, C. Hu, and M. Chen, "Sample and fetch-based large flow detection mechanism in software defined networks," *IEEE Communications Letters*, vol. 20, no. 9, pp. 1764–1767, Sep. 2016.

[66] (). Caida anonymized internet traces 2016, [Online]. Available: `http://www.caida.org/data/passive/passive_2016_dataset.xml`..

[67] (). Caida: The cooperative association for internet data analysis, [Online]. Available: `http://www.caida.org/`.

[68] Cisco, "Introduction to cisco ios netflow - a technical overview," Tech. Rep. [Online]. Available: `http://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_white_paper0900aecd80406232.html` (visited on 10/2016).

[69] T. Cymru. (). Bogon bit notation list v7.0 27 april 2012 - team cymru, [Online]. Available: `http://www.team-cymru.org/bogon-bit-notation.html`.

[70] (). Floodlight, [Online]. Available: http://www.projectfloodlight.org/floodlight.

[71] (). Ntop, [Online]. Available: http://www.ntop.org.

[72] (). Open vswitch, [Online]. Available: http://openvswitch.org/.

[73] (). Sflow, [Online]. Available: https://www.sflow.org/.

[74] (). Snort, [Online]. Available: https://www.snort.org/.

[75] (). Tcpdump & libcap, [Online]. Available: http://www.tcpdump.org/.

[76] (). Wireshark, [Online]. Available: http://www.wireshark.org/.