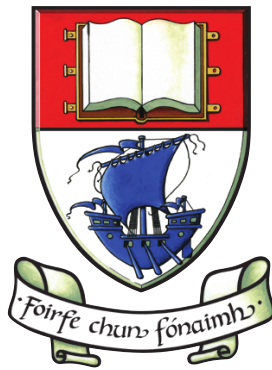


# Advances in Traffic Classification Techniques and in Network-aware Allocation of Datacenter Resources



**Runxin Wang, BSc, MSc**

School of Science and Computing

Waterford Institute of Technology

This dissertation is submitted for the degree of

*Doctor of Philosophy*

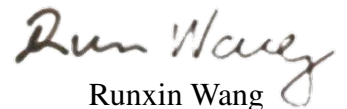
Supervisors: Dr Lei Shi and Dr Brendan Jennings

September 2016

*To my wife Xin Wang and my family*

## **Declaration**

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Doctor of Philosophy, is entirely my own work and has not been taken from the work of others save to the extent that such work has been cited and acknowledged within the text of my work.

A handwritten signature in black ink that reads "Runxin Wang". The signature is written in a cursive style with a large, sweeping initial 'R'.

Runxin Wang  
September 2016

## **Acknowledgements**

I would like to express my sincere thanks and appreciation to my supervisors, Dr. Lei Shi and Dr. Brendan Jennings, who provided me with their brilliant guidances throughout my PhD research. They are both great researchers who have visions and ideas while also possess practical skills and experiences. Lei Shi, who has acted as my supervisor and also friend of mine in my life, spent his great deal of time to teach me efficient skills to conduct research, and inspired me to ask interesting research questions and to deal with the challenges. Brendan Jennings, as my mentor, who always kept me on a right path in research with his great perspective, taught me how to do rigorous research with his great attention on details, and supported me to challenge different ideas and experiences. I am fortunate and thankful to have them on my journey to PhD.

I am also grateful to all the co-authors in my publications and the engineers and researchers that I have worked with, I learned a lot through working with them and getting valuable advices from them. The work in this thesis is a result of collaboration from them, these brilliant people include Bernard Butler, Michael Barros, Lei Xu, Biao Xu, Alan Davy, Mícheál Ó Foghlú, Eric Robson, Juliano Wickboldtz, Rafael Esteves, Lisandro Granville, Simone Mangiante, Anaëlle Maillard, John Furlong and Longhao Zou, their contributions of efforts and intelligences are valuable to my research.

Finally, to my parents, my wife and my family, who are always supportive, thank you for being with me.

## **Abstract**

Emerging applications create tremendous volume of data traffic every day, increasingly stressing the capabilities of the networks hosting these applications. Understanding the communication patterns of these applications can significantly benefit the corresponding network management tasks. Motivated by today's wide use of data analysis techniques, this thesis studies how to perform effective network management through using network data analysis. We investigate problems in managing the QoS targets and resources in enterprise networks and datacenter networks, presenting two applications of traffic analysis.

The first application of traffic analysis addresses the utilisation of traffic classification techniques in enterprise networks to differentiate servicing flows of different classes or applications. The traffic classification techniques developed based on statistical analysis produce advantages over the existing techniques based on deep packet inspections or port matching in the scenarios that new applications emerge and packets are encrypted. This dissertation presents enhanced traffic classification techniques based on traffic analysis.

The second application addresses the requirement of effective resource allocation in datacenters, where predictable performance is essential to the applications hosted in datacenters. This thesis proposes resource allocation techniques that can allocate datacenter resources in a way that the QoS requirements of Virtual Machines (VM) are met while minimizing the amounts of required resources. This is achieved through the integrated use of bin packing algorithms, effective bandwidth estimation techniques and software-defined networking technologies, where network data are analyzed to identify the minimum amounts of bandwidth required to meet the QoS targets for provisioning VMs on demand.



# Contents

<b>Abstract</b>	<b>vi</b>
<b>List of Figures</b>	<b>xi</b>
<b>List of Tables</b>	<b>xiv</b>
<b>List of Algorithms</b>	<b>xv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Contributions . . . . .	3
1.2 Structure of the Thesis . . . . .	4
<b>2 Literature Review</b>	<b>6</b>
2.1 Traffic Classification . . . . .	6
2.2 Datacenter Resource Allocation . . . . .	10
2.3 Progress Beyond the State of the Art . . . . .	17
<b>3 Training traffic classifiers with arbitrary packet sets</b>	<b>19</b>
3.1 Introduction . . . . .	19
3.2 Problem Statement and Analysis . . . . .	20
3.3 Training Traffic Classifiers on arbitrary packet sets . . . . .	26
3.3.1 The Orders of Packets in Flows . . . . .	26

---

3.3.2	Feature Set Design . . . . .	26
3.3.3	STD of Packet Length and Inter-packet Length . . . . .	27
3.3.4	C4.5 Decision Tree . . . . .	28
3.4	Experimental Evaluation . . . . .	29
3.4.1	Evaluation Methodology . . . . .	29
3.4.2	Experimental Results . . . . .	31
3.5	Summary . . . . .	33
<b>4</b>	<b>Ensemble classifier for traffic in presence of changing distributions</b>	<b>36</b>
4.1	Introduction . . . . .	36
4.2	Changing Traffic Distributions . . . . .	37
4.2.1	Change Detection . . . . .	38
4.3	The Traffic Classification System . . . . .	40
4.3.1	Ensemble Traffic Classifier . . . . .	40
4.3.2	Feature Set & Base Classifier . . . . .	42
4.3.3	Weight Computation . . . . .	43
4.3.4	Ensemble Training . . . . .	44
4.4	Simulation on New Patterns Interference . . . . .	45
4.4.1	Experimental Setup . . . . .	45
4.4.2	Experimental Results . . . . .	46
4.5	Summary . . . . .	49
<b>5</b>	<b>Network Aware VM Placement Using Effective Bandwidth Estimations</b>	<b>51</b>
5.1	Introduction . . . . .	51
5.2	Effective Bandwidth Review and Residual Bandwidth Estimation . . . . .	53
5.2.1	Effective Bandwidth and its Estimation . . . . .	53
5.2.2	Residual Bandwidth Adjustment . . . . .	54
5.3	MAPLE Architecture . . . . .	57



---

5.3.1	EB Agents . . . . .	57
5.3.2	MAPLE Controller . . . . .	58
5.4	Network-aware VM Placement Algorithms . . . . .	60
5.4.1	MAPLEx—Supporting Anti-colocation Constraints . . . . .	64
5.5	Evaluation . . . . .	65
5.5.1	Experimental Setup . . . . .	65
5.5.2	Residual Bandwidth Required to Place New VMs . . . . .	68
5.5.3	Analysis of QoS Violations . . . . .	69
5.5.4	Analysis of Rejection Rates of VM Ensemble Requests . . . . .	73
5.5.5	Analysis of MAPLEx . . . . .	75
5.6	Summary . . . . .	77
<b>6</b>	<b>QoS-aware Multipathing in Datacenters Using Effective Bandwidth Estimation and SDN</b>	<b>79</b>
6.1	Introduction . . . . .	79
6.2	Effective Bandwidth in multipath topologies: Effective Bandwidth Coefficient	81
6.3	MAPLE-Scheduler . . . . .	83
6.4	MAPLE Flow Scheduling . . . . .	86
6.5	Evaluation . . . . .	89
6.5.1	Experimental Setup . . . . .	89
6.5.2	The Analysis of QoS Violations . . . . .	90
6.5.3	Analysis of Throughput Performance and Link Utilization . . . . .	92
6.6	Summary . . . . .	94
<b>7</b>	<b>Conclusions &amp; Future Work</b>	<b>97</b>
7.1	Future Work . . . . .	98
	<b>Bibliography</b>	<b>101</b>



# List of Figures

2.1	Three different approaches of traffic classification. . . . .	7
2.2	VMs connected through virtual link over a datacenter has multipath topology.	11
3.1	The selection of packets for traffic classification. . . . .	21
3.2	The impact of packet selection on the calculation of inter-packet arrival time.	23
3.3	The comparison of three techniques based on recall metric. . . . .	32
3.4	The comparison of three techniques based on precision metric. . . . .	33
3.5	Recall on three different classes. . . . .	34
3.6	Precision on three different classes. . . . .	35
4.1	Distributions of P2P traffic over 3 days in the UNIBS trace. . . . .	39
4.2	The Offline and Online Modules of the adjustable traffic classification system.	41
4.3	Classification accuracy in simulation of changing distributions. . . . .	47
4.4	Recall on the injecting traffic. . . . .	48
4.5	Precision on the injecting traffic. . . . .	49
4.6	Updating time on growing training flows. . . . .	50
5.1	The estimation of residual bandwidth based on effective bandwidth technique.	55
5.2	The growth of required bandwidth when VMs being added to a link. . . . .	56
5.3	The MAPLE system architecture. . . . .	57
5.4	The Logical View of MAPLE Controller. . . . .	58

---

5.5	A matrix of required traffic rates based on VMs' topology. . . . .	59
5.6	The datacenter testbed for MAPLE. . . . .	66
5.7	Additional bandwidth required to allocate a new VM placed on a server. . .	69
5.8	QoS violation levels for QoS target (0.02s,0.1) for MAPLE and Oktopus variants. . . . .	71
5.9	CDF of the mean link utilization measured on the datacenter testbed, resulted by MAPLE and Oktopus variants. . . . .	72
5.10	QoS violation rates for different delay based QoS targets, resulted by MAPLE and Oktopus variants. . . . .	73
5.11	Rejection rates of ensemble replacement request for MAPLE and Oktopus variants. . . . .	74
5.12	QoS violation rates for different delay based QoS targets, resulted by MAPLE and MAPLEx. . . . .	75
5.13	Rejection rates of ensemble placement request for MAPLE and MAPLEx. .	76
5.14	CDF of the mean link utilization on the datacenter testbed, resulted by MAPLE and MAPLEx. . . . .	77
6.1	Throughput estimation at endpoint while multiple up links exist. . . . .	80
6.2	Capturing traffic on virtual switch by using port mirroring. . . . .	83
6.3	The logical architecture of MAPLE-Scheduler and MAPLE system. . . . .	85
6.4	The datacenter testbed for MAPLE-Scheduler. . . . .	89
6.5	QoS violation levels for QoS target (0.01s,0.1) for MAPLE-Scheduler and ECMP. . . . .	91
6.6	QoS violation levels for QoS target (0.04s,0.05) for MAPLE-Scheduler and ECMP. . . . .	92
6.7	QoS violation levels for QoS target (0.02s,0.1) for MAPLE-Scheduler and ECMP. . . . .	93
6.8	Throughput achieved under the QoS constraint (0.04s, 0.05), resulted by MAPLE-Scheduler and ECMP. . . . .	94

---

6.9	CDF of the mean link utilization on datacenter testbed, resulted by MAPLE-Scheduler and ECMP under the QoS constraint (0.02s, 0.1). . . . .	95
6.10	CDF of the mean link utilization on datacenter testbed, resulted by MAPLE-Scheduler and ECMP under the QoS constraint (0.04s, 0.05). . . . .	96

# List of Tables

3.1	The feature set. . . . .	27
3.2	Traffic Traces. . . . .	30
3.3	The overall classification accuracy. . . . .	31
3.4	The classification accuracy on three classes. . . . .	33
4.1	The feature set. PS denotes Payload Size. . . . .	43
4.2	Traffic Traces. . . . .	46
4.3	The data blocks injected with new traffic. . . . .	46
4.4	Blocks with Change, +: true -: false. . . . .	49
6.1	Before rescheduling. . . . .	86
6.2	After rescheduling. . . . .	87

# List of Algorithms

1	Change Detection. . . . .	39
2	Accuracy Weighted Ensemble Algorithm. . . . .	44
3	MAPLE VM Ensemble Placement . . . . .	62
4	allocBetweenNodes function . . . . .	63
5	MAPLEx VM Ensemble Placement . . . . .	65
6	MAPLE Flow Scheduling Algorithm . . . . .	88

# Chapter 1

## Introduction

Traffic analysis has been a long running research topic in the area of networking and data communications. Early work on analyzing traffic data focused on generating accurate statistics (Claffy, 1994) and traffic modelling (Paxson and Floyd, 1995) to profile networks and guide network planning and engineering. Today there is a significant trend towards more and more data analysis applications being deployed in different business areas to extract valuable information from collected data. Motivated by the wide use of data analysis, this thesis studies how to perform effective network management using traffic analysis techniques. In particular, it investigates problems relating to managing the Quality of Service (QoS) and resource allocations in enterprise networks and in datacenter networks, where the specific challenges and issues drive the choices of the applied traffic analysis techniques. Essentially this thesis focuses on the applications of traffic analysis, in contrast to the earlier work that focused on the implementation and accuracy of the techniques.

The first application of traffic analysis studied in this thesis addresses the problem of developing advanced traffic classification techniques. Traffic classification plays an important role in many network management and security tasks of enterprise networks, in which traffic flows need to be identified before appropriate configurations can be applied. For instance, network security appliances need to identify malicious incoming traffic patterns and block their sources at the network edge. In recent years traffic classification has become an increasingly challenging task and the drawbacks of the traditional classification methods have gained increased attention.

Port based classification and Deep Packet Inspection (DPI) are the traditional methods used in traffic classification. The former becomes unreliable when dynamic port assignment is



employed and the latter cannot deal with encrypted traffic. These issues motivated researchers to study new techniques to develop effective traffic classification. Recently a number of methods based on traffic analysis techniques have been proposed with promising results. These methods classify traffic based on flow statistics, making them more flexible and effective in practical networking environments. In this dissertation, we investigate how to implement advanced traffic classification techniques based on traffic analysis, where we found there exists two issues on the related state of art. Firstly, existing techniques typically rely on using the first five packets in a flow to proceed, result in decreasing performance when first five packets are unavailable, or maliciously injected. Secondly, traffic statistics change over time can also result in decreasing classification accuracy. This thesis presents novel proposals to addresses these issues by using improved traffic analysis techniques and evaluates the proposals based on the experiments using publicly available traffic traces.

The second application of traffic analysis addresses the problem in resource allocation in datacenters. Nowadays datacenters have been used as the shared infrastructure that provide elastic computing resources on demand to hundreds and thousands of end users. In fact, most of the widely used Internet-based applications today cannot survive without deploying their essential components in datacenters; these applications include search (e.g., Google, Bing), video streaming(e.g., YouTube, NetFlix), social networking (e.g., Facebook, Twitter), and big data analytics, to name a few. In particular, cloud computing provides a typical usage of datacenters, where computing resources from some datacenters can be leased to public end users to provide a elastic, agile and cost-effective approach to satisfy the customers' ever-growing IT needs.

Without adequate network infrastructures, datacenters cannot properly support the performance requirements of mission critical multi-tier applications. Studies in Greenberg et al. (2009) and Mogul and Popa (2012) have identified that the unpredictable network performance in datacenters had become a bottleneck to many datacenter-hosted applications. Recent work on datacenter network management has focused on allocating network resources to tenants' VMs at optimal cost while ensuring performance requirements are met. Typically, these approaches can be divided into two directions: 1) network-aware VM placement techniques that aim to jointly allocate sufficient resources to the requested VMs when they are placed, where the joint resources can be shared among the VMs from the same tenants to meet the performance needs; and 2) flow schedulers that can dynamically (re)schedule flows over links to ensure the links attached to the VMs are not congested, so their end performance can be guaranteed. We found that these existing approaches greatly

focus on providing predictable throughputs for tenant VMs, while lacking of mechanisms that directly address the needs for latency-aware applications.

In addition, there is a gap between network-aware VM placement systems and the Software-Defined Networking (SDN) (Hu et al., 2014; McKeown et al., 2008) based flow schedulers used in datacenter networks. On one hand the VM placement techniques have little control over the flows: if subsequently overloading occurs on the links connected to the VMs the required performance may not be maintained. On the other hand, flow schedulers allocate flows to the links without knowing the performance requirements of the end VMs connected to links; thus, some opportunities are missed as the affected performance can be resolved by effectively allocating the flows based on the performance requirements of the corresponding VMs. This thesis proposes to construct resource allocation techniques based on the use of traffic analysis to address the described issues and evaluate the proposed techniques in datacenter testbeds.

## 1.1 Contributions

This thesis emphasizes the applications of traffic analysis on solving practical problems in network management, where the conditions present in enterprise networks and datacenter networks variously drive the choices of traffic analysis techniques to be used. The material in this work reviews the challenges and experiences on deploying and utilizing data analysis techniques in networking environments, and outlines major implementation details and building blocks, with the end goal to provide experiences and insights to network operators who are interested to apply traffic analysis techniques in their network management systems. The contributions of this thesis can be summarized as follows:

- We propose a traffic classification techniques based on the use of arbitrary packet sets, which relaxes the dependency of the existing techniques that use strictly first five packets to classify traffic. Our proposal enables existing traffic classification systems to deal with flows with missing packets. Moreover, the proposed technique can be adopted to mitigate the issue that payload mutation techniques are used by malicious applications intending to evade classification.
- We propose an adjustable traffic classification system, the key technique of which is ensemble classification, assisted with a change detection method. Our system enables traffic classifiers to be effectively updated in response to the changing statistical

distributions occurring in traffic data. Existing traffic classification techniques cannot cope with statistical changes in data in an effective manner, which over time means traffic data exhibits new characteristics, often resulting in decreased accuracies in the existing classification techniques. Our proposed technique can maintain a reasonable accuracy in such circumstances.

- Studying the application of traffic analysis onto datacenter resource management, we present MAPLE, a network-aware VM placement system that accomplishes the following objectives: 1) provision of predictable network performance to tenant's VM ensembles; 2) optimal joint-allocation of network and computing resources; and 3) satisfaction of applications' QoS targets. The traffic analysis technique used in this work is effective bandwidth estimation, where effective bandwidth is defined as "the minimum amount of bandwidth required by a traffic source to maintain specified QoS targets." The essential functionality of MAPLE is a network-aware VM placement algorithm that uses an effective bandwidth technique in conjunction with the first fit decreasing approach to generate near-optimal VM placement solutions in a timely manner.
- Extending MAPLE, we present a flow scheduler, MAPLE-Scheduler, which ensures that the QoS requirement can be maintained over time after the initial placement of the application VMs. This proposed system addresses the gap between: 1) VM placement that has no control to the workloads which can change over time resulting in QoS decrease, and 2) the flow scheduler that is not QoS-aware and may only consider balancing the workload ignoring that some VMs may need more bandwidth to maintain the QoS requirements specified during the VM placement.

## 1.2 Structure of the Thesis

This chapter presents the motivations and introduces the proposals of applying traffic analysis techniques to traffic classification and datacenter network management, and summarizes the contributions of this research. The remainder of this thesis is organized as follows.

Chapter 2 respectively introduces the backgrounds and the state-of-the-art techniques in traffic classification and resource allocation in datacenters and highlights the identified contributions beyond this state-of-the-art.

Chapter 3 describes the proposal for training traffic classifiers based on using arbitrary packet sets. §3.3 presents the feature set of traffic statistics selected to train the traffic classifier, the applied analysis techniques and the technique for generating features from arbitrary packet sets. §3.4 describes the experimental setup and evaluates the proposed classifier.

Chapter 4 presents the adjustable traffic classification system designed to address the issue of changing distributions in traffic statistics. §4.2 illustrates the issue of changing distributions and proposes a method to detect changes in the traffic distribution. §4.3 presents the system design and implementation of the proposed adjustable traffic classification system. §4.4 describes a simulation of traffic in presence of new patterns, where the proposed traffic classifiers were compared to two exiting techniques.

Chapter 5 presents MAPLE, the proposed network-aware VM placement system. The traffic analysis techniques applied in this proposal is empirical effective bandwidth estimation, the concept of which is introduced in §5.2. In §5.3 and §5.4 we respectively describe MAPLE system design, and the network-aware VM placement algorithm and its algorithmic variant, MAPLEx, which facilitates with anti-colocation constraints. Experimental evaluation is presented in §5.5, where MAPLE/MAPLEx are evaluated and compared to Oktopus (Ballani et al., 2011) based on a testbed constructed by using four servers.

Chapter 6 presents MAPLE-Scheduler, a QoS-aware flow scheduler developed based on MAPLE. The concept of effective bandwidth coefficient is introduced in §6.2, which is an effective bandwidth technique applied in this proposal to reduce the overhead of deploying traffic sniffers throughout datacenters. In §6.3 and §6.4 the MAPLE-Scheduler is described and its flow scheduling algorithm is specified. In §6.5, MAPLE-Scheduler is evaluated based on a testbed constructed using four physical servers, where hundreds of VMs, multi-layers of virtual switches and SDN controller were used to emulate a SDN enabled datacenter.

Chapter 7 provides concluding remarks of this thesis and outlines the potential future work.

# Chapter 2

## Literature Review

### 2.1 Traffic Classification

Finding out what applications are running on a network, provides information for many other tasks in network management. Traffic classification is the task of identifying the source applications that cause traffic on a network. It is an important part in the construction of QoS networks. With the growing complexity and dynamics of Internet applications and the use of traffic classification evasion techniques, this task is becoming increasingly difficult.

As illustrated in 2.1, there are basically three different methods of traffic classification. The earliest traffic classification method, namely port based classification, classifies traffic by matching the well known ports that are regulated by Internet Assigned Numbers Authority (IANA) (IANA, 1972) to identify the source applications, e.g., traffic running on ports are classified as HTTP traffic as specified by IANA. Due to more and more applications are using ports that are not formally documented, port based classification become inaccurate. Deep packet inspection (DPI) is a relatively advanced method, which classifies traffic by learning the special strings or patterns exhibited in the content of packets, while it become less effective when packet payloads are encrypted.

Traffic analysis based traffic classification technique has been proposed in recent years. Significant research has shown that there are statistical differences in the way that different Internet applications communicate, and these statistics can be learned and applied to classify traffic flows and distinguish them from each others. As this technique does not rely on port numbers nor reading the packet payload, it does not have any of the shortcomings that exist in other techniques. Below discusses some selected research published in the literature, for

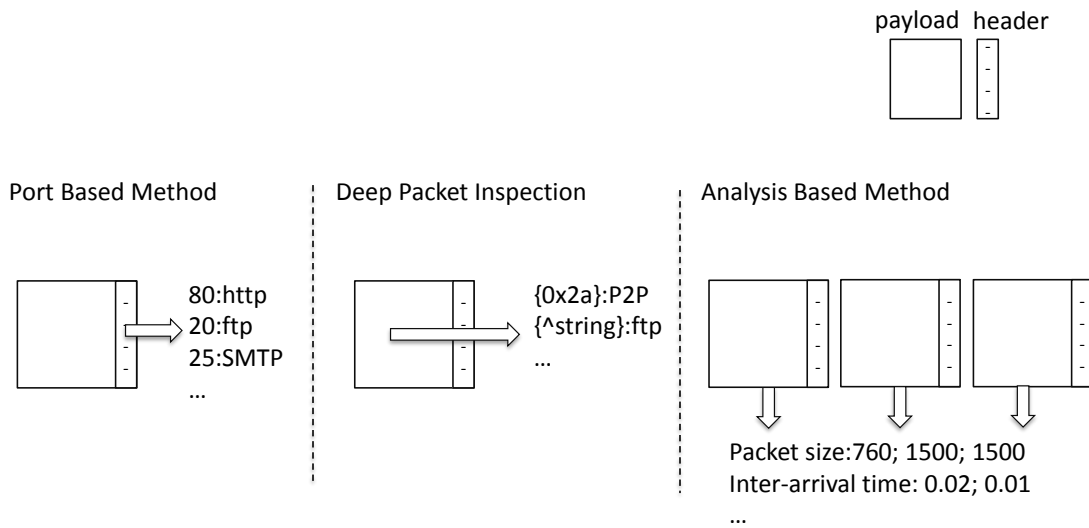


Fig. 2.1 Three different approaches of traffic classification.

further reading, regarding the recent developments and issues in traffic identification and classification techniques can be found in Callado et al. (2009) and Dainotti et al. (2012).

Among all the traffic classification studies, one common problem is how to ensure the ground truth of the applications, without which the related experiments cannot be appropriately tested. Szabó et al. (2008) and Gringoli et al. (2009) respectively provided their methods to generate the ground truth data for testing novel traffic classification proposals. Alternatively, some implemented traffic classification packages, such as Tie (Dainotti et al., 2009), tstat (Finamore et al., 2011) and NeTraMark (Lee et al., 2011), can be applied to the public traffic traces to generate the referencing results.

In 2004, Roughan et al. (2004) found that a key factor holding back widespread QoS adoption is the absence of classifying different Internet traffic to different QoS classes. The authors accordingly propose an approach to classify traffic in terms of the class of services. Different traffic applications may be classified into the same class if they require the same level of service, for example, FTP traffic and Kazaa traffic are all classified as bulk data. A 3-stage process is used to achieve the classification of quality of service, namely statistics collection, classification and rule creation. They demonstrate the performance of Nearest Neighbor and Linear Discriminant Analysis algorithms on a pre-processed traffic dataset.

Moore and Zuev (2005) developed one of the early proposals which demonstrated flow statistics can be applied to build traffic classifiers. One problem in those early proposals of ML based traffic classification is that many flow statistics used by those methods can be computed only after flows are terminated.

To address the issue, Bernaille et al. (2006a) and Bernaille et al. (2006b) investigate classifying traffic flows at an early stage by using only the first few packets in flows. Their classification method is based on clustering. In experiments, they showed that generally traffic flows could be accurately classified by only using the statistics of their first 5 packets. Our work applies their concept, aiming to classify flows by only using the first few packets. They used Bayesian Analysis to classify traffic flows based on the statistical features of traffic. They investigated 248 statistical features on classify traffic flows, where they demonstrated that the classification accuracy can be significantly improved by carefully selecting some useful features. They also demonstrate that the Bayes Kernel (BK) estimation is an effective method to classify traffic flows, as it overcomes the drawback of Gaussian assumption made by using the naïve Bayes method. However, the proposed method need to calculate numeric features of a traffic flow, making it infeasible for online classification, since one may not block or apply QoS treatment on the flow after it is finished.

Nguyen and Armitage (2006) proposed to train traffic classifiers on subflows, partial flows have the consecutive packets that within some time windows arriving after the start of flows. This proposal aimed to address the practical issue in real networks that traffic classifiers may not always see the actual start of a flow. The authors applied the Naïve Bayes algorithm to train classifiers and demonstrated some subflows provide good sources to train traffic classifiers with good accuracy.

Instead of focusing on the flow behaviours, Karagiannis et al. (2005) proposed a novel idea for classifying traffic based on identifying the behaviours of hosts that transport the traffic. They present a heuristic technique for classifying traffic in the dark, called BLINC, which tries to capture the profiles of hosts, in terms of the destinations and ports they communicate with and identify applications that the hosts are engaged in. Once the profiles of the hosts are constructed, the traffic flows connecting to the known hosts can be easily classified by labelling them as the applications that the known hosts engaged with.

Kim et al. (2008) performed investigation on traffic classification based on traffic traces collected from 4 different links from an academic network. The authors reported some interesting findings that the number of bytes and the duration of flows are less correlated. They also found that flash flows contribute great fluctuation in flow counts, therefore, the

authors proposed that ignoring flash flows can improve the accuracy of the involved traffic classification systems.

Este et al. (2009) studied the performance of Support Vector Machine (SVM) on 3 public traffic traces. They demonstrated how to develop a multi SVM classifiers to manage the diverse Internet applications, where each SVM classifier is trained to identify one specific application. They proposed a feature set for the SVM classifier, which is different from those feature sets used commonly in the previous proposals. Based on their evaluation, the SVM classifier is effective at discriminating traffic of different applications.

Finamore et al. (2010) developed a traffic classification engine KISS, which is specialized to classify UDP traffic. KISS utilizes both DPI technique and SVM, it generates statistical signatures for traffic data by using the Chi-Square ( $\chi^2$ ) test. Experimental results showed that their method had high accuracy on identifying UDP applications. Our method, on the other hand, is developed to classify both TCP and UDP traffic.

Lim et al. (2010) studied the sources of discriminating power of ML based techniques on classifying traffic flows. They compared a couple of techniques using different metrics, where they demonstrated that flow direction is important and some packets are relatively more significant than the others for traffic classification. Our work adapts some of their suggested features and a classifier.

Nechay et al. (2009) specially study the performance guarantees on the resource-intensive flows in traffic classification, aiming to control the False Alarm Rate (FAR) and False Discovery Rate (FDR) of these flows. They proposed two traffic classifiers based on Neyman-Pearson classification and learning satisfiability framework. The proposed methods are validated on traffic dataset provided by a Canadian ISP, which are shown to largely reduce the FAR and FDR.

Dainotti et al. (2009) and Dainotti et al. (2011) applied some combining classification techniques and ensemble techniques to construct traffic classifiers, with the goal to improve the classification accuracy. They tested several ensemble techniques with promising results. However, their proposals did not focus on applying ensemble techniques to handle the changing distributions issue, also they did not investigate how to detect change in traffic.

Zander and Armitage (2011) implemented a traffic classification software DIFFUSE, which is released as open source. Their main consideration is to make the software working on high-speed network to classify hundred thousands concurrent flows while using commodity



hardware resources. In experiments, they also demonstrate that DIFFUSE can classify traffic of subflows with good accuracy.

Lee and Lee (2012) proposed an early work on applying big data technique to accelerate traffic classification process in order to address the scalability issue due to Internet traffic can easily grow into tremendous data set requires matching computing and storage resources. The author present a Hadoop-based traffic monitoring system that performs traffic analysis that can deal with multi-terabytes of traffic in a scalable manner.

Ng et al. (2015) reported their learned experiences on developing an SDN based traffic classification prototypes and evaluated them for an enterprise network. The authors implemented two types of traffic classifiers, one is policy based traffic classifier and the other is statistical classifier. The essential benefit of using SDN is resulted from its central controller and adding programming ability into traffic classification systems. The SDN based traffic classifier can mark the flows for continuing observations rather than making static decisions based on fix number of packets, and it has tables to hold the records of the flows thus the states of flows can be changed on dynamic basis. In particular, on contrast to traditional traffic classifiers that must return determinative types for the types of the flows, SDN based traffic classifier can return actions to the controller, which provides more flexibility to the system to define the types of the flows (e.g., flows can have multi-types) and make corresponding decisions.

## 2.2 Datacenter Resource Allocation

Datacenters are used widely by the range from small medium enterprizes and research institutes to large multi-sites global companies. Empowered by the virtualization technologies, datacenters provide an economic way to share computing resources to end users, enabling them to be flexibly increase their need resources to meet their computing demands. The use models that allow virtual machines can be leased from the public clouds offering Infrastructure as a Service (IaaS) attracts more interest of use and adoptions.

Given that effective resource utilization is the one of the major factors driving the increasing use of datacenters, how to effectively and or optimally provide the virtualized resources to meet the ever-increasing demands is a question has been drawing a great deal of attention. Essentially, datacenter are networked systems, their network performance often found to be the bottleneck of the applications running on datacenters. Allowing user to define network

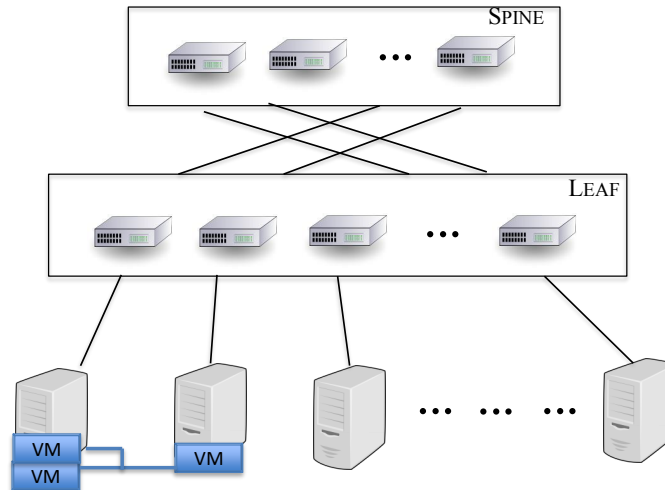


Fig. 2.2 VMs connected through virtual link over a datacenter multipath topology.

need, then accordingly providing bandwidth guarantee to tenant VMs is a straightforward way to realise predictability, though in the long run it tends to waste network resource.

Hierarchical topologies are commonly used in datacenter networks where multipaths are provided to create redundant paths between end-to-end servers. As a typical example given in 2.2, spine layer refers to the aggregated switches used to provide redundant paths to the end switches, the leaf layer that connects to end servers. Without a dedicated flow control mechanism, different sizes of traffic flows are statically allocated to links without sensing the current link utilization, which can result in transient network congestion that impacts on the latency experienced by users of the hosted applications. To address this issue, recent advances in datacenter network largely apply Software-Defined Networking (SDN) (Hu et al., 2014; Kreutz et al., 2015) technologies to implement novel effective flow scheduling techniques that can effectively distribute workloads over the network links. SDN has gained great interests in the literature due to it separates the control plane from the data plane on OpenFlow-enabled switches (Lara et al., 2014; McKeown et al., 2008), which opens opportunity to novel flow forwarding and controlling mechanisms can be implemented and deployed to those switches.

Below proposals presents novel ideas on addressing the aforementioned issues. For further references, the surveys Bitar et al. (2013); Mogul and Popa (2012) cover the broad discussions and development of technologies and protocols used in datacenter networks.

Guo et al. (2010) proposed SecondNet, which describes the idea of leasing virtual data center (VDC) rather than only VMs to cloud users. The proposal allows users to specify their bandwidth guarantee in their VDCs by using traffic matrices. An entity in a traffic matrix represent the bandwidth requirement between a VM pair. The main component in this proposal is the VDC manager, which creates VDCs based on the traffic matrices. Three basic services are defined in this system: a high priority end-to-end guaranteed service, a better than best-effort service that offers bandwidth guarantees for the first/last hops of a path, and a best effort service. To realize the bandwidth guarantee, port-switching source routing (PSSR), a packet forwarding mechanism is used, which forwards packets using predefined port numbers instead of MAC addresses. SecondNet moves information about bandwidth reservations from switches to hypervisors (Popek and Goldberg, 1974), and achieves high scalability by using PSSR. In addition, it allows virtual resources to be dynamically added to or removed from VDCs to realize elasticity.

Benson et al. (2011a) described CloudNaaS, which focuses on deploying enterprise applications in clouds. It studied the requirements of typical enterprise applications including application-specific address spaces, broadcasting and bandwidth reservation etc., and accordingly design the architecture of the virtualized datacenters. Although many previous data center virtualization proposals have already addressed some of the requirements of enterprise applications, they do not fully address all of the above requirement, which motivates the proposal of CloudNaas. The network requirements is also defined by using traffic matrix. The implementation of CloudNaas is largely relied on using the technique of software defining networking – openflow (McKeown et al., 2008) and Open vSwitch. The openflow controller implemented in CloudNaas determines the placement of VMs and generates forwarding rules that can be installed on openflow switches. For VM allocation, the system developed a greedy bin packing heuristic for placing VMs subject to the VMs' communicating behaviours.

Rodrigues et al. (2011) proposed gatekeeper, another approach aims to provide predictable network performance. Gatekeeper provides the single big switch model as well, where each tenant VMs visually connected to one switch, implemented by using Open vSwitch (Pfaff et al., 2009). This switch is responsible for guaranteeing each its connected VM's in and out bandwidth. Gatekeeper uses a hypervisor based control mechanism to implement its

bandwidth allocation, where a VM can actually consume more bandwidth than its guarantee, if there are other VM underutilise bandwidth. The bandwidth requirement of a VM can be defined in terms of either or both maximum bandwidth and minimum bandwidth, a hypervisor can therefore address the actually bandwidth a VM needs at runtime. In this way, gatekeeper attempts to prevent congestions in a datacenter network.

Ballani et al. (2011) proposed Oktopus, which presents two virtual network abstractions, namely, virtual cluster and virtual oversubscribed cluster. These clusters are designed to control the trade-off between the performance guarantees and the corresponding costs. A virtual cluster is a virtual topology that all VMs are connected to a single virtual switch. A cloud users can choose the abstraction and the degree of the over-subscription of the virtual network based on the communication patterns of the application the user plans to deploy on the cluster. Oktopus offers better flexibility to the cloud providers, while allows cloud users to find a balance between application performance and cost. To allocate the virtual resources, Oktopus uses a greedy algorithm to map a cluster onto a physical data center network, instead of reserving every required bandwidth, the algorithm tries to save bandwidth amounts that VM pairs cloud not use.

Shieh et al. (2010) proposed Seawall, which establishes congestion-control tunnels for communicating pairs, through the tunnels, traffic are controlled according to some networking policies. Seawall aims to avoid using strict bandwidth reservation to guarantee network performance between VMs. Instead, it assigns weights to each VMs, where bandwidth is allocated based on weights in a proportional way. In general, Seawall allows VM to share bandwidth proportionally and accordingly enforce performance isolation on network. However, with Seawall, VMs can share network in a reasonable way, they may still lack of predictability due to the congestion-control tunnels dynamically allocate bandwidth based on weights.

Lam et al. (2012) described NetShare, another approach attempts to share datacenter network based on given weights. NetShare assigns bandwidth by virtualizing a data center network using a statistical multiplexing mechanism. The key novelty in Netshare is that, network links are shared in the level of services, applications, or corporate groups, rather than each single VM pairs. In this way, one service, application or group cannot consume more available bandwidth by opening more connections. Netshare needs to implement a shim layer at each end host while the benefit is that no change need to make in switches or routers. This weight based bandwidth allocation approach does not provide direct bandwidth guarantee to the VMs.

Meng et al. (2010) commented that most the current techniques for VM placement only consider CPU, memory and storage, yet ignoring that bandwidth is also an important resource. Without carefully allocating network resource, it could result in a VM pairs with intensive interactions being placed onto two distant location, where huge bandwidth could be used. The authors formulate the corresponding problem as traffic-aware VM placement (TVMP) problem, and solve the problem by using a two-tier approximate algorithm. The idea of the solution is to align VM communication patterns with the communication distances between VMs, i.e., communication cost can be saved if putting a VM pair that has heavy interaction into a location within short distance.

Shi et al. (2012) studied the problem of optimizing the allocation of VMs having different capacity requirements and placement constraints. Given a set of physical machines with known specifications, in order to achieve the objective of maximizing the cloud provider's revenue. The proposed approach is based on the formulation of the problem as an integer linear programming problem, which produces an optimal placement for VMs with different placement constraints and devises a plan to allocate VMs to servers in a way that maximizes revenue, having due regard both to customer requirements and server capacities.

Popa et al. (2012) analysed the constraints and trade-off in the design space for sharing datacenter networks, and described a set of policies in FairCloud that expresses the trade-off between bandwidth guarantee and network allocation proportionality. However, this proposal requires expensive support from switches.

Popa et al. (2013) extended some idea presented in FairCloud, and developed ElasticSwitch. This approach can effectively trade off the accurate bandwidth guarantee and high utilisation of network capacity, in order to provide bandwidth guarantee to VMs in a work-conserving way, as it does not required strict bandwidth reservation. ElasticSwitch created two layers to realise its design objectives: the guarantee partitioning layer ensures the bandwidth guarantees of each VM pair connection are made, and rate allocation layer observes the actual bandwidth each connection need and reallocate some unused bandwidth to the active connection, thus link utilisation is improved. Each layers is respectively placed on each hypervisors, they do not need any communication with the centre controller, as well no any hardware updated is needed.

Most proposals reviewed so far fail to capture that the cloud-hosted applications' network demands vary over time. Xie et al. (2012) proposed Proteus, a datacenter network placement system that allocates network resource based on the time-varying profiles of the applications. An observation is that the networking requirement of many applications experience

significant changes in bandwidth throughout their execution. Hence provisioning a single bandwidth  $B$  for an entire cluster or for each pair of VMs throughout the job execution is clearly wasteful. Proteus is developed to address the issue, which models the time-varying nature of the networking requirement of cloud applications, once stable bandwidth requirements of each VMs are known in some given periods, Proteus can accurately assign bandwidth to each VMs within a predictable time scale. However, the time scale used in this proposal is likely not sufficient long, on the order of tens of seconds, resulting in the management system need to frequently interact with the network.

LaCurts et al. (2013) described Choreo, another datacenter network management system that based on application profiling. Choreo has three sub-systems: a measurement component to obtain inter-VM network rates, a component to profile the data transfer characteristics of a distributed application; based on the previous traffic profiling component, an algorithm is designed to place VMs into the datacenter, in a way that the VM pair communicate often that are placed with a link of higher rate. The traffic profiling work mainly involves two tasks: Profiling the application to find its network demands; and Measuring the network to discover the bandwidth available between VM pairs; With using this information, VM workloads can be optimally placed to result in a predictable network performance.

VM bandwidth demands are variable, it is difficult to assign a fix value to indicate the bandwidth requirement between VMs. Wang et al. (2011) use random variables to characterize the future bandwidth usage, which makes a better profiling to the uncertainty of bandwidth demand. The VM bandwidth usage is assumed to form a Gaussian distribution, regarding the time scale is at least a week long. By this set up, the authors formulate the traffic-aware VM placement problem as a stochastic bin packing (SBP) problem in order to model the volatility of the demands of the network and subsequently to propose an online packing algorithm to optimise it. Their approximation algorithm is based on an variants of FFD.

Breitgand and Epstein (2012) extended the above work, to improve the VM placement algorithm from having a 2 approximation ratio to having a  $(2 + \epsilon)$  approximation ratio. The authors also assume the VM bandwidth demands obeys Gaussian distribution, as such the TVMP problem is also modelled as a SBP problem. Their technique can consolidate the VMs onto the minimal physical servers, with meeting all the constraints of resource requirements and network performance guarantee. As VMs do not require their maximal bandwidth simultaneously, this sharing resource can be multiplexed. With using their

proposed technique, they propose that cloud providers can offer some probabilistic guarantee for the network bandwidth on their service level agreement.

Jiang et al. (2012) proposed an approach based on joint VM placement and routing. Similar to the TVMP problem, the communication cost is captured by the links that the VMs will use, modelled by using the cost function proposed in previous study that is not based on data center network. The authors designed a dynamic online algorithm using Markov approximation technique to solve the online VM placement problem, where they assume the VM requests form a stationary distribution with a  $M/M/\infty$  queue.

Chrysos et al. (2014) describes a deep analysis of multipath routing outlines the weakness of ECMP and encourages more advanced packet-based routing mechanisms in datacenter networks. In the literature there are many proposals implementing such advanced mechanisms; we provide here an overview of the most significant of those that embodying SDN technologies and approaches using both centralized and distributed architectures.

MicroTE Benson et al. (2011b) proposes a fine-grained traffic engineering system using OpenFlow and a centralized network controller, but it requires minor modifications to end hosts. Hedera Al-Fares et al. (2010) implements an adaptive flow scheduling system for multi-stage commodity switching fabric and unmodified hosts, leveraging OpenFlow and a centralized controller. In contrast, the focus of our work is to improve static load-balancing methods (e.g., ECMP) in multipath tree topologies towards an efficient usage of network resources, concentrating on QoS target satisfaction.

Other approaches, such as Kim et al. (2010), Ishimori et al. (2013) and Curtis et al. (2011), try to improve QoS in OpenFlow networks by extending the OpenFlow protocol to support rate limiters, packet schedulers and more efficient actions. Particularly interesting is how DevoFlow (Curtis et al., 2011) achieve a scalable flow management: it keeps flows in the data plane as much as possible to avoid overheads in the controller, but it maintains enough visibility over network flows to provide effective aggregated flow statistics. It uses a trigger-based flow statistic gathering mechanism which inspired us to design an efficient link monitoring component.

Multipath TCP (MPTCP) (Raiciu et al., 2011) is a TCP extension being standardized at the IETF that enables a pair of hosts to use multiple paths to exchange data for a single connection. It establishes multiple parallel subflows using multiple IP addresses and each subflow performs its own congestion control. MPTCP can greatly improve performance in today's datacenters by exploring multiple paths simultaneously and by balancing traffic

across the subflows based on perceived congestion, leading to both higher network utilization and fairer allocation of capacity to flows.

Presto (He et al., 2015) pushes the load-balancing functionality to the edge nodes, requiring modifications to edge switches (implemented using Open vSwitch (Pfaff et al., 2009)). It achieves a near-optimal load balancing in a proactive manner, leveraging symmetry in the network topology and an improved version of Generic Receive Offload in Linux kernel which prevents packet reordering.

Many researchers have been exploring distributed solutions to QoS and congestion issues in networks. We highlight DIFANE (Yu2, 2010), one of the first contributions to this area, and the more recent Conga (Alizadeh et al., 2014), which proposes a congestion-aware load balancing mechanism for datacenters using a distributed approach and custom ASICs inside networking fabric. It is optimized for a 2-tier Clos topology (also called Leaf-Spine) typically deployed in enterprise datacenters. It uses a novel mechanism to convey path-wise congestion metrics between pairs of leaf switches operating on flowlets to achieve a higher granularity of control. Recently a new direction in flow balancing, out of SDN architecture, has been developed in Chowdhury et al. (2014), introducing the concept of coflows. Recent publications from the industry also show that SDN has been applied into wide area network. SWAN (Hong et al., 2013), published by Microsoft, is one of the first application of SDN in wide area networks and B4 (Jain et al., 2013) was successfully deployed in a significant portion of Google’s public facing global network, providing efficient centralized traffic engineering.

## 2.3 Progress Beyond the State of the Art

Existing traffic classification techniques that apply statistical analysis rely on using the first 5 packets. However, in today’s network, packet losses and reordering are still inevitable in many scenarios (Benson et al., 2010; Gill et al., 2011; Jain et al., 2013), while the absence of any of the first five packets can result in decreasing accuracy for these existing traffic classification techniques. In addition, the dependency of first 5 packets opens another issue that malicious applications can apply payload mutation technique in order to to evade classification (Cheng et al., 2012). We propose to train classifiers based on using arbitrary packet sets (Wang et al., 2013b), which relaxes the dependency of using first five packets, subsequently mitigates the aforementioned issue.



Another issue with existing techniques that construct classifiers based on using traffic statistics is that traffic data sets may occur statistical changes over time. As a result traffic classifiers trained on historical traffic data can suffer inaccuracy on new traffic, in which the underlying statistical models have changed. We propose an adjustable traffic classification system to cope with the issue (Wang et al., 2013a). Comparing to the two existing methods used in practice – 1) training a new classifier only on new traffic or 2) training a classifier on a large dataset that combines all data collected in different periods, the proposed technique provide comparable classification accuracy while having better efficiency on the training process of classifiers.

Regarding resource allocation in datacenters, existing network-aware VM placement techniques rely on static analysis of datacenter traffic to pre-define the bandwidth requirement for individual VMs, due to the need of throughput guarantee, they open tend to over-provision the network resources over time. Our proposed technique, MAPLE (Wang et al., 2016a, 2014), applies traffic analysis at run time, which can provide guaranteed latency for the placed VMs, while optimizing the resource provisioning. In addition, existing techniques generally focus on providing guaranteed throughput, while MAPLE focuses on providing guaranteed latency, in which guaranteed can be precisely specified in the following format “no more than 2% of packets should be delayed by more than 50ms”.

There is a gap between existing network-aware VM placement techniques and flow scheduling algorithms. Most network-aware VM placement techniques are designed based on the assumption that network resources can be abstracted as a hose model, and as long as the network requirements of VMs do not exceed the capacity specified, adequate resources can be secured. This assumption relies on the presence of a flow scheduler to ensure that every link has equal networking conditions, thus the hose model may not be aware if VMs are allocated with congested links (see illustrated example in Chapter 6). In addition, most existing flow scheduling techniques ignore the respective latency requirement needed by each VM—they only ensure that, overall, the flows are evenly allocated to the given links. To address the issue, this thesis describes MAPLE-Scheduler (Wang et al., 2016b), a technique built based on as an extension of MAPLE and the use of SDN, which ensures the latency required by the individual placed VM can maintained over time.

# Chapter 3

## Training traffic classifiers with arbitrary packet sets

### 3.1 Introduction

Chapter 2 has presented the motivations to implement traffic identification techniques based on traffic analysis. As discussed in the literature view, there is a practical requirement that traffic flows need to be identified sufficiently early in order not to delay the dependent network tasks. The main stream of research in advanced traffic identification has focused on inspecting the first five packets of flows to perform the identification (Bernaille et al., 2006a,b). Considering that the first five packets of flow are not always available, Nguyen and Armitage (2006) proposed to classify traffic using subflow based method, where a subflow is defined as a window of some consecutive packets within a flow.

However, the existing methods that require either first five packets or consecutive packets would suffer with loss in accuracy, when the required packets of the given flows are unavailable, if packet losses and out-of-orders are common in a given network. Even though a subflow based method can manage the cases that some packets in first five packets are not available, it cannot manage the case that some packets of a subflow are unavailable.

Unfortunately, recent studies show that packet losses and reordering are still inevitable in many scenarios (Benson et al., 2010; Gill et al., 2011; Jain et al., 2013). Another issue in the existing methods is that evasion techniques can be employed to confuse them. For example, payload mutation (Cheng et al., 2012) allows an attacker to intentionally change the payload size of packets, the evaded packets can then mislead a ML based traffic classifier to make

incorrect classification, if the classifier relies on inspecting the packet payload sizes to identify traffic.

Since traffic classifiers can be trained with using first five packets or some subsflows, with the aim to address the described issues, it is natural to raise a question: is it possible to train classifiers using packets arbitrarily selected from flows – packets are neither first five packets (e.g., packets of {1,2,3,4,5}) nor consecutive packets (e.g., packets of {4,5,6,7,8})? Instead they could be arbitrary sequences like {1,4,7,8,10} or {3,5,7,9,11}.

The work in this chapter demonstrate that it is possible to train traffic classifiers using some arbitrary packet sets with reasonable accuracy. The intention would be to employ such classifiers as auxiliary classifiers to use in conjunction with a classifier trained using first five packets. These auxiliary classifiers are switched on to perform traffic classification for those flows known to have packet losses or retransmissions. Moreover, this approach could be adopted to address the issue of evasion techniques. By using random packets, attackers need to guess the combination of the packets in order to evade the corresponding packets. Even though they knew the combination, a network operator can change the packet combination periodically, to increase the complexity to the attackers.

The rest of this chapter is structured as follows. §3.2 provides the problem statement on using arbitrary packet set to train traffic classifiers. §3.3 presents the feature set, the ML algorithm and the technique for generating features from arbitrary packet sets. §3.4 describes the experimental setup and evaluates the proposed classifier. A summary is drawn in §3.5.

## 3.2 Problem Statement and Analysis

ML based traffic classification techniques generally apply a set of statistics of traffic packets to build the classification model. This analysis use Support Vector Machine (SVM) as the classification model. The fundamental assumption here is that there exists a classification model built by SVM classifying the traffic flows on a network, Este et al. (2009) provided some studies on implementing traffic classification techniques using SVM. SVM classifies instances by finding a hyperplane that can perfectly separate the involved instances into two sides. 3.1 gives the hyperplane function, which is also the classification function.

$$w \cdot X + b = y \quad (3.1)$$

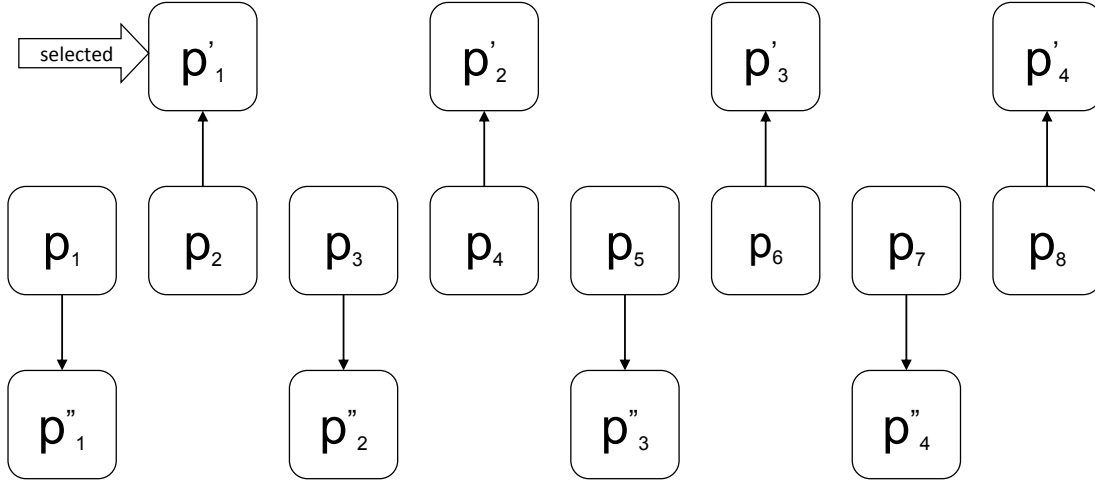


Fig. 3.1 The selection of packets for traffic classification.

Packet length and packet inter-arrival time are the base statics in use, from which generally a set of features  $X = x_1, x_2, \dots, x_n$  are derived, e.g., mean packet length of first 4 packets and mean inter-arrival time, then the the feature set is used for traffic classification.

The random selection of packets would directly impact the derived statics of packet length and inter-arrival time, the question here is if the impact would defect the classification function?

Our argument is that the selection of packets only introduces modification to the original classification function but does not defect the function. In other words, the original function with appropriate modification would still be capable to classify traffic flows.

This work illustrate the argument by addressing two problems, which respectively correspond to packet length and inter-arrival time. We start with packet length as inter-arrival time is a relatively difficult problem.

**Problem 1:** Given  $X = \{\text{mean length of first } N \text{ packets}, X = \sum_{i=1}^N p_i / N\}$ ,  $p_i$  is the length of  $i$ th packet, the classification function is  $w \cdot X + b = y$ , find out the classification function for  $X' = \{\text{mean length of } K \text{ packets selected from } N\}$ .

**Solution 1:** We use  $p'_i$  denotes the length of  $i$ th packet in the selected  $K$  packets, and  $p''_i$  denotes the length of  $i$ th packet in the unselected packets.

$$\begin{aligned}
X' &= \frac{\sum_{i=1}^K p_i'}{K} \\
&= \frac{\sum_{i=1}^N p_i}{N} - \frac{\sum_{i=1}^{N-K} p_i''}{N-K} \\
&= X - \frac{\sum_{i=1}^{N-K} p_i''}{N-K}
\end{aligned}$$

$X'$  is the mean length of  $K$  packets, so  $X' = E[p']$ , likewise  $\sum_{i=1}^{N-K} p_i''/N-K = E[p'']$ , hence

$$E[p'] = X - E[p'']$$

We assume that  $K$  packets selected over  $N$  packets is a binomial process with rate  $q$  and the length of packet has a distribution function  $\phi(T)$  corresponding to time  $T$ , where the  $N$  packets are all generated within time  $T$ . Hence, we have  $E[p'] = \sum_{i=1}^k q\phi(T)$  and  $E[p''] = \sum_{i=1}^{N-K} (1-q)\phi(T)$ .

$$\frac{E[p'']}{E[p']} = \frac{\sum_{i=1}^{N-K} (1-q)\phi(T)}{\sum_{i=1}^K q\phi(T)}$$

$$\frac{E[p'']}{E[p']} = \frac{(N-K)(1-q)}{Kq}$$

$$E[p''] = \frac{(N-K)(1-q)}{Kq} \cdot E[p']$$

Apply the above equation to  $E[p'] = X - E[p'']$  with  $X' = E[p']$

$$E[p'] = X - \frac{(N-K)(1-q)}{Kq} \cdot E[p']$$

$$X' = X - \frac{(N-K)(1-q)}{Kq} \cdot X'$$

$$X = \left(1 + \frac{(N-K)(1-q)}{Kq}\right) X'$$

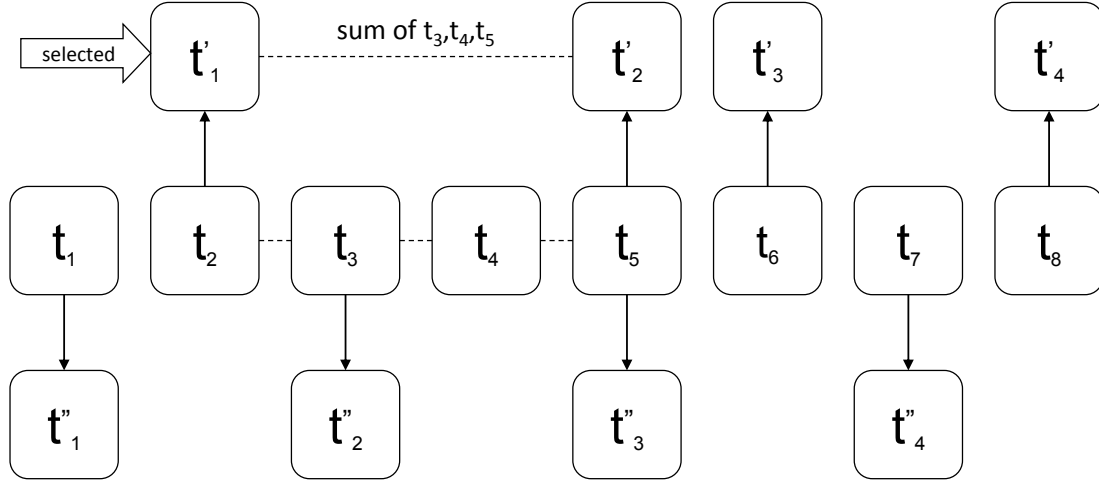


Fig. 3.2 The impact of packet selection on the calculation of inter-packet arrival time.

Apply the above equation to the classification function  $w \cdot X + b = y$ ,

$$w \cdot X + b = w \cdot \left(1 + \frac{(N-K)(1-q)}{Kq}\right) X' + b$$

Hence the classification function for  $X'$  is  $w \cdot X' + b = y$  with parameter  $w' = w \cdot \left(1 + \frac{(N-K)(1-q)}{Kq}\right)$ .

**Problem 2:** Given  $X = \{\text{mean inter-arrival time of first } N \text{ packets}, X = \sum_{i=1}^N t_i / N\}$ ,  $t_i$  is the interval time of  $i$ th packet, the classification function is  $w \cdot X + b = y$ , find out the classification function for  $X' = \{\text{mean inter-arrival time of } K \text{ packets selected from } N\}$ .

**Solution 2:** We use  $t'_i$  denotes the interval-arrive time of  $i$ th packet in the selected  $K$  packets, and  $t''_i$  denotes the interval-arrive time of  $i$ th packet in the unselected packets.

In this problem, the random selection of packet impact the inter-arrival time in the selected packets, the following illustrates the change introduced by the random selection, the value of  $t'_i$  is not equal to a center  $t_i$  but becomes the sum of a series of  $t_i$ .

$$t'_i = \sum^s t_i$$

$s$  is the period between the  $t_i$ .

$$X' = \frac{\sum_{i=1}^K t'_i}{K}$$

$$= \frac{\sum^K \sum^s t^i}{K}$$

Based on the above, the sum of  $t'_i$  is related to the sum of  $t_i$ , there are 2 cases in total depending on if the first and last packets are both selected:

$$\sum_{i \in K} t'_i = \begin{cases} \sum t_i & \text{first and last packets selected} \\ \sum t_i - \Delta & \text{else} \end{cases}$$

For first and last packets are selected, with  $\frac{\sum_{i=0}^N t_i}{N} = X \Rightarrow \sum_{i=0}^N t_i = NX$ ,

$$X' = \frac{\sum^K \sum^s t^i}{K} = \frac{\sum_{i=0}^N t_i}{K} = \frac{NX}{K}$$

Hence,  $X = \frac{K}{N}X'$ ,  $w \cdot X + b = y \Rightarrow w \cdot \frac{K}{N}X' + b = y$ , the classification function for  $X'$  is  $w' \cdot X' + b = y$  with parameter  $w' = \frac{Kw}{N}$ .

For the case either first or last packet is not selected,  $\Delta$  is the difference caused by excluding the sum arrival time of first or last packets:

$$\Delta = \sum_{i=1}^{n-k} t_i$$

$$X' = \frac{\sum_{i=0}^K t_i}{K} - \frac{\Delta}{K}$$

$$= \frac{NX}{K} - \frac{\Delta}{K}$$

$$E[t''] = \frac{\sum_{i=1}^{n-k} t_i}{N-K} = \frac{\Delta}{N-K}$$

$$\Delta = (N-K)E[t'']$$

$$X' = \frac{N}{K}X - \frac{(N-K)E[t'']}{K}$$

Again assuming  $K$  packets selected over  $N$  packets is a binomial process with rate  $q$ . and the interval time of packet has a distribution function  $\phi(T)$  corresponding to time  $T$ , where the  $N$  packets are all generated within time  $T$ . Hence, we have  $E[t'] = \sum_{i=1}^k q\phi(T)$  and  $E[t''] = \sum_{i=1}^{n-k} (1-q)\phi(T)$ .

$$\frac{E[t'']}{E[t']} = \frac{\sum_{i=1}^{N-K} (1-q)\phi(T)}{\sum_{i=1}^k q\phi(T)}$$

$$\frac{E[t'']}{E[t']} = \frac{(N-K)(1-q)}{Kq}$$

$$E[t''] = \frac{(N-K)(1-q)}{Kq} \cdot E[t']$$

Combining the above equation to  $X' = E[t'] = X - E[t'']$ ,

$$E[t'] = \frac{N}{K}X - \frac{(N-K)^2(1-q)}{K^2q} \cdot E[t']$$

$$X' = \frac{N}{K}X - \frac{(N-K)^2(1-q)}{K^2q} \cdot X'$$

$$\frac{N}{K}X = \left(1 + \frac{(N-K)^2(1-q)}{K^2q}\right) X'$$

$$X = \left(\frac{K}{N} + \frac{(N-K)^2(1-q)}{KNq}\right) X'$$

Apply the above equation to the classification function  $w \cdot X + b = y$ ,

$$w \cdot X + b = w \cdot \left(\frac{K}{N} + \frac{(N-K)^2(1-q)}{KNq}\right) X' + b$$

Hence the classification function for  $X'$  is  $w' \cdot X' + b = y$  with parameter

$$w' = w \left(\frac{K}{N} + \frac{(N-K)^2(1-q)}{KNq}\right).$$



### 3.3 Training Traffic Classifiers on arbitrary packet sets

This section describes our proposed traffic classifier, which is constructed based on learning the statistics on arbitrary sets of packets in traffic flows. The intuition is, as the statistics of a sequence of packets can be learned by using some ML algorithms, that of the arbitrary subset of packets should be also learn-able. The proposed classifier using arbitrarily packets provides an auxiliary function to the existing traffic classification systems, addressing the issues that when the existing classifiers cannot classify traffic due to some packet losses and retransmissions in flows. In particular, it could be adopted to address the issue that an evasion technique is applied to evade the classification.

#### 3.3.1 The Orders of Packets in Flows

We apply the 5-tuples definition of flow to group packets: a flow consists of packets with identical transport protocol, source IP, destination IP, source port and destination port. In this work, flows from two directions (upstream and downstream) are grouped as one connection, the classification is operated based on connections. Packets are selected based on their absolute orders in one direction, which can be known by reading the sequence numbers in the TCP headers. Note that, the orders are identified by the packet headers, but not the arrival orders of packets arriving at a NIC card. To precisely locate the packet orders in a flow, the initial sequence number (ISN) specified by the first packet is required. The corresponding implementation is based on tcptrace (Ostermann, 2003) to process the information from packet headers.

#### 3.3.2 Feature Set Design

Constructing the features of traffic flows is an essential task in this work. The features used in our classifier are subject to the condition that the packets in flows are arbitrarily selected. First, features related to inter-arrive time (IAT) are not used in this work, as the time interval would be varied significantly depending on the selected packets. For example, the inter-arrival time could be enlarged if the two adjacent selected packets are the first packet and the fifth packet. Second, features related to packet payload size are not used, as not every packet has payload content, whilst the arbitrarily packet selection cannot guarantee only the packets with payload to be selected. In our classification system some packets without payload could be involved, such packets may contain less information though they are useful

Table 3.1 The feature set.

Name	Description
mean1, mean2	mean of PL of the selected packets on up, down stream
std1, std2	STD of PL of the selected packets on up, down stream
md1, md2	median of PL of the selected packet on up, down stream
mi1, mi2	mean of IPL of the selected packets on up, down stream
si1, si2	STD of IPL of the selected packets on up, down stream

for discriminative purpose, as the applications with payloads and without payloads on some specific packets can differentiate from each others. Third, the features that require a specific packet are not used, for example, Lim et al. (2010) suggests the packet payload of third packet in a TCP flow offers important information for traffic classification, but this feature will not fit with our constraint that packets are arbitrarily selected.

In this work, we only consider the features that relate to packet length, which can be separated into two dimensions, packet length (PL) and inter-packet length (IPL – the length difference between two adjacent selected packets). Not like IAT, the time difference will be enlarged as the interval between the selected packets is enlarged, the length difference between packets is relatively robust. Table 4.1 describes our chosen feature set. To be robust to the condition that some packets are used and some packets are unused, our features are generally computed based on the mean, median and standard deviation (STD) of the selected packets' properties, the value of which would not be strongly dependent on the presence of one or two packets.

### 3.3.3 STD of Packet Length and Inter-packet Length

To compute the STD value for PL, the classifier would need to store each of the packet length till the last selected packet arrives before it can compute the STD value. Moreover, if the last selected packet arrive slowly, then this procedure could be also time-consuming for real time traffic classification. Nowadays network can receive thousands concurrent flows per second, where classification is just one of the task in network management, particularly for those network architecture using commodity machines, it is necessary to use a more effective and scalable way to compute feature sets. In this work, we apply the method of computing STD from Knuth (1997). The essential property of the method is its recurrence formula:

$$M_1 = x_1,$$

$$M_k = \frac{M_{k-1} + (x_k - M_{k-1})}{k} \quad (3.2)$$

$$S_1 = 0,$$

$$std_k = std_{k-1} + (x_k - M_{k-1}) \times (x_k - M_k) \quad (3.3)$$

In the above equations,  $x_k$  is the PL value or IPL value of the  $k^{th}$  packet. The recurrence formulas offers two benefits: First, the STD can be computed immediately once the last selected packet arrived, as most of the computation are done in the previous steps; Second, it does not need to store all the values for each selected packets, only the mean value and std value need to be stored. For IPL, one more value to be stored is the previous packet length so that the length difference can be computed.

### 3.3.4 C4.5 Decision Tree

We select decision tree as the traffic classifier in this work, this selection is based on the suggestion from Kim et al. (2008) and Lim et al. (2010) that decision tree is generally the most fast and accurate ML based traffic classifier. C4.5 developed in Quinlan (1993) is applied as the training algorithm to build decision tree. The data quality is essential in training decision tree. Not enough representative instances or too much noisy instances in the training set can cause a decision tree to learn classification rules that are tied to the training instances but not fit the overall instances, an issue known as over-fitting. As we can see in the later experiments, the arbitrary usage of packets indeed decreases the data quality for training classifiers. There are mainly two reasons: first, some training flows have relatively small number of packets, as a result those flows are not used in training eventually, the size of training set is reduced. Second, the arbitrary usage of packets likely introduces more randomness into the mean value and STD value on those selected packets, making their properties more difficult to be learned. Post-pruning tree is the common approach used to address over-fitting in decision tree. This work applies the pessimistic pruning (Quinlan, 1993) as the post pruning method, which refines a tree model by removing some

under-performing subtrees. Given a tree  $T$ , a candidate pruned tree  $T'$ , pessimistic pruning removes a subtree  $t$  according to the following condition:

$$Er(T', S) \leq Er(T, S) + \sqrt{\frac{Er(T, S) \times (1 - Er(T, S))}{|S|}} \quad (3.4)$$

$Er(T, S)$  estimates the error rate of tree  $T$  on  $S$ ,  $S$  is the set of training instances that reach the subtree  $t$ .  $Er(T, S)$  estimates the error using continuity correction:

$$Er(T, S) = \frac{|S'|}{|S|} + \frac{|T.Leaves|}{2|S|} \quad (3.5)$$

$S'$  denotes the set of misclassified instances in all the leaves of the subtree  $t$ . If the equation 3.4 is met, subtree  $t$  will be removed, as suggested by Quinlan (1993) that a node should be pruned if its error rate is within one standard error from a reference tree.

## 3.4 Experimental Evaluation

In these experiments we compare the classifiers trained based on three different ways of using packets of flows. The study in Lim et al. (2010) showed that the sizes of the first five consecutive packets in a TCP flow offer great discriminative power for traffic classification. We do not assume the classifiers using arbitrary packet sets outperform the classifier using first five packets, likewise we do not assume every packet sets are equally effective for training classifiers, though it is interesting to know the difference in performance between these classifiers, and it is also valuable to know which are the good packet sets for training traffic classifiers.

### 3.4.1 Evaluation Methodology

The classifiers compared here were trained using three different ways of using packets, namely the method using first five packets, the method using subflow and the method using arbitrary packet sets. As the first few packets can be viewed as the first subflow of a flow and a subflow could be a case of arbitrarily selected packets, to be precise on the evaluation, we strictly separated all these cases. The arbitrary packet sets were randomly selected while excluding the cases with more than 4 consecutive packets in a set. Subflow used all the

Table 3.2 Traffic Traces.

Application	Number of Flows	Contributor
HTTP	35000	WIDE
HTTPS	4000	WIDE
Bittorrent	6000	UNIBS
Edonkey	13000	UNIBS
MSN	10000	TSTAT

combinations of consecutive packets excluding the subflow that starts with the first and second packet.

Testing every combination of packets could be exhaustive, in this study we limited our selection on any 5 packets within the first 15 packets, as packets arrive later than which may delay the classification process. A classifier trained on a specific packet selection is subsequently tested on flows processed using the same packet selection. For example, a classifier trained on the packet set {2,4,6,8,10} will be tested on flows with features generated based on the packet set {2,4,6,8,10}.

We combined the traffic traces from three different locations to construct our experimental data. We applied the trace collected from WIDE project (Cho et al., 2000) in 2012 to produce the http and https flows. These flows were then combined with the flows of bittorrent, edonkey and MSN applications, collected from UNIBS (Gringoli et al., 2009) and TSTAT (Finamore et al., 2011), in order to form a traffic dataset with multiple application classes. Table 4.2 presents the number of flows we have in each applications and their contributors. About 70% of the flows were reserved for training the classifiers and 30% were left for testing the classifiers. All the traffic flows are TCP flows, in which packets are accessible in both upstream and downstream directions. We are aware of the issue caused by asymmetric routing (Crotti et al., 2009) in traffic classification, yet in this study we assume packets of both directions are accessible.

**Ground Truth.** WIDE trace is an anonymized trace where data payloads are eliminated. For this trace, we only use the flows of http or https, which are firstly identified by using port match method. Afterwards we apply some filtering rules to filter out the samples that are not likely belonging to http or https. We consider http and https are fundamentally web traffic, thus the TCP connections that have no payload on downstream direction are filtered, as the packets from the server sides are expected to have payload. For traffic flows from UNIBS and TSTAT, the ground truth is provided.

Table 3.3 The overall classification accuracy.

Method	Testcases	Ave Accuracy	Best	Worst
First 5 Pkts	1	0.96	0.96	0.96
Subflow	9	0.91	0.96	0.87
Arbitrary 5 Pkts	30	0.92	0.95	0.87
First5@arbitrary	30	0.49	0.87	0.16

### 3.4.2 Experimental Results

Table 3.3 presents the experimental results for the classifiers trained on three different ways of using packets. The accuracy of classifiers using arbitrary packet sets and subflow are respectively averaged over their testcases. Our results confirm the finding in (Lim et al., 2010) that the first five packets contain important discriminative power on classifying traffic, indeed the classifier using first five packets produces the highest accuracy 96% among all the comparing classifiers in our experiments. For the best cases, classifiers trained using subflow and arbitrary packet sets produce very comparable accuracy to the classifier trained using first five packets, which respectively yields 94% and 95% classification accuracy. (Nguyen and Armitage, 2006) found that not every subflows equally effective for traffic classification, similarly in our experiments not every sets of arbitrary packets are equally effective on classifying traffic flows. The variations in accuracy among the classifiers trained using different combinations of packets are quite significant, in worse case, a classifier trained using an arbitrary packet set only produces 87% accuracy. For the classifier using first five packets, we have an additional experiment to test it over the 30 test sets that contain the flows of arbitrarily selecting packets, as expected, the corresponding accuracy is heavily reduced, as shown at the last row in Table 3.3.

We apply recall and precision to measure the per-application performance of the classifiers, where  $\text{recall} = \frac{\text{TruePositives}}{\text{TruePositives} + \text{FalseNegatives}}$  and  $\text{precision} = \frac{\text{TruePositives}}{\text{TruePositives} + \text{FalsePositives}}$ . Fig. 4.4 and Fig. 3.4 respectively show the recall and precision made by the comparing classifiers. All the classifiers have a good performance on identifying edonkey and msn traffic, whilst classifiers using subflows and arbitrary packet sets have relatively less-accurate performance on the bittorrent traffic. By examining the results in details for classifiers trained on arbitrary packets, we found that there is a big variation in recall between each classifier constructed by using different packet sets, there is a 20% difference in recall between the best case and worst case.



Fig. 3.3 Recall on Traffic. F5 denotes first 5 packets; S5 denotes subflow 5 packets; R5 denotes arbitrary 5 packets.

As we have seen classifiers trained on different arbitrary packet sets yielded different performance, particularly with big variation on identifying bittorrent. We then assigned the classifiers using arbitrary packet sets into three different classes – good, middle and bad, according to their accuracy and recall on the bittorrent traffic, some classifiers are labeled as bad even though their accuracy is not bad if they cannot achieve good recall on bittorrent. Table 3.4 presents their accuracy and the corresponding number of test cases, and Fig. 4.5 and Fig. 3.6 present their per-application recall and precision. Although the bad cases produced poor performance on bittorrent, we find out that in the good cases the classifiers using arbitrary packet sets could produce comparable accuracy to the classifier using first five packets. Here we list the packet sets yielding good classifiers in our experiments: they are packets of {1,2,3,7,8}, {1,3,5,7,9}, {2,3,4,5,10}, {2,3,4,9,10}, {2,4,6,8,10} and {3,4,7,8,12}.

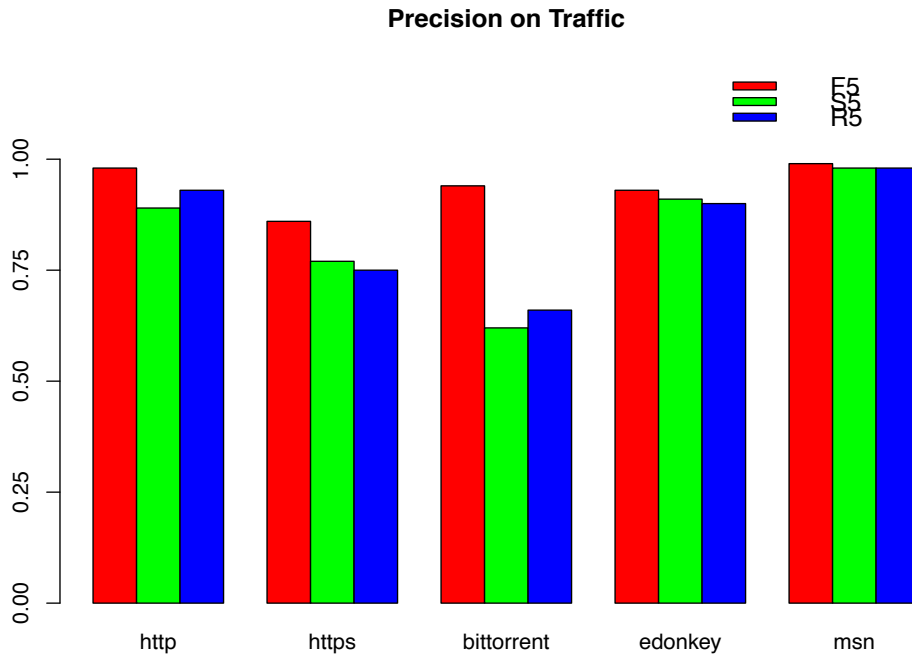


Fig. 3.4 Precision on Traffic. F5 denotes first 5 packets; S5 denotes subflow 5 packets; R5 denotes arbitrary 5 packets.

Table 3.4 The classification accuracy on three classes.

class	number of cases	Ave Accuracy
good	6	0.94
middle	13	0.92
bad	11	0.90

### 3.5 Summary

With the concern that packets losses and retransmissions are common in practical networking environment, our proposed classifier using arbitrary packet sets can assist the classifier using strictly first five packets to classify the flows with missing packets. To effectively deploy the corresponding classification system, a network operator needs to provide multiple traffic classifiers using arbitrary sets to ensure at least one of the proposed classifier can deal with flows with missing packets. Moreover, the proposed classifier can be adopted to mitigate the issue that payload mutation techniques are used by some malicious applications to evade classification. Experiments show that although not every packet sets





Fig. 3.5 Recall on three different classes.

are equally effective on training accurate classifiers, with using some good packet sets, the proposed traffic classifier yields comparable accuracy to the classifier using first five packets.

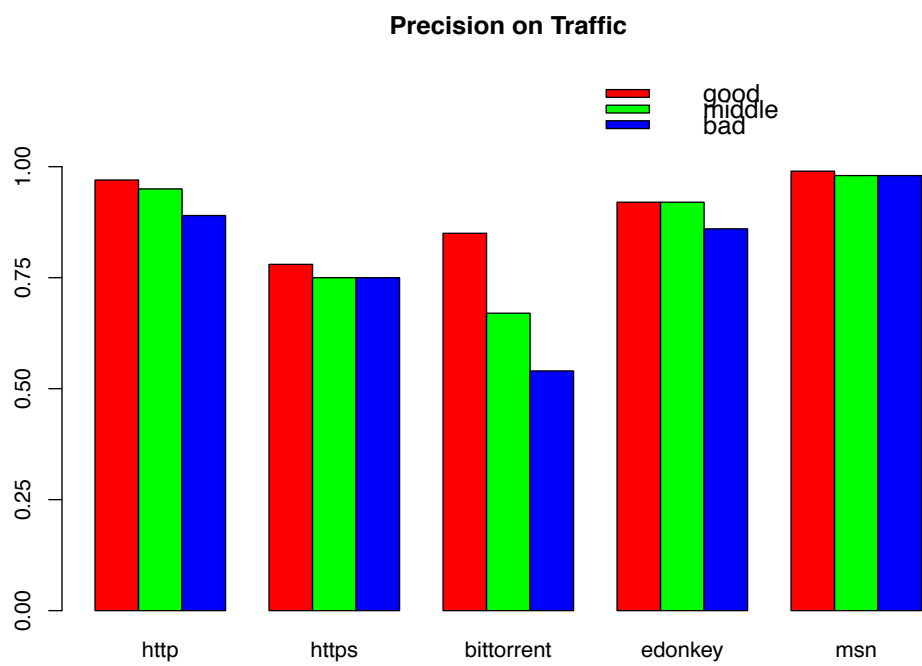


Fig. 3.6 Precision on three different classes.

# Chapter 4

## Ensemble classifier for traffic in presence of changing distributions

### 4.1 Introduction

The accuracy of analysis based traffic classification is reliant on the statistics learned on flow traffic, however flow statistics can be changed over the mid to long term as the mix of applications across the network changes and user demands for them vary. When the new distribution of data diverges from those data with which the classifiers were trained, the ML based traffic classifiers trained from previous traffic mix suffer loss of accuracy as a consequence. We term this issue as changing distribution and illustrate the issue in §4.2, showing how traffic data over time can be viewed as being drawn from varying statistical distributions. This issue is of practical concern, given the fact that some characteristics of Internet applications (e.g., P2P) are intentionally modified to work around the traffic classifiers in order to gain unfair advantage over other traffic types.

Updating a traditional traffic classifier is time consuming (Callado et al., 2009), while the literature lacks explicit recommendations on how often a classifier should be updated or when a new classifier would be needed. Two existing methods used in practice come with two different shortcomings; 1) Training a new classifier only on new traffic leads to some learned knowledge lost along with the abandoned data and classifiers; 2) Training a classifier on a large dataset that combines all data collected in different periods leads to performance issues. Map-Reduce can be applied to speed up training classifiers on large traffic datasets (Lee and Lee, 2012), nevertheless it requires more computation resources.

Furthermore, if a particular period has significant large data volume compared to other periods, it will dominate the analysis of the traffic. To avoid such situations the representative samples need to be carefully selected from different periods to construct the combined dataset.

This chapter develop an adjustable traffic classification system, where the update to classifiers is effectively implemented by using ensemble classification. The ensemble technique utilizes the previous trained classifiers, thus the corresponding learned information is not lost whilst the previous data are not reprocessed. To update the classifier in time, we develop a method to detect traffic change. We show that change in traffic can be detected by measuring the statistical difference between the previous traffic samples and the new traffic samples from the prevailing traffic classes. By using the ensemble classification and the change detection method, the proposed system is more sensitive to change and produces consistent accuracy for traffic classification.

The rest of this chapter is structured as follows. §4.2 illustrates the issue of changing distributions and proposes a method to detect change in traffic distribution. §4.3 presents the proposed adjustable traffic classification system. §4.4 describes a simulation of traffic in presence of new patterns, where three methods are compared. A conclusion is drawn in §4.5.

## 4.2 Changing Traffic Distributions

A well trained traffic classifier can become less-accurate over time if the traffic's statistical distribution changes. In this section we illustrate the issue from the probability aspect. Using probability based classification, a traffic classifier is constructed based on computing how likely a given flow, described by a vector of features  $X=(x_1, x_2, \dots, x_n)$ , belongs to a class  $Y$ . As  $X$  is known, the classification is dependent on the posterior probability  $P(Y|X)$ , the probability of a flow in class  $Y$ , having a feature vector  $X$ . The classification decision is assigning a  $X$  with the  $Y$  that maximizes the posterior probability  $P(Y|X)$ . As we know  $P(Y|X)=\frac{P(XY)}{P(X)}=\frac{P(Y)P(X|Y)}{P(X)}$ . Assuming each feature  $x_i$  is independent, we have  $P(X|Y) = \prod_{i=0}^n P(x_i|Y)$ . A classifier can be defined as a mapping function  $F : X \rightarrow Y$ , from flow  $X$  to class  $Y$ , which can be written as:

$$F(X) = \arg \max_Y P(Y|X)$$

$$\begin{aligned}
&= \arg \max_Y \frac{P(Y)P(X|Y)}{P(X)} \\
&= \arg \max_Y \frac{P(Y) \prod_{i=0}^n P(x_i|Y)}{P(X)} \tag{4.1}
\end{aligned}$$

In Equation 4.1, we can see that any change in  $P(x_i|Y)$  could be propagated into the joint distribution, in other words, any change in the distribution of each feature (e.g., a feature could be packet payload size) may lead to a change in  $P(Y|X)$ , resulting in the corresponding classifier not learning the true distribution. Similarly, change in  $P(Y)$  and  $P(X)$  will also affect the distribution  $P(Y|X)$ , subsequently affecting the classification.  $P(Y)$  measures the fraction of flows that belongs to a class  $Y$  on a given training set.  $P(X)$  measures the probability of a flow taking on a value  $X$ ,  $P(X)=\prod_{i=0}^n P(x_i)$ , where  $P(x_i)$  can be calculated as the probability distribution of  $x_i$  on a given training set.

To illustrate the changing distributions in traffic data, we generated some density plots from the UNIBS (Gringoli et al., 2009) trace, which has three data blocks representing traffic samples collected in three adjacent days, three plots are accordingly generated in Fig. 4.1. We study the mean packet payload size of the P2P flows in this trace, where the P2P flows are generated by two different P2P applications, namely edonkey and bittorrent. Day 3 receives a big jump in the P2P volume, which has approximated 13000 P2P flows, while Day 1 has approximated 5200 P2P flows and Day 2 has only approximated 1000 P2P flows. As shown in Fig. 4.1, Day 3 exhibits some changes in the P2P distribution. Particularly comparing Day 3 to Day 1, Day 3 has relatively larger fraction of flows with mean payload size distributed in the range of  $[0, 100]$ , while relatively smaller fraction of flows have mean payload size larger than 200.

### 4.2.1 Change Detection

If the time when a traffic classifier should be updated can be accurately identified, the classifier would not need to be updated every fixed interval defined by network operators according to their own experiences. In this work, we propose a method to detect change in traffic. We observe that when new traffic patterns appear, most likely they will be randomly classified as some prevailing traffic patterns, depending on whether a new pattern has some similarity to the known patterns. When new patterns grouped into the http class, for instance, it may change the statistical structure of the http data, leading to a shifted distribution of http.

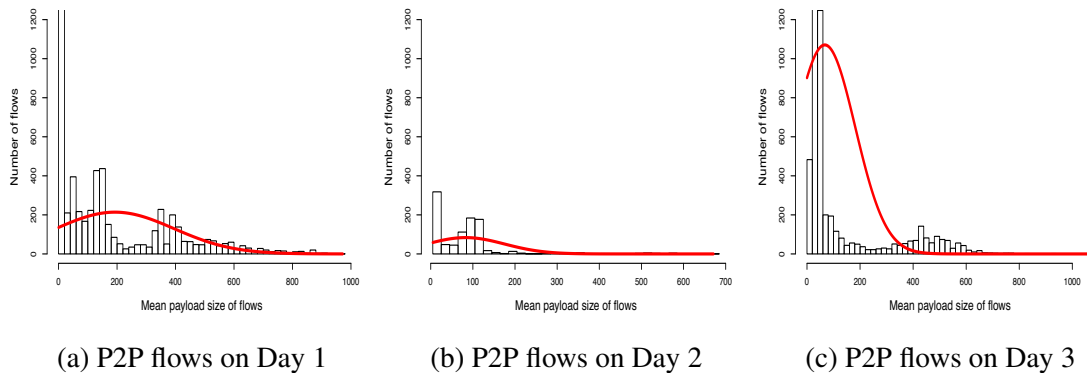


Fig. 4.1 Distributions of P2P traffic over 3 days in the UNIBS trace.

---

**Algorithm 1** Change Detection.

---

```

0: Input: a new flow set  $D$ , an old flow set  $G$ 
1: for each class  $y$  in  $\{\text{http}, \dots, \text{ftp}\}$  do
2:    $d := \{f_i.\text{mean}(\text{payloads}), f_i \in D \text{ classified as } y\}$ 
3:    $g := \{f_j.\text{mean}(\text{payloads}), f_j \in G \text{ classified as } y\}$ 
4:    $U = \text{mww.test}(d, g)$ 
5:   if  $U < 0.01$  then
6:     raise “change detected”
7:   end if
8: end for

```

---

We can apply a statistical test between a previous http dataset and the new dataset classified as http, regarding to their features such as mean packet payload size or payload size variance, if test result indicates that the two datasets are not from the same distribution, change may have occurred. Algorithm 1 presents the procedure of change detection.

We apply the Mann Whitney Wilcoxon (MWW) U-test to test the statistical difference in datasets. This method offers two benefits, first it is a non-parametric test that it does not require users to assume the data distributions prior to test, generally for using the parametric tests, data are assumed to be normally distributed; Second, the two comparing datasets are unnecessary to have the same number of instances, this is useful to this work as traffic datasets collected in different periods are mostly different in sizes. A detail description of the MWW U-test can be found in (Mann and Whitney, 1947) and its implementation is applied from R (Ihaka and Gentleman, 1993).

If a traffic classifier has a low precision, a large amount of instances are misclassified as a specific prevailing traffic, the statistical distribution of the prevailing traffic could be

consequently changed, even though there is no new traffic pattern interfering. As a consequence, the change detection method may produce false alarms. To reduce the false alarms, the detection method is suggested to be used with a classifier with good precision. Another potential issue of the proposed method is that it may be subject to temporary anomalous in traffic, it would be very difficult to tell if the statistical difference is caused by temporary anomalous or new patterns. For either case the method raises alarm to the network operators.

### 4.3 The Traffic Classification System

The proposed adjustable traffic classification system consists of two modules — the offline training module and the online classification module, which respectively provides the functions to train and maintain traffic classifier offline in the network management overlay and the functions to operate it online in the production network. The online module is suggested to be installed at edge networks or some appropriate positions that enable the classifier to monitor packets from both ingress and outgress directions. In a more complex case, the positioning would suffer the issue of asymmetric routing (Crotti et al., 2009), in which case the traffic classifier needs to be further adapted.

Fig. 4.2 presents the overall architecture. The online module consists of three components: flow identifier, feature generator and traffic classifier. Packets are grouped into flows by flow identifier using 5-tuple {transport protocol, source IP, destination IP, source port, destination port}. The offline module also consists of three components: a sample storage, measurement and ensemble training. The sample storage stores the traffic flows captured online. Every few hours (or few days) the offline module measures the captured flows in order to detect if change occurred. If it did, the offline module will update the existing ensemble classifier and then upload the updating configuration to the online modules. The updating configuration is a classifier and a set of weights, which is related to ensemble classifier that will be presented below.

#### 4.3.1 Ensemble Traffic Classifier

In this work we apply ensemble technique to construct a traffic classifier. An ensemble classifier is a group of classifiers with their corresponding weights. Equation 4.2 presents the logical structure of an ensemble classifier. To avoid ambiguity, the classifiers composing the

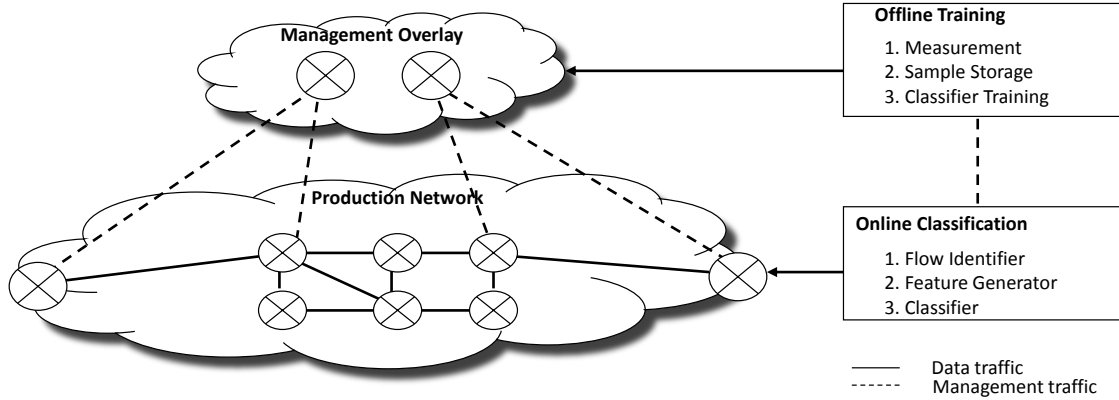


Fig. 4.2 The Offline and Online Modules of the adjustable traffic classification system.

ensemble are defined as **base classifiers**. Each base classifier  $F_i(X)$  has a weight  $w_i$ , ensemble classification is made based on the weighted majority voting given by the base classifiers, as shown in the Equation 4.3.

$$X \rightarrow \left\{ \begin{array}{l} w_1 \cdot F_1(X) = Y_1 \\ \cdot \\ \cdot \\ w_n \cdot F_n(X) = Y_n \end{array} \right\} \rightarrow \arg \max_Y \left( \sum_{Y=Y_i} w_i Y_i \right) \quad (4.2)$$

$$E(X) = \arg \max_Y \left( \sum w_i F_i(X) \right) \quad (4.3)$$

There are two benefits on using ensemble classification. First, it allows to incrementally train or update the traffic classifier, and each update is performed on the most recent dataset only. As it is unnecessary to train a new classifier on a large dataset that combines data from previous phases, the training time is reduced. Second, the training process of ensemble classifier can avoid the situation that a phase with large data volume dominates the training process. In general, the most recent phase should be treated as the most important phase. However, traditional method combines data from all previous phases to construct a single classifier, where the phase with very large data size can dominate the analysis of the combined dataset, resulting in the phase with very large data volume instead of the most



recent one becomes the most influential phase in training a traffic classifier. In contrast, ensemble method trains base classifiers individually on different phases and dynamically computes weights for different base classifiers, thus their influences can be controlled. A theoretical proof is given in (Wang et al., 2003) to show why ensemble classifier can outperform a single classifier in terms of accuracy.

The structure of ensemble classifier allows it to be updated with new knowledge through combining new base classifiers into its ensemble. Base classifiers trained in different phases give complementary knowledge to each other, thus the ensemble classifier has a complete knowledge. To obtain an accurate ensemble classifier, the task accordingly becomes how to learn a good combinations of 1) base classifiers and 2) their corresponding weights, which together compose a good ensemble traffic classifier.

### 4.3.2 Feature Set & Base Classifier

Decision tree is applied as the base classifier in this work, as suggested in (Lim et al., 2010) and (Kim et al., 2008), decision tree is very discriminative on classifying Internet traffic. Table 4.1 lists the chosen features that describe the traffic flows, where the decision tree training algorithm J48 (Hall et al., 2009) is applied to train the traffic classifier. TCP flows from two directions (upstream and downstream) are grouped as one connection if they have the same tuples, the classification is operated based on connections. Since flow directions is an important information on classifying TCP traffic (Lim et al., 2010), packets from two different directions of a connection are processed respectively. Furthermore, we separate two situations: 1) for no packets on a direction, like UDP connections only have packets on one direction, -1 are set to the corresponding features of the no-packets direction (e.g.,  $\text{mean}_2 = -1$ ); 2) for packets but zero payload on a direction, like TCP acknowledge traffic, 0 are used (e.g.,  $\text{mean}_2 = 0$ ). By doing that, we do not need an additional feature to explicitly specify a connection is of TCP or UDP. Features such as duration and number of PUSH packets, which are only available when flows are terminated, are not used. Although eight features are used in total, the only information needed from the packet headers is data payload size (PS) and flow directions.

Note that features related to inter-arrival time are not used as they are subject to transient variations caused by network congestion, link failures and other network conditions. This design may help mitigating the potential issue caused by temporary anomalous in traffic, particularly for the change detection method.

Table 4.1 The feature set. PS denotes Payload Size.

Name	Description
mean1, mean2	mean PS of first 5 packets on upstream, down stream
var1, var2	PS variance of first 5 packets on upstream, down stream
max1, max2	max value of PS in first 5 packets on upstream, down stream
packet3	PS of the 3rd packet on upstream, down stream
packet2	PS of the 2nd packet on upstream, down stream

### 4.3.3 Weight Computation

After getting the base classifiers, the next step is to compute the weights for them. In this work we apply the training algorithm Accuracy Weighted Ensemble (AWE) from (Wang et al., 2003). In AWE the weight is computed reversely proportional to the expected error of a base classifier, namely, the base classifier that has smaller expected error has higher weight, thus higher influence in the ensemble classification. Given a classifier  $t$  that has an estimation  $f_t(X)$ , the expected error of  $t$  on a dataset  $D$  can be calculated as:

$$Err(t) = \frac{1}{|D|} \sum_{X \in D} (1 - f_t(X))^2$$

$f_t(X)$  is defined to estimate the classification made by classifier  $t := F_t(X)$  such that  $f_t(X) = P(Y = F_t(X) | X)$ .

Suppose that we have a random guess — the chance of  $X$  being classified as  $Y$  equals to  $Y$ 's probability distribution  $P(Y)$  over the entire class space  $C := \{\text{all classes}\}$ . A random classifier classifying instances in such way has the expected error as follows:

$$Guess = \sum_{Y \in C} P(Y)(1 - P(Y))^2$$

For example, a binary class {head,tail} with uniform class distribution, the random guess on this classification task will have 0.25 expected error. Note that for a determined dataset, the guess is a fixed value. AWE guarantees to select base classifiers perform at least better than a random guess, the weight  $w(t)$  of a base classifier  $t$  is calculated as follows:

$$w(t) = \begin{cases} Guess - Err(t) & \text{if } Guess - Err(t) > 0 \\ 0 & \text{else} \end{cases} \quad (4.4)$$

---

**Algorithm 2** Accuracy Weighted Ensemble Algorithm.

---

- **Input:** a dataset  $D = \{(X_1, Y_1), \dots, (X_n, Y_n)\}$ , a set of classifiers  $T = \{t_1, \dots, t_n\}$
- **Output:** an ensemble classifier  $E$

- 1: Initialize a vector  $W$
- 2: build classifier  $m$  on  $D$
- 3: compute  $w(m)$  on  $D$  using eq 4.4
- 4: add  $w(m)$  to  $W$
- 5: **for** classifier  $t_i \in T$  **do**
- 6:   compute  $w(t_i)$  based on  $D$
- 7:   add  $w(t_i)$  to  $W$
- 8: **end for**
- 9: sort  $W$
- 10:  $E = W[1 : k]$ , selecting the best  $K$  classifiers
- 11: return  $E$

---

Equation 4.4 ensures that if the classifier  $t$  performs not better than a random guess, its weight will be set to 0, making it not to be used in ensemble. Also this setup ensures that if error is bigger, the weight would be smaller.

### 4.3.4 Ensemble Training

As discussed before, we prefer that the dataset collected in most recent phase to have most influence in training an ensemble classifier. To implement this requirement, we train a base classifier on a new dataset and evaluate all the base classifiers based on the new dataset, thereby the new base classifier would have relatively better accuracy, in turns it has higher weight and more influence in the majority voting.

Algorithm 2 presents the procedure of constructing an ensemble traffic classifier based on a new dataset. The inputs are a collection of base classifiers and a new dataset. Line 3 and 4 describe that a new base classifier is generated on the new dataset and the corresponding weight is calculated. In the for loop, each previous learned base classifier is evaluated to gain a weight using the Equation 4.4. Finally, the base classifiers are sorted according to their weights and the top  $K$  classifiers are selected to construct the final ensemble, as shown in line 9 and 10.

## 4.4 Simulation on New Patterns Interference

This experiment aims to compare the performance of three different classifiers trained using three different methods. The comparing classifiers are tested on a simulated distribution-changing traffic dataset, which consists of 9 data blocks. The first data block provides the prevailing traffic while in some of the subsequent data blocks new traffic patterns are intentionally injected to simulate the changing distribution. This section presents the details of the data, experimental setup and the results.

### 4.4.1 Experimental Setup

A challenge in this experiment is that no existing public traffic trace is provided regarding the problem of changing distributions. Furthermore, most trace repositories provide traces without associated ground truth of the traffic classes. To simulate the changing traffic distributions, first we use the 63 hours long trace, collected from WIDE project (Cho et al., 2000) in 2012, to provide the base traffic such as http, ftp, email and so on, and then we use the skype, P2P, MSN, IPTV traffic, collected from UNIBS (Gringoli et al., 2009) and TSTAT (Finamore et al., 2011; Mellia and Meo, 2010), as the interfering traffic to simulate the scenario that new applications emerge into network traffic. Table 4.2 presents the involved applications, their corresponding amount of flows and contributors. For WIDE trace, we selected 9 different time periods to generate 9 data blocks. In some of these blocks, we injected some new traffic patterns to simulate the changing distributions. Table 4.3 presents the blocks injected with new traffic. In each data block, 70% flows were reserved for training and 30% for testing.

**Ground Truth.** WIDE is an anonymized trace where data payloads are eliminated. In order to retrieve the ground truth of the traffic class, first we classify the flows based on their port numbers, then we apply some filtering rules to filter out the suspicious traffic patterns, which are not used in the later experiments. For http, TCP connections that the packets from the server-side having zero payloads are filtered out, as we consider http are fundamentally web application or client-server based applications where server-side packets are expected to have content. For mail or FTP traffic, only the connections that have zero payloads on both directions are filtered, in which case no analysis can be done. Since some of these applications can keep connecting to the servers by using keep alive protocol, it is reasonable that they have a lot of control packets with zero payload. Some flows that have too small

Table 4.2 Traffic Traces.

Application	Number of Flows	Contributor
HTTP	65000	WIDE
HTTPS	8000	WIDE
SSH	7000	WIDE
FTP	3000	WIDE
SMTP	2500	WIDE
IMAP	2500	WIDE
Skype	1200	UNIBS
Bittorrent	6000	UNIBS
Edonkey	13000	UNIBS
MSN	37000	TSTAT
IPTV	1100	TSTAT

Table 4.3 The data blocks injected with new traffic.

Block	#2	#4	#5	#7	#8
New Type	Skype	Bittorrent	Edonkey	MSN	IPTV

numbers of packets are not used, most of which are TCP flows terminated without receiving the FIN packets, appropriate analysis cannot be operated on those short flows. For the other traces collected from UNIBS and TSTAT, the ground truth is provided along with the data.

An assumption in this work is that all the training data are provided with known classes thus supervised learning method can be applied directly. Collecting the ground truth of class is a complex task; some studies has been proposed to produce the ground truth for traffic classification (Gringoli et al., 2009)(Szabó et al., 2008).

#### 4.4.2 Experimental Results

We evaluated three types of classifiers, namely: 1) the decision tree classifier (denoted as *tree* in Fig. 4.3), trained on the first data block and never updated; 2) the combining classifier (denoted as *comb* in Fig. 4.3), updated every time by being re-trained on a dataset that combines the new data with old data. 3) the ensemble classifier (denoted as *awe* in Fig. 4.3), updated only based on new data. All the classifiers are implemented using the weka data mining package (Hall et al., 2009).

Fig. 4.3 presents the accuracies of the comparing classifiers in this simulation. As we can see the decision tree classifier is heavily affected by the injecting traffic, whereas ensemble

classifier maintains good accuracy about 95% over the 9 testing blocks, and the combining classifier produces similar performance. One interesting observation within this simulation is that with using combining classifier, the misclassified edonkey flows tend to be recognized as http or imap rather than bittorrent, though edonkey and bittorrent are in the same P2P application category. Note that the experiment assumes the combining method always know traffic change, which rarely occurs in practice, thus the experimental accuracy of the combining method is optimized.

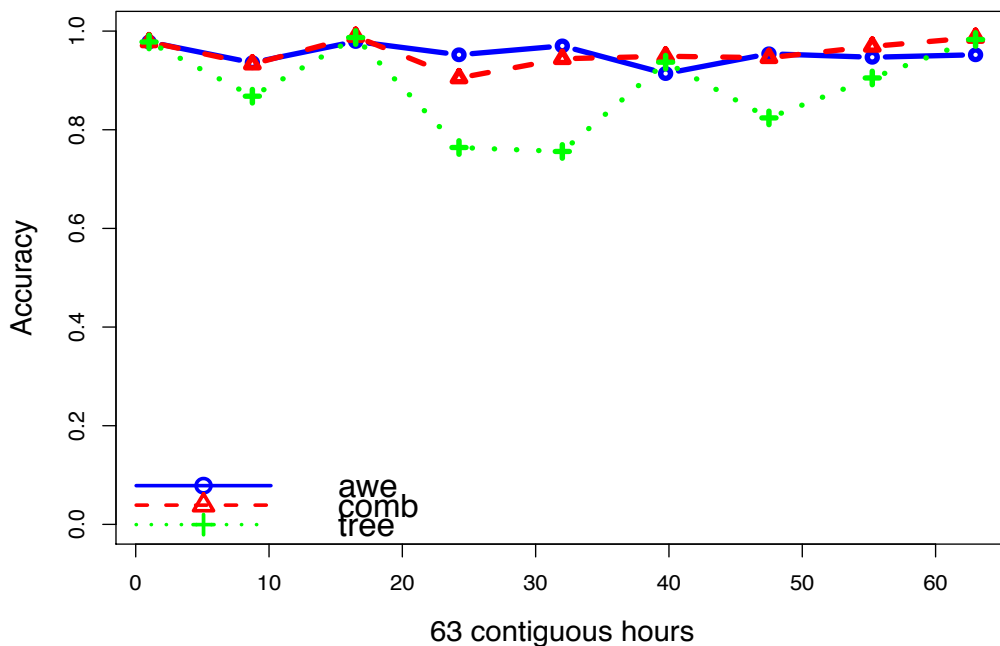


Fig. 4.3 Classification accuracy in simulation of changing distributions.

We apply recall and precision to measure the per-application performance of the classifiers,  $\text{recall} = \frac{\{\text{TruePositives}\}}{\{\text{TruePositives} + \text{FalseNegatives}\}}$  and  $\text{precision} = \frac{\{\text{TruePositives}\}}{\{\text{TruePositives} + \text{FalsePositives}\}}$ . Fig. 4.4 compares the recall of the ensemble classifier and the combining classifier on the injecting traffic; the decision tree classifier is not compared as it cannot recognize any injecting traffic. Fig. 4.4 shows that the comparing classifiers generally have the same performance, except that the ensemble classifier shows better recall on the two P2P traffic patterns. In fact, in the worst case, the combining classifier only has 76% recall on bittorrent, while the ensemble classifier has 99% recall on such traffic.

Fig. 4.5 compares the precision of the ensemble classifier and the combining classifier. In general the ensemble classifier shows better or equal precision to the combining classifier, except that the combining classifier yields better precision on bittorrent. This shows an interesting fact that the ensemble classifier is capable to identify more bittorrent flows, as indicated by the recall, whilst it makes slightly more mistakes (3% more) on classifying flows of other applications as bittorrent, as indicated by the precision. Considering the performance difference in classifying edonkey traffic, regarding both recall and precision, the ensemble classifier in overall performs better on classifying P2P traffic.



Fig. 4.4 Recall on the injecting traffic.

Fig. 4.6 compares the updating time required by the combining method and the ensemble method. As shown the updating time cost by the combining method is strictly greater, which appears to be grow linearly with the flow numbers. In contrast, the updating time of ensemble method is not accumulated, due to no computation overhead being needed to re-process the learned data. Table 4.4 presents the results of our change detection method tested on this simulation, in which a data block is compared to its previous block to examine if changing distribution exists. Block 6 and block 9 are measured to have no changing distribution, as indicated in Fig. 4.6 zero updating time are spent on these blocks by the ensemble method. The detection method miss-detects block 3 to have change, though the

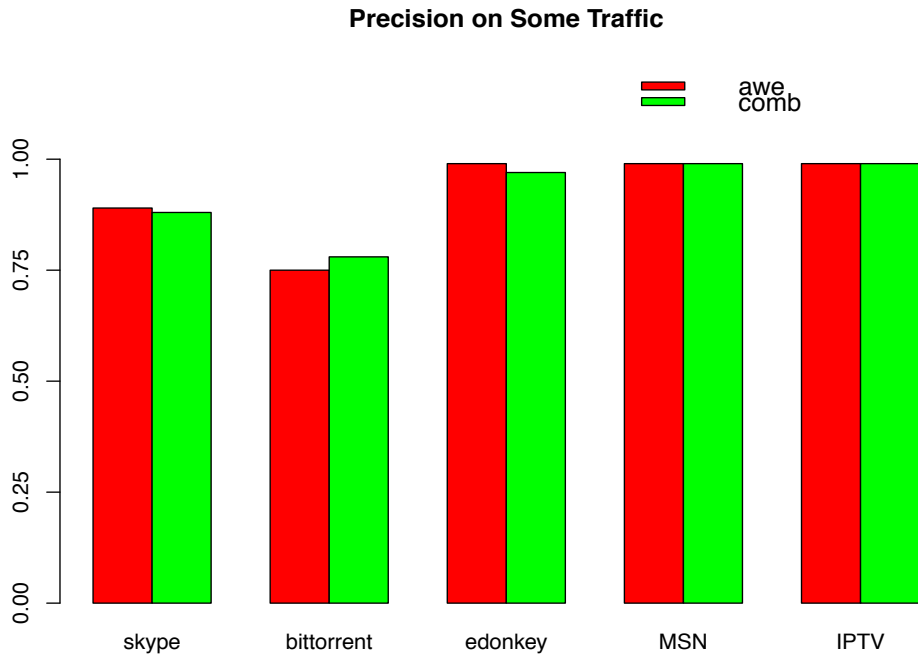


Fig. 4.5 Precision on the injecting traffic.

Table 4.4 Blocks with Change, +: true -: false.

Block	#2	#3	#4	#5	#6	#7	#8	#9
Detect Change	+	+	+	+	-	+	+	-
Fact	+	-	+	+	-	+	+	-

resulting updating time of the ensemble method is still less than that of the combining method.

## 4.5 Summary

The discussion and analysis in this chapter shows static machine learning based traffic classifiers will suffer decreased accuracy as aggregate traffic flow statistics change in line with changing applications. The work in this chapter proposed an adjustable traffic classification system, the key technique of which is ensemble classification and change detection. In experiments, we compared three different traffic classifiers trained using three different methods on a simulated distribution-changing traffic dataset. As expected the



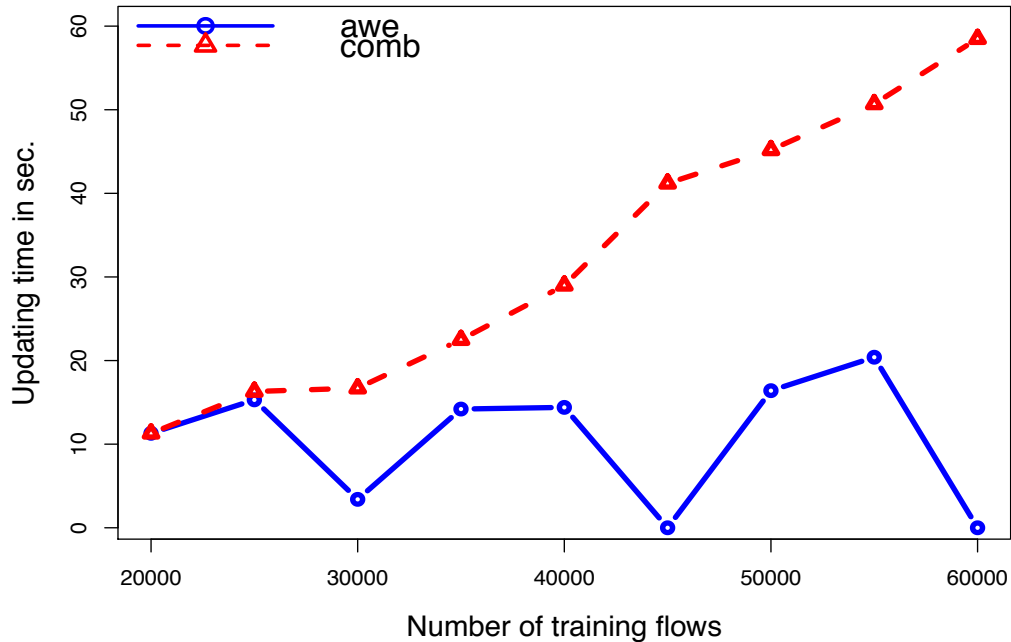


Fig. 4.6 Updating time on growing training flows.

ensemble classifier and the combining classifier outperform the static decision tree classifier in terms of classification accuracy. Comparing the ensemble method with the combining method, the results show that the ensemble classifier produces comparable accuracy to the combining classifier, whilst using much shorter updating time. Meanwhile the ensemble classifier also shows better performance on P2P traffic. Since the experiment assumes the combining method always know traffic change, which rarely occurs in practice, thus the experimental accuracy of the combining method is optimized, nevertheless the proposed method still show benefits over it.

# Chapter 5

## Network Aware VM Placement Using Effective Bandwidth Estimations

### 5.1 Introduction

Previous two chapters explored how to leverage traffic analysis techniques to implement advances traffic classification systems. This chapter and next chapter will study the applications of traffic analysis techniques in datacenter, a fast-growing technological area. We applied traffic analysis techniques aiming to manage the dynamics in datacenter traffic and the varying demands of resource from the applications hosted in datacenters.

As discussed in chapter 2 section 3, where the background of datacenter network management is introduced, performance guarantee is an essential requirement to many mission critical multi-tier applications hosted in datacenters. As discussed in Chapter 5, the prevalent approach in the industry to achieve relatively predictable datacenter networks is to strictly reserve bandwidths for tenants' Virtual Machine (VM) ensembles. The term 'ensemble' here is used to refer to the group of VM instances involved in the delivery of an application's functionality. This is the case, for example, in Amazon EC2 (Amazon Inc., 2006) and Rackspace (Rackspace Inc., 2006) Infrastructure-as-a-Service (IaaS) offerings.

However, strict bandwidth reservation does not efficiently utilize network resources—unused bandwidth is wasted during periods when VM traffic demands are below the provisioned peak rates. Nevertheless, overprovisioning is a pragmatic approach, given that it is difficult for tenants to predict accurately and in advance inter-VM bandwidth requirements for their VM ensembles. Thus, tenants can be provided with weak statistical guarantees that

Quality-of-Service (QoS) targets in their Service Level Agreements (SLAs) will be met. However, VM placement systems relying on overprovisioning typically lack the capability to adjust the VM bandwidth allocations following initial placement.

In response to overprovisioning issues in datacenters, recent work on datacenter network management has focused on allocating network resources to tenants' VM ensembles at optimal cost by implementing 'network-aware' VM placement algorithms and accompanying traffic control mechanisms (LaCurts et al., 2013; Lam et al., 2012; Popa et al., 2013; Wuhib et al., 2015; Xie et al., 2012). Often, these approaches seek to minimize bandwidth wastage by reallocating bandwidth not being used by a VM to other VMs in the same cluster. However, these approaches largely focus on statistically guaranteeing the throughput performance of VMs; they do not address the potential for increased delays due to transient overloads of network links that may occur as a result of a less strict bandwidth allocation regime. This thesis addresses the problem of efficiently utilizing datacenter network resources while ensuring that QoS delay targets specified in SLAs are met. In particular, we seek to support probabilistic QoS targets expressed in the following manner: "no more than 2% of packets should be delayed by more than 50ms."

The proposed approach is to provide a network-aware VM placement scheme in which VMs within an ensemble that need to be placed on different servers are placed in a manner that ensures that the "effective bandwidth" available on the network path between servers is sufficient. Kelly (1996) defined effective bandwidth as the "minimum amount of bandwidth required by a traffic source to maintain specified QoS targets." We require neither *a priori* bandwidth reservation nor the implementation of traffic control mechanisms in server hypervisors. Once a VM ensemble is placed, VMs are enabled to asynchronously reach their peak throughputs. We rely on the use, in the admission decision, of empirically computed estimates of effective bandwidth on the network paths to ensure that the likelihood of SLA violation is minimal. This approach allows the datacenter provider to specify QoS targets for hosted applications in terms of delay, whilst ensuring that network resources are utilized efficiently.

The rest of this chapter is structured as follows. The concept of effective bandwidth is introduced in §5.2. In §5.3 and §5.4 we respectively describe MAPLE system design, and the network-aware VM placement algorithm and its algorithmic variant, MAPLE<sub>x</sub>, that copes with anti-colocation constraints. Experimental evaluation is presented in §5.5, where MAPLE/MAPLE<sub>x</sub> are evaluated and compared to Oktopus (Ballani et al., 2011). Finally §5.6 provides concluding remarks for the chapter.

## 5.2 Effective Bandwidth Review and Residual Bandwidth Estimation

### 5.2.1 Effective Bandwidth and its Estimation

Effective bandwidth is the minimum amount of bandwidth required by a traffic source (for example, a VM or an application) to maintain specified QoS targets. In a communications network, the effective bandwidth of traffic sources depends not only on the sources themselves, but also on the whole set of systems, including link capacity, traffic characteristics, and the QoS target (Courcoubetis et al., 1998). For example, consider the scenario of a link with capacity of 1 Gbps and two VMs that generate traffic with mean throughput of 300 Mbps and peak throughput of 550 Mbps, where the probability of peak throughput is 10% for each VM, and the QoS target specifies that no more than 5% of the traffic suffers delays higher than 50ms. The question that arises is whether the given link can accommodate these two VMs. If the two VMs reach the peak throughput at the same time, the aggregated throughput would be 1100 Mbps, exceeding the total link capacity of 1 Gbps. Assuming there is a shaping policy, the exceeding traffic would be delayed more than 50ms. However, the probability of this situation actually happening is only 1% (assuming VMs are independent). Therefore, the QoS target is not violated, and the link can accommodate the VMs. To allow each VM to asynchronously reach its peak, the allocated bandwidth of each VM should be higher than its mean throughput. Moreover, it is not necessary to allocate peak throughput to the VMs because the chance that they both reach peak is small enough to statistically guarantee that the QoS target is not violated. This example indicates that the effective bandwidth lies somewhere between the source's mean throughput and peak throughput (Kelly, 1996).

Effective bandwidth can be analytically estimated through the application of large deviation theory (Courcoubetis et al., 1998), or via more computationally simple approximations based on fluid-flow and stationary models (Guerin et al., 1991). However, in MAPLE, we employ an empirical approach based on the analysis of traffic traces collected at each server. At set intervals, we estimate the effective bandwidth for a given delay-based QoS target for the aggregated traffic generated by the server in question over the trace duration. This value, plus the peak rate of the VM that is a candidate to be placed on the server, are compared to the available bandwidth so to assess whether the VM can be placed on that server. The

effective bandwidth estimation technique is taken from (Davy et al., 2008) and is summarized as follows.

Let  $delay_{max}$  be the nominal maximum delay and let  $p_{delay}$  be the percentage of traffic that can exhibit delay greater than  $delay_{max}$ . We define the effective bandwidth  $R_{eff}$  of a traffic source for delay QoS target  $(delay_{max}, p_{delay})$  as the minimal rate  $R$  such that if we simulate a FIFO queue with unlimited buffer and processing rate  $R$ , the percentage of traffic that will exhibit delay greater than  $delay_{max}$  will be less than  $p_{delay}$ . To estimate the effective bandwidth of a particular traffic source on the network, we take a recorded packet trace of that source. We observe that if we simulate a FIFO queue (initially assumed to be empty) with the same input traffic trace  $\{T_M\}$  for different process rates  $R_1 > R_2$  and estimate the percentages  $p_1$  and  $p_2$  of traffic delayed more than  $delay_{max}$  for different rates respectively, then  $p_1 \leq p_2$ . This means that the percentage of traffic,  $p$ , delayed more than  $delay_{max}$  is a monotonically decreasing function of processing rate  $R$ . Based on this observation, we employ a simple bisection algorithm for a recorded packet trace to find the minimal value of a queue rate such that the percentage of traffic delayed more than  $delay_{max}$  is less than  $p_{delay}$  (a full specification of this algorithm and analysis of its estimation accuracy is provided (Davy et al., 2008)).

### 5.2.2 Residual Bandwidth Adjustment

Existing network-aware VM placement techniques often apply the hose model (Mogul and Popa, 2012) to control the maximum data rates that the datacenter-hosted VMs can reach. With the hose model, bandwidth is allocated on a per-VM basis. The residual bandwidth of a server is then calculated as the server's link capacity minus the sum of the bandwidths allocated to VMs placed on the server. If a server's outgoing link capacity is  $C$ , with  $N$  VMs placed on it, each allocated a bandwidth of  $B$ , then the residual bandwidth of this server is taken to be  $(C - N \cdot B)$ . As described above, if VM bandwidth allocations are based on peak expected traffic rates, this approach is likely to lead to significant underutilization of the available bandwidth and the potential rejection of VM admissions that could be successfully provisioned.

In MAPLE, on the other hand, we apply effective bandwidth estimation to determine the minimum bandwidth required to provision the allocated VMs whilst meeting QoS targets. Accordingly, the residual bandwidth of a server is calculated as  $(C - R_{eff})$ , where  $R_{eff}$  in this case refers to the corresponding empirically estimated effective bandwidth of the

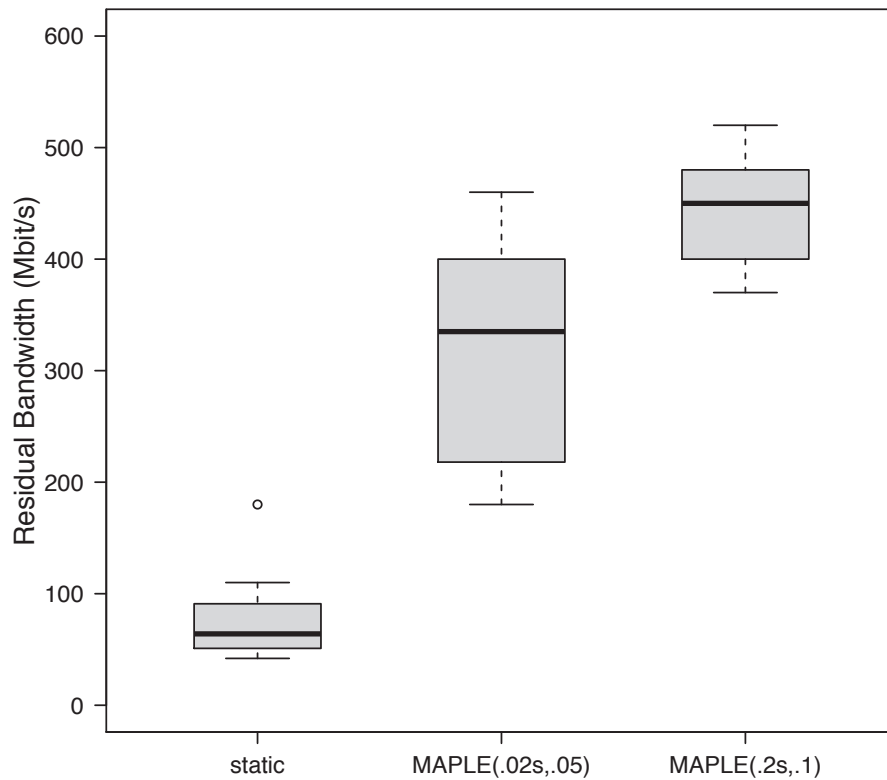


Fig. 5.1 Box plots of the residual bandwidth of a server having egress link capacity of 1 Gbps, obtained using our test-bed. The value inside the brackets indicate the QoS target, the first term being the target delay, the second the maximum percentage of violations of this delay target. When peak bandwidths are reserved for each VM, little residual bandwidth remains. In contrast, when bandwidth is allocated based on effective bandwidth, the available residual bandwidth is significantly higher, with the value depending on the stringency of the QoS target.

aggregated traffic currently transferred across the link. Fig. 5.1 illustrates the two different methods to calculate the residual bandwidth of a server that is hosting two VMs and has link capacity of 1000 Mbps. Static reservation refers to the method that reserves maximum throughput (on average 940 Mbps) for the VMs and computes the residual bandwidth accordingly, whereas effective adjustment refers to adjusting the residual bandwidth based on the effective bandwidth estimates. In this case, the latter determines that on average an aggregated bandwidth of 640 Mbps is sufficient for the two VMs for achieving a given QoS target (0.02s, 0.05) – no more than 5% of packets suffer delay longer than 0.02s, whereas given a lower QoS target (0.1s, 0.2) the residual bandwidth is higher.

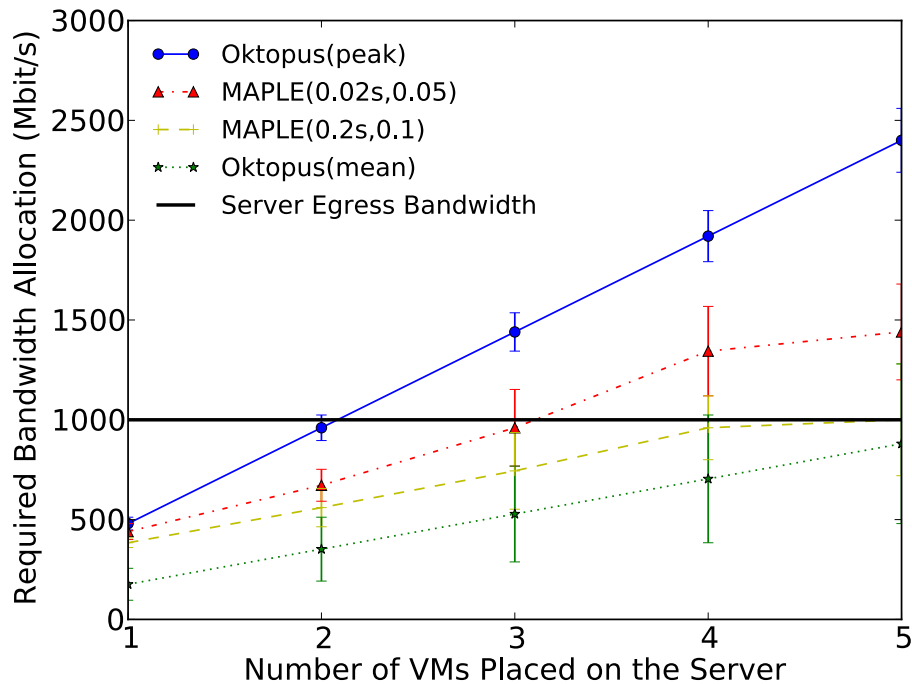


Fig. 5.2 As additional VMs are placed on a server, the required egress bandwidth increases. If peak or mean throughputs per VM are statically allocated as in Oktopus, this bandwidth increases linearly. If effective bandwidth estimations are used, the rate of increase reduces because of the smoothing effect of the statistical multiplexing of traffic from numerous sources. MAPLE can accommodate more VMs on each server (whilst satisfying QoS targets) than Oktopus (whilst allocating based on peak throughput).

Fig. 5.2 illustrates how an effective bandwidth technique can lead to more efficient utilization of computing resources in comparison to allocation of peak expected bandwidth when hosted VMs heavily use a server's egress bandwidth. We emulate the scenario where 5 VMs share an egress bandwidth of 1 Gbps. Given a QoS target of  $(0.02s, 0.05)$ , with more VMs arriving (recall that all VMs generate SCP traffic), it will be, as illustrated in Fig. 5.2, impossible to place more than 2 VMs on the server if the peak bandwidth of each VM must be strictly provisioned for their use. However, using MAPLE, it is possible to place up to 3 VMs on the server without violating the QoS target  $(0.02s, 0.05)$ . Given a lower QoS target of  $(0.2s, 0.1)$ , it is possible to host up to 5 VMs on the server, as shown in Fig. 5.2. Placing VMs on the basis of strictly provisioning for VMs' mean throughput would allow the placement of more VMs on the server. However, as discussed below, this would be at the cost of a significant number of QoS violations.

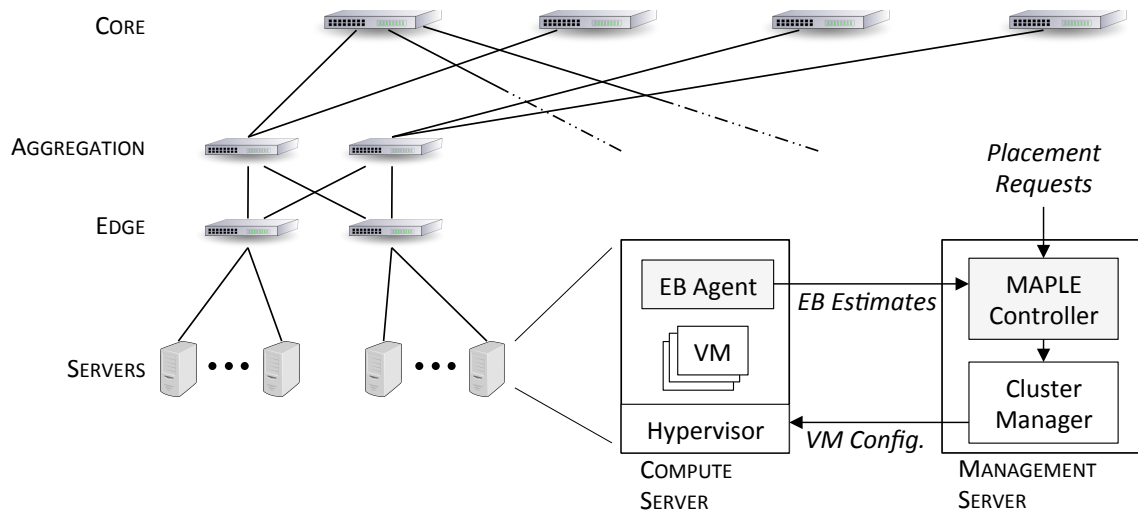


Fig. 5.3 The MAPLE system architecture. MAPLE is comprised of Effective Bandwidth Agents (EB Agents) residing in each server and a MAPLE Controller residing in a management server. EB Agents send effective bandwidth estimates upon request to the MAPLE Controller. The MAPLE Controller processes VM placement requests from tenants and instructs the Cluster Manager, which in turn configures VMs on the selected servers. Whilst the figure illustrates a Fat Tree datacenter topology (see Portland (Mysore et al., 2009)), MAPLE is agnostic of the topology since it collects bandwidth estimates at servers only.

## 5.3 MAPLE Architecture

MAPLE and its extension, MAPLE<sub>x</sub>, are designed to manage the joint allocation of network and computing resources in datacenter clusters to provision VM ensemble requests from tenants in a manner that provides statistical guarantees that QoS targets specified in SLAs are satisfied. Based on the design objectives described in §5.1, MAPLE has two main components. As illustrated in Fig. 5.3, these components are: 1) the MAPLE Controller, which is deployed on a cluster management server and interacts with a cluster manager such as VMware vCenter (VMware Inc., 2014); and 2) effective bandwidth estimation agents (EB Agents) deployed on each server, which analyze outgoing aggregated traffic to estimate its effective bandwidth and periodically send this information to the MAPLE Controller. We now describe these components in more detail.

### 5.3.1 EB Agents

EB Agents are deployed on every server in which MAPLE can place VMs. They are responsible for collecting traces of traffic emanating from the server using utilities such as



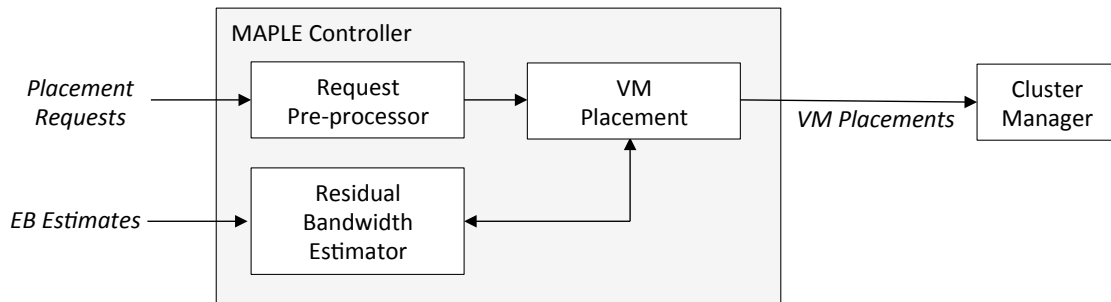


Fig. 5.4 The MAPLE Controller. Incoming requests for VM ensemble placement are first preprocessed to account for VMs that need to be placed together. VM placement decisions are then made taking into account the estimated available residual bandwidth at each server. Identified placements are passed to a Cluster Manager for configuration.

`tshark` (a command line version of the `Wireshark` network analyzer (Wireshark, 1998)). Traces are then used to estimate effective bandwidth using the approach described in §5.2. EB Agents collect traces at  $K$  minute intervals, each time storing  $S$  minute long traces. The effective bandwidth for the QoS target(s) specified by the MAPLE Controller is estimated for each trace. Whenever the MAPLE Controller requests an effective bandwidth estimate, the EB agent selects the maximum estimate from the last five stored estimates.

The effective bandwidth estimation process executed by EB Agents is an offline process, in the sense that it is not executed every time a VM ensemble placement request arrives. Given that, we have not undertaken a detailed study of its overhead, but our experience with our experimental evaluation of MAPLE indicates that EB Agents utilize an insignificant portion of the resources of the PM on which they are hosted. Moreover, this overhead can be controlled by adjusting the values of  $K$  and  $S$ . Finally, since all trace analysis is done locally on the server, there is minimal overhead incurred in EB Agent to MAPLE Controller communications.

### 5.3.2 MAPLE Controller

The MAPLE Controller handles incoming requests for VM ensemble placements, deciding on a one-by-one basis if a given request can be accepted and computing the placement if it can. As depicted in Fig. 5.4, it is comprised of three functional entities, which we now describe.

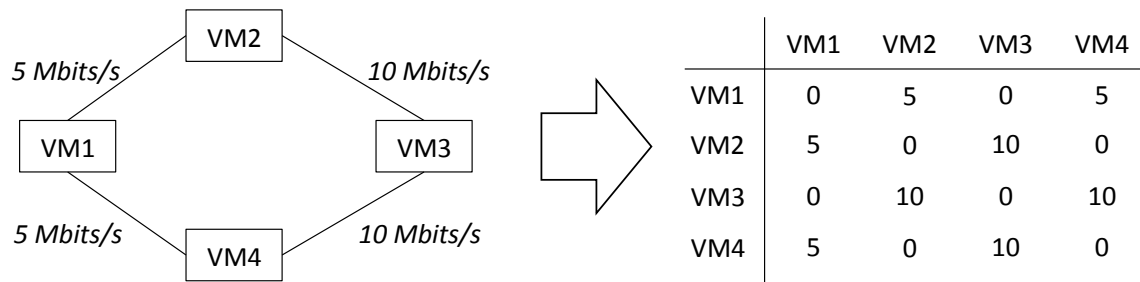


Fig. 5.5 A tenant VM ensemble topology specification indicates the expected peak traffic rates between VMs comprising the tenant application, together with the corresponding traffic matrix. In cases where VMs are to be co-located on the same server, the Request Preprocessor in the MAPLE Controller groups these VMs and generates a simplified traffic matrix.

### 5.3.2.1 Request Preprocessor

The Request Preprocessor receives VM ensemble placement requests that specify one of a small number of QoS classes offered by the datacenter provider. These QoS targets are specified in terms of packet delays rather than simply in terms of overall throughput levels. As depicted in Fig. 5.5, VM ensemble placement requests prescribe a topology of VMs that comprise an application and indicate peak traffic flow rates between VMs on a pairwise basis. For simplicity, this work assumes that the same amount of bandwidth is utilized in both directions between a VM pair; thus the total peak traffic rate for a VM in a server is simply the sum of the rates in the relevant row of the traffic matrix.

The Request Preprocessor also performs some preprocessing aimed to optimize the subsequent VM placement. It allows placement requests to specify that particular VMs should be placed together on the same server, given the expectation that they will interact heavily with one another. Given this, the Preprocessor and MAPLE Controller treat such VMs as a single VM, thus simplifying the traffic matrix accordingly. The opposite scenario, in which specified VMs are not to be co-located, is handled by MAPLE<sub>Ex</sub>, as described in §5.4.1.

### 5.3.2.2 Residual Bandwidth Estimator

This entity manages and queries EB Agents deployed on the compute servers under the control of the MAPLE system. Whenever a VM ensemble placement request arrives, the Residual Bandwidth Estimator queries a number of servers, who reply with their effective bandwidth estimate. A number of options are possible to govern which servers are queried.

For example, a number of servers can be sampled at random, with the number being calculated as a function of the overall occupancy of the datacenter clusters—in lightly loaded clusters fewer servers would need to be queried in order to find a viable placement. Alternatively, only servers already hosting VMs could be sampled initially so that VMs are consolidated for energy optimization purposes. For simplicity, we assume here that all servers in the cluster are queried; alternative strategies will be explored in future work. When effective bandwidth estimates are received, the residual bandwidth as seen by the servers is calculated and the server IDs and associated residual bandwidth estimates are passed to the VM Placement entity.

### 5.3.2.3 VM Placement

This entity takes as input the preprocessed VM ensemble placement requests. It requests the Residual Bandwidth Estimator to provide it with a set of candidate servers for the placement and the estimates of residual bandwidth available on the egress links of those servers. It then executes the MAPLE or MAPLE<sub>x</sub> network-aware VM placement algorithms specified in §5.4. If the VM ensemble can be placed, the placement details are passed to the Cluster Manager that applies configurations on the servers accordingly.

## 5.4 Network-aware VM Placement Algorithms

The MAPLE Controller seeks to minimize both the nominally allocated network bandwidth and the number of servers in which VMs are placed, such that QoS targets are met. As multi-objective VM placement problems are known to be NP-hard (Breitgand and Epstein, 2012; Meng et al., 2010; Wang et al., 2011), in this work MAPLE uses the heuristic algorithms specified in Alg. 3 and Alg. 4 to produce VM ensemble placement results in a timely manner. The proposed algorithms employ First Fit Decreasing (FFD) approach to search for VM placement solutions—FFD has been widely applied to VM placement problems (Breitgand and Epstein, 2012; Shi et al., 2013b; Wang et al., 2011). Theoretical proof shows that FFD approach can produce asymptotic approximation ratio of  $\frac{11}{9}$  (Yue, 1991).

As shown in 3, the proposed algorithms first sorts servers in increasing order of their  $\beta$  values using Eqn. 5.1 and commences searching. The approach is known as “first fit” because it stops searching once it finds the first feasible placement. FFD approaches

generally give sub-optimal results, but are able to find feasible solutions in a timely manner. In our algorithm, VM placements are optimized in the sense that, since servers are sorted in increasing order of residual bandwidth, the first fit placements are most likely found on servers that have relatively smaller residual bandwidths. In this way, VMs are placed into servers until there is no available computing capacity or sufficient residual bandwidth to meet the QoS target. This serves to both minimize the number of servers used and to ensure that the overall bandwidth nominally allocated to VMs is minimized.

The  $\beta$  metric, computed using Eqn. 5.1, is used to ensure that the MAPLE algorithm sorts servers in increasing order of residual bandwidth. The coefficient  $g \in [0, 1)$  is a configurable control parameter that influences the degree to which  $\beta$  reflects the residual number of VM slots. For example, when  $g = 0$ , the residual number of VM slots in the servers will have no effect; otherwise, when servers have the same level of residual bandwidths, those with fewer remaining VM slots will be searched first. Note that in our experimental evaluation we have used  $g = 0.5$ . In previous work (Shi et al., 2013b) we investigated the approach of formulating a metric similar to  $\beta$  as the square root of the sum of squares of the components of interest, verifying this approach's effectiveness in terms of resource utilization efficiency; other authors, for example (Sheikhalishahi et al., 2014) have adopted a similar formulation.

$$\beta = \sqrt{\left(\frac{\text{residual\_BW}}{\text{total\_BW}}\right)^2 + g \cdot \left(\frac{|\text{residual\_slots}|}{|\text{all\_slots}|}\right)^2} \quad (5.1)$$

Notice that both the residual bandwidth level and residual slot level are expressed as proportional values. This reflects the fact that the absolute values of the residual bandwidth and number of slots can vary significantly, leading one to dominate the other.

The input to the MAPLE algorithm is a VM ensemble request  $\langle N, B \rangle$ , where  $N$  is a set of VMs in request and  $B$  is the set of associated expected peak bandwidth utilization. To simplify the presentation of the algorithm without losing generality, we assume that all VMs expect the same peak bandwidth utilization. The algorithm assumes that the datacenter topology can be represented by a tree structure, which is the case for fat-tree topologies typically used in datacenter networks (Greenberg et al., 2009) (Mysore et al., 2009) (Bitar et al., 2013). When  $n$  requests arrive at a time  $t$ , these requests will be sorted based on their  $\beta$  values using Eqn. 5.1, and then processed one by one. The algorithm handles three cases:

**Algorithm 3** MAPLE VM Ensemble Placement**Input:**  $N, B, node$ **Output:** True or False

---

```

1: done=False, subtrees=node.subtrees
2: if sizeof(subtrees) = 0 then
3:   if node.remainSlots  $\geq$  sizeof( $N$ ) and
   node.remainBW  $\geq$  sizeof( $N$ )* $B$  then
4:     allocate  $N$  to node
5:     return True
6:   else
7:     return False
8:   end if
9: else
10:  sort subtrees based on the  $\beta$  values Eqn.5.1
11:  for subtree in subtrees do
12:    done = MAPLE( $N, B, subtree$ )
13:    if done = True then
14:      return done
15:    end if
16:  end for
17: end if
18: if done == False then
19:  tail = allocBetweenNodes( $N, B, subtrees$ )
20: end if
21: if tail !=  $N$  then
22:  done = MAPLE(tail,  $B, node$ )
23: end if
24: return done

```

---

1. Case I (shown in lines 2-8)—when a given node is a server (the lowest subtree that has no further subtree, as shown in line 2), MAPLE attempts to allocate the entire VM ensemble placement request into the same server;
2. Case II (addressed in lines 10-16)—if a node has subtrees, the subtrees will be sorted increasingly based on their  $\beta$  values, and MAPLE attempts to allocate the entire VM ensemble placement request into the same subtree. Notice that here we assume that the routing cost is relatively less expensive when VMs are located in the same subtree;
3. Case III (addressed in lines 18-23)—when the algorithm cannot find any subtree that can host the entire VM ensemble, it attempts to allocate the requested VMs into different subtrees. VMs are divided into two groups: *head* and *tail*. Function

**Algorithm 4** allocBetweenNodes function**Input:**  $N, B, nodes$ **Output:**  $tail$ 


---

```

1:  $tail=N$ 
2: sort  $nodes$  based on the  $\beta$  values eq.5.1
3: for  $node$  in  $nodes$  do
4:   if  $node.remainSlot = 0$  then
5:     continue
6:   end if
7:    $m=node.remainSlot, n=sizeof(N), t=\min(m, n - m)$ 
8:   if  $node.remainBW \geq t*B$  then
9:      $head=N[0 : m], tail=N[m : n]$ 
10:    if  $MAPLE(head, B, node) = \text{True}$  then
11:      return  $tail$ 
12:    else
13:      continue
14:    end if
15:  end if
16: end for
17: return  $tail$ 

```

---

allocBetweenNodes() (specified in Alg. 4) returns  $tail$ , which is the group of VMs not yet allocated after it successfully allocated the VMs in the  $head$  group; otherwise, Alg. 4 returns the original input, indicating that the input VMs cannot be allocated into different subtrees.

In an approach similar to that taken by Oktopus (Ballani et al., 2011), whenever a VM ensemble request cannot be entirely placed into one subtree (case III), the requested VMs will be divided into two groups. The aggregate bandwidth needed by the group being placed is determined as the minimum aggregated bandwidth between the two groups. As illustrated in algorithm 4 in lines 7-8, the head group has  $m$  VMs, and the tail group has  $n - m$  VMs. The algorithm will only allocate  $\min(m, n - m) \times B$  as the aggregated bandwidth needed by the VM group has a placement. However, differently than in Oktopus, MAPLE estimates the residual bandwidth of a server based on effective bandwidth. The variable  $node.remainBW$  (line 3) in Alg. 3 is calculated by  $(C_i - R_{eff}^i)$ , where  $C_i$  is the link capacity available at server  $i$ , and  $R_{eff}^i$  is the aggregated effective bandwidth (for the QoS target sought by the request under consideration) of the VMs already allocated to server  $i$ .

### 5.4.1 MAPLEx—Supporting Anti-colocation Constraints

Anti-colocation constraints for VM placements have been studied previously (Bin et al., 2011; Jayasinghe et al., 2011; Shi et al., 2013a), however the MAPLE algorithm as introduced above cannot accommodate them. In this section, we describe the MAPLEx algorithm, specified in Alg. 5, which enables us to use effective bandwidth to place VM ensemble requests involving anti-colocation constraints. The MAPLEx algorithm has a similar procedure to MAPLE, the essential difference bring in the inputs the algorithms take. The input to the MAPLEx algorithm is a VM ensemble request  $\langle N, B, x \rangle$ , where the new parameter  $x$  represents the number of VM slots that will be occupied by a given request on each server. The introduction of  $x$  enables the algorithm to cope with different variations of VM location constraints that can be arbitrarily defined by the parameter  $x$ . In particular, two types of anti-colocation constraint can be supported:

- Setting  $x = 1$  means that MAPLEx will be forced to allocate a maximum of one VM slot on each server to a given VM ensemble request. This equates to the anti-colocation constraint that *VMs from the same ensemble cannot be placed on the same server*.
- Setting  $x = z$ , where  $z := \{\text{the maximum number of slots on a server}\}$  means that MAPLEx will be forced to mark all the VM slots of a server occupied, once the server is allocated with a tenant's VM, regardless that the actually number of VM slots required is less than  $z$ . For example, a tenant requests 5 VMs ( $|N| = 5$ ), while a given server has 10 slots ( $x = z = 10$ ), then after placed with 5 tenant's VMs, the remaining 5 slots of that server will be marked as occupied, so the entire server is reserved to the tenant. This setup copes with the anti-colocation constraint that *the VMs from an ensemble cannot be placed on a server hosting VMs from other ensembles*.

The algorithmic structure of MAPLEx is similar to MAPLE—it proceeds in a similar manner. We thus explain only the MAPLEx specific aspects of Alg. 5. Line 5 creates the set of VMs  $X$  to be allocated to a given server. The size of  $X$  is the minimum number between  $n$  (size of  $N$ ) and  $x$ , as we can see in lines 7-11, for the cases  $x \geq n$ , MAPLEx will assign the  $n$  VMs to the server and mark the remaining slots ( $x - n$ ) of the server as occupied; while  $x < n$ , MAPLEx allocates  $(n - x)$  VMs to the server, reset  $N = N - X$  (in line 11) and return false, subsequently the iterative process will pick up the updated  $N$  and continue to allocate  $N$  until the size of  $N$  equals to 0. The MAPLEx algorithm, as specified here, is generic in the

**Algorithm 5** MAPLE<sub>x</sub> VM Ensemble Placement**Input:**  $N, B, x, node$ **Output:** True or False

---

```

1:  $done=False, subtrees=node.subtrees$ 
2: if  $sizeof(subtrees) = 0$  then
3:    $n = sizeof(N)$ 
4:   if  $node.remainSlots \geq x$  and
      $node.remainBW \geq n*B$  then
5:      $X := \{VM_i \in N | 0 \leq i < \min(n,x)\}$ 
6:     allocate  $X$  to  $node$ 
7:     if  $x - n \geq 0$  then
8:       mark all the rest of slots  $(x - n)$  as occupied
9:       return True
10:    else
11:       $N = N - X$ 
12:      return False # then continue with the others
13:    end if
14:  else
15:    return False
16:  end if
17: else
18:  sort  $subtrees$  based on the  $\beta$  values eq.5.1
19:  for  $subtree$  in  $subtrees$  do
20:     $done = MAPLE_x(N,B,x,subtree)$ 
21:    if  $done = True$  then
22:      return  $done$ 
23:    end if
24:  end for
25: end if

```

---

sense that different forms of anti-colocation constraints can be specified by adjusting the  $x$  parameter in the input of VM ensemble request.

## 5.5 Evaluation

### 5.5.1 Experimental Setup

For our experimental platform we used 4 Dell R720 servers and 1 physical switch to create an Emulated Datacenter (EDC). Each Dell server has 16 cores running at 2.6GHz, 128GB of



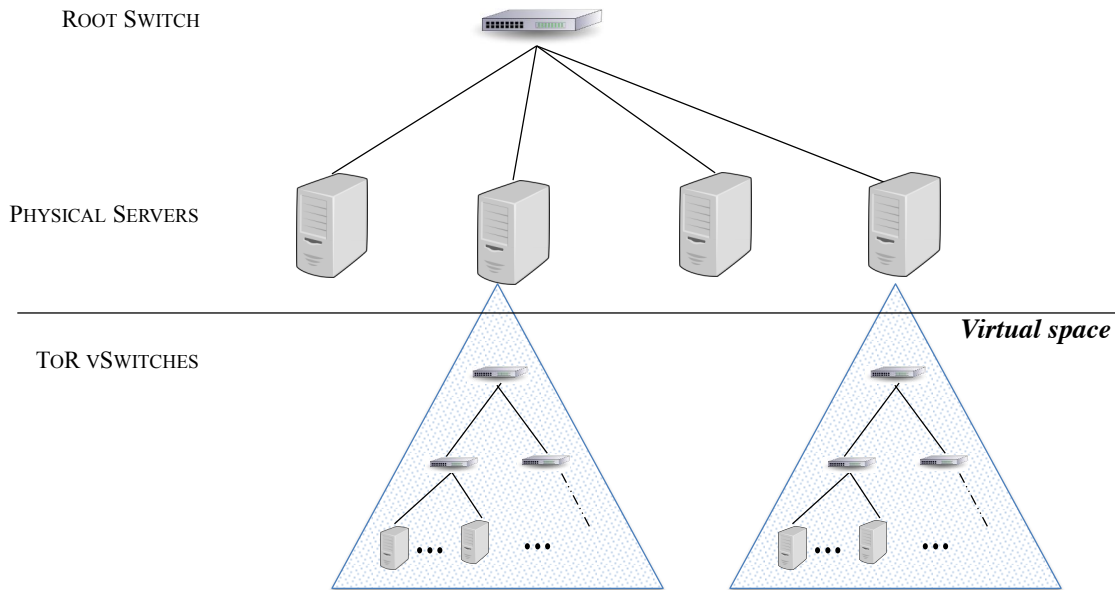


Fig. 5.6 The emulated datacenter has a single root switch connecting 4 physical servers. Each server hosts 70 VMs that are connected by 2 levels of vSwitches.

RAM, and two 600GB hard disks. Each server is initially installed with Ubuntu 12.04 system (the host OS), with the `libvirt` library (Libvirt Community, 2005) being used to manage up to 70 guest VMs deployed on each physical server. Fig. 6.4 presents our EDC, which is configured to emulate a simplified datacenter tree topology, without multiple paths between switches. The topology comprises virtual servers and virtual racks, as follows:

- *Virtual Servers (vServer)*—we group 5 VMs together to emulate a vServer, wherein all VMs are directly connected to a Virtual Switch (vSwitch). This vSwitch is accordingly denoted as vNIC (virtual Network Interface Card), functioning like a NIC of a server. Each vNIC is limited to have only 1 Gbit upstream and downstream bandwidth, to emulate the scenarios that 5 VMs share one bottleneck link within a server;
- *Virtual Rack (vRack)*—each of the 4 physical servers is used to emulate a rack of vServers. Within each physical server, there are 2 levels of vSwitches to connect 70 VMs: the top level vSwitch, denoted as Top-of-Rack (ToR) vSwitch, connects the low level vSwitches to the root physical switch; the low level vSwitches connect the VMs in groups.

Three types of data-intensive applications are employed to run on the EDC: 1) For a data analysis application, we used a Hadoop application that runs word count on distributed documents; 2) for a data serving application, we use the combination of YCSB (data client) and Cassandra (data server), a benchmark recommended in cloudsuite (Ferdman et al., 2012); finally, 3) for a data backup application, we created a program based on SCP utility, which simulates the scenario that a copy of data file is copied to a couple of remote nodes for backup purposes. To evaluate the efficacy of our emulation approach to building a test environment we performed a preliminary investigation of the mean/maximum throughputs and run times of the three applications when deployed on the emulated EDC in comparison to an equivalent deployment on a real OpenStack™ based IaaS environment in a mixed used production/experimental datacenter in Waterford Institute of Technology. Results of this investigation, which are not presented here, confirm that the emulation approach provides a reasonable approximation of a production environment.

For VM ensemble requests, we created a program that randomly generates requests specified in a format  $\langle N, B, T \rangle$ . It is similar to the hose model  $\langle N, B \rangle$  with an additional parameter  $T$  to indicate the type of application.  $N \in [2, 10]$  is an integer value randomly drawn from a Gaussian distribution  $\mathcal{N}(5, 1)$  with mean 5 and standard deviation 1. Because we use a mean of 5, each request will, on average, ask for 5 VM instances. In turn, we know that the emulated datacenter, having 280 VM slots, can more or less accept 56 requests. To simulate the dynamics of VM requests, each request is configured with a service time (selected randomly with an average value of 2100s); when this elapses, the corresponding VMs will be removed from the EDC, some VM slots will therefore become available again. For each experimental run we generated at least 60 VM ensemble requests, the requests are divided into 10 batches, and every 100s one batch was fed to MAPLE/Oktopus. Given this, assuming the last request comes after 500s a single experimental run lasted 2600s. Results presented are based on 10 independent simulation runs.

We compared the performance of MAPLE with two variants of Oktopus (Ballani et al., 2011): Oktopus allocating network resources based on expected mean throughput, and Oktopus allocating network resources based on expected peak throughput. The mean throughput and peak throughput were the average values measured on the aggregated traffic traces collected at the initial experiments. After that, at each experimental run, all algorithms were given with the same sets of VM ensemble placement requests. Only the corresponding bandwidth requirements were substituted, according to the respective Oktopus variants. In our experiments for MAPLE/MAPLEx we configure EB Agents to start the process of

generating new effective bandwidth estimates every  $K = 300s$ , with estimates being made on traces for intervals of  $S = 50s$ . Thus, when effective bandwidth for links change due to changes in VM placements on PMs, it can take up to 300s for this to be recognized by the MAPLE controller.

The goal of this evaluation is to compare the involved VM placement techniques in two metrics: reject rates of VM ensemble requests and QoS violation rates. All QoS target violation rates were calculated offline based on the collected traffic traces, where QoS targets are represented in the format  $(delay, proportion)$ . We use the following QoS targets:  $(0.02s, 0.05)$ ,  $(0.02s, 0.1)$ ,  $(0.04s, 0.05)$ , and  $(0.2s, 0.1)$ . We highlight that QoS delay requirements vary depending on the hosted applications, but delays of up to 200ms are generally considered acceptable for real-time interactive services (Cisco Online Document, 2006) and private communications with an operator of a large enterprise datacenter supporting a range of transactional web-based services suggested that a similar delay target is often acceptable.

### 5.5.2 Residual Bandwidth Required to Place New VMs

We start our study with an interesting question arose from our previous work (Wang et al., 2014), where the experiments showed that the use of effective bandwidth can result in a relatively optimal VM placement schemes, by which effective bandwidth is always an value between the involved applications' mean throughput and peak throughput. The question is “whether Oktopus produce a similar placement results by allocating bandwidths based on  $\theta \times \text{mean\_rate}$ , for  $\theta > 1$ ?” The major issue of the described approach is that setting a fix value of  $\theta$  cannot adjust to the dynamics of bandwidth requirements, thus resulting in bandwidth overprovisioning over time. The approach will only work unless there is a dedicated mechanism in place to dynamically reset  $\theta$  in response to changes in network. In fact, the aggregated effective bandwidth needed by applications that does not grow linearly, e.g., the aggregated effective bandwidth of 3 flows is not equal to the sum of the three individual effective bandwidth of the flows, rather, it generally less than the sum. Thus, if we allocate the 3 flows with the bandwidth amount of  $3 \times \theta \times \text{mean\_rate}$ , that could result in over-provisioning.

Fig. 5.7 depicts the change in the level of additional bandwidth to be allocated when a new VM is placed on the server. This is calculated as  $\frac{R_{eff}}{|VMs|}$ , that is, the aggregated effective bandwidth,  $R_{eff}$ , divided by the number of allocated VMs,  $|VMs|$ . Here, we can observe the

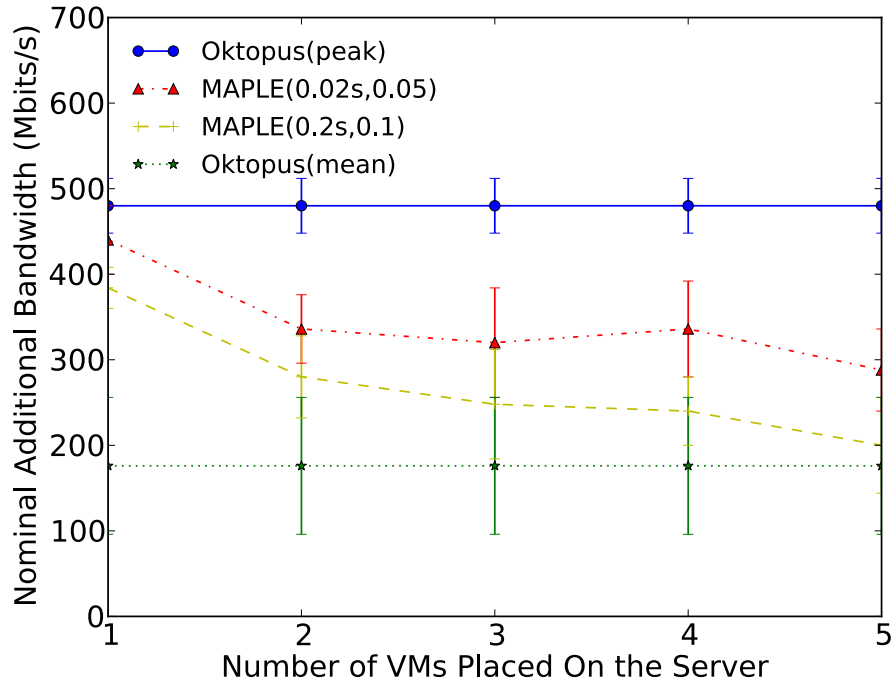


Fig. 5.7 Additional bandwidth that needs to be allocated when a new VM is placed on a server. MAPLE can place additional VMs based on the availability of less residual bandwidth than can Oktopus using peak throughput estimations, since it takes into account the statistical multiplexing effect captured by the effective bandwidth estimation.

non-linear nature of effective bandwidth—reflecting the statistical multiplexing of traffic from numerous sources, which means that less incremental bandwidth needs to be provided to attain the same level of QoS. A detailed experimental and theoretical analysis regarding this nature of effective bandwidth can be found, respectively, in the work of (Davy et al., 2008) and Kelly (Kelly, 1996).

### 5.5.3 Analysis of QoS Violations

We analyze QoS violations in the EDC that are associated with the operation of the VM placement algorithms. First, note that the QoS violation rate is calculated based on individual server's egress links. Namely, the violation rate is the proportion of a server's egress packets that experience delays longer than that specified in the QoS target. To examine the QoS violation at the overall datacenter scale, we employ two metrics: the *overall violation rate* as defined in Eqn. 6.5, and the *averaged violation rate* as defined in Eqn. 6.6. The *overall violation rate* is the sum of violation rates of all links divided by the total number of all

observed links in the datacenter. However, this metric only shows the overall performance; it does not reflect the situation that the violations are very localized—most of the links have no, or tiny numbers, of QoS violation levels, while links that face QoS violations suffer frequent, significant violations. In this case, the averaged violation rate can indicate how strong the violations are on the links that suffer violations. It is important to have these complementary metrics, since both MAPLE and Oktopus algorithms attempt to allocate VMs in a same subtree with small residual bandwidths, in order to save routing costs, which will frequently result in scenarios where VMs are densely co-located around some links.

$$\text{overall\_violation\_rate} = \frac{\text{sum}(\text{QoS\_violation\_rates})}{|\text{all\_links}|} \quad (5.2)$$

$$\text{averaged\_violation\_rate} = \frac{\text{sum}(\text{QoS\_violation\_rates})}{|\text{links\_with\_violations}|} \quad (5.3)$$

The evaluation of QoS target violation was calculated off-line based on the collected traffic traces. Given the scale of our EDC, it is not viable to collect and analyze traffic traces on every link for each run. Thus, we only focus on the links that have inter-vServer/inter-group traffic. Regarding the “all link” described in Eqn. 6.5, we mean all the links that in observations, namely the links with active traffic sniffers. Fig. 6.6 depicts QoS violations where the QoS target is (0.02s, 0.1). The results compared the QoS violations incurred by two different sets of VM placements given by MAPLE and Oktopus when allocating based on mean throughput (denoted as Oktopus (mean) in the figure). Since VM placements given by Oktopus allocating using peak throughput generally cause no QoS violations or small violation rates, Fig. 6.6 only presents results for the other two algorithms.

From Fig. 6.6 we can observe that when allocating using mean throughputs to all VMs, Oktopus tends to have high probability ( $\approx 60\%$ ) that packets will suffer with packet delays greater than 0.02s. In particular, for the links where QoS violations actually occurs, we observe very strong violation rates, on average  $\approx 77\%$ . In contrast, MAPLE, which allocates on the basis of effective bandwidth estimates, generally keeps QoS violations within the target range, however, regarding the links with strong violations, MAPLE also suffers with strong violation rates, on average  $> 65\%$ . The CDF of mean link utilisation levels averaged across the duration of an experimental run presented in Fig. 5.9 indicates that the majority of violations result from overloads of a small number of highly loaded links.

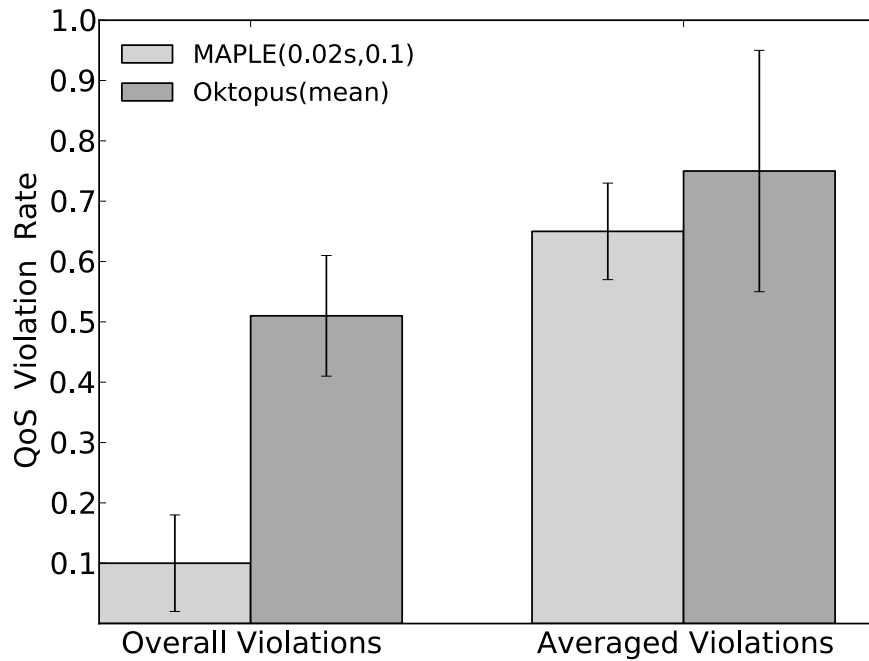


Fig. 5.8 QoS violation levels for QoS target (0.02s,0.1) for MAPLE and Oktopus. Oktopus over allocates network resources resulting in significant overall and localized levels of QoS violation. MAPLE's usage of effective bandwidth estimates means that QoS violations are within the acceptable range.

For a QoS sensitive network, it is also important to be able to control the level of QoS violation, being flexible to allocate just enough resources to maintain the QoS targets. Given three different QoS target (0.02s,0.05), (0.04s,0.05), and (0.2s,0.1), Fig. 5.10(b) and Fig. 5.10(c) show that MAPLE can keep QoS violation at acceptable levels for the targets (0.04s, 0.05) and (0.2s, 0.1), in terms of their *overall violation rates*. However, for the strictest QoS target, (0.02s, 0.05), MAPLE did suffer with some congested links that lead to violation rates greater than 0.05, resulting in Fig. 5.10(a), where we see that the overall violation target is not met. Whilst MAPLE can meet the target in most cases, Oktopus results in varying violation rates, depending on the different delay targets—it never succeeds in keeping the rates within the acceptable range.

As shown in Fig. 6.6 and Fig. 5.10 that all the compared techniques have high *averaged violation rate*, as previously discussed low *overall violation rate* does not necessarily mean the *averaged violation rate* should be low, as in the cases described above for MAPLE. We investigated the traffic samples to find out the reason why *averaged violation rates* tend to be high, the summary of our finding is that this relates to a bursty application (Hadoop) running

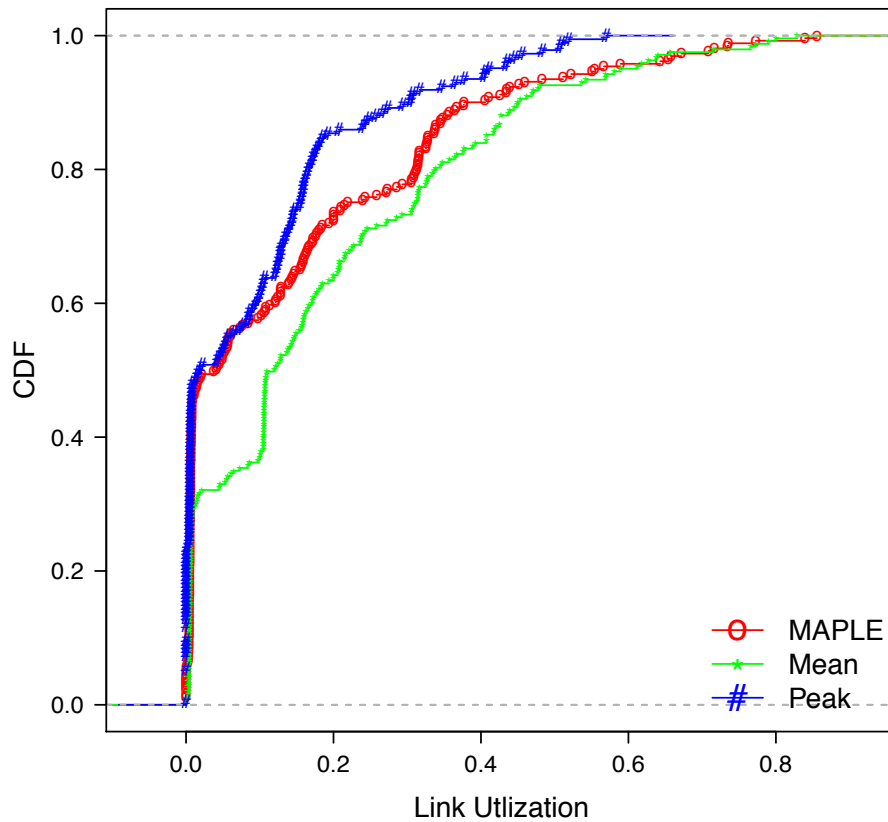


Fig. 5.9 CDF of the mean link utilization of all DC links for which effective bandwidth estimates are generated, averaged over the full duration of an experimental run. We see that the majority of links are lightly loaded when MAPLE and both variants of Oktopus are used. Most QoS violations result from overloads of the smaller number of highly loaded links.

on a bursty network. As described by (Chowdhury et al., 2011) Hadoop traffic tends to be very bursty for some specific periods. Regardless of Hadoop, some SCP and YCSB traces also contribute some violation samples; we attribute this mainly to our Open vSwitch controlled network. Specially, during the initial configuration of the test-bed, we followed the open vSwitch community guidance (Openvswitch Community, 2014) to set up the rate limit, where it is suggested to set a higher burst parameter for the vSwitch controlled network in order to manage bursty traffic, resulting in a bursty network and possibly congestion.

Although MAPLE occasionally fails to meet the QoS targets for some links that have strong violations, for most links (> 80%) it meets the targets. Overall, it has much better performance than Oktopus using mean throughput regarding achieving QoS control.

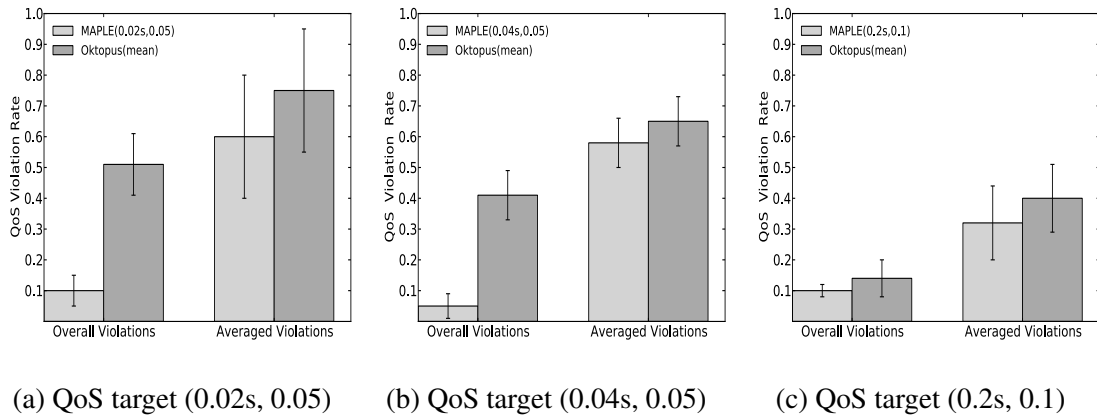


Fig. 5.10 QoS violation rates for different delay based QoS targets. MAPLE mostly can keep violations below the specified target level or, if not, at a relatively low level, whereas Oktopus (mean) never does.

### 5.5.4 Analysis of Rejection Rates of VM Ensemble Requests

Given that we know our test-bed generally can accommodate simultaneous placement of 56 requests, if the compared placement algorithms are provided with the number of requests that far beyond the corresponding EDC's capacity, this will result in large rejection rates. Therefore, in each run we only give 60 requests to a completely empty test-bed; We intentionally set the proportion of VM requests to be removed to be approximately 20% (i.e.,  $\approx 12$  requests) in each experimental run, subsequently we give 18 requests to the test-bed in a state that some of its requests and the corresponding VMs have been removed.

In each run, the EDC was populated with VMs until it was almost fully loaded, that is, when it could not accept new VM ensemble placement requests any more. Fig. 5.11 depicts the reject rates for MAPLE with four different QoS targets and Oktopus using mean and peak throughput. We observe that the peak throughput approach has a significantly higher reject rates, compared to the other algorithms. The first two members of MAPLE's family seem to have similar reject rates, which is somehow reasonable as they have the same delay limit. In particular, when many flows are involved, the aggregated effective bandwidth with similar QoS targets tends to converge. With a less strict QoS requirement (0.04s, 0.05), MAPLE produces a comparable rejection rate to Oktopus (mean), which has the lowest reject rate (at the expense of QoS violations for reasonable QoS targets).

Generally, given a less strict QoS requirement, MAPLE should produce lower rejection rates. However, from the box plot in Fig. 5.11, we observe that when comparing the maximum reject rates between MAPLE(0.04s, 0.05) and MAPLE(0.2s, 0.1), the latter shows a higher



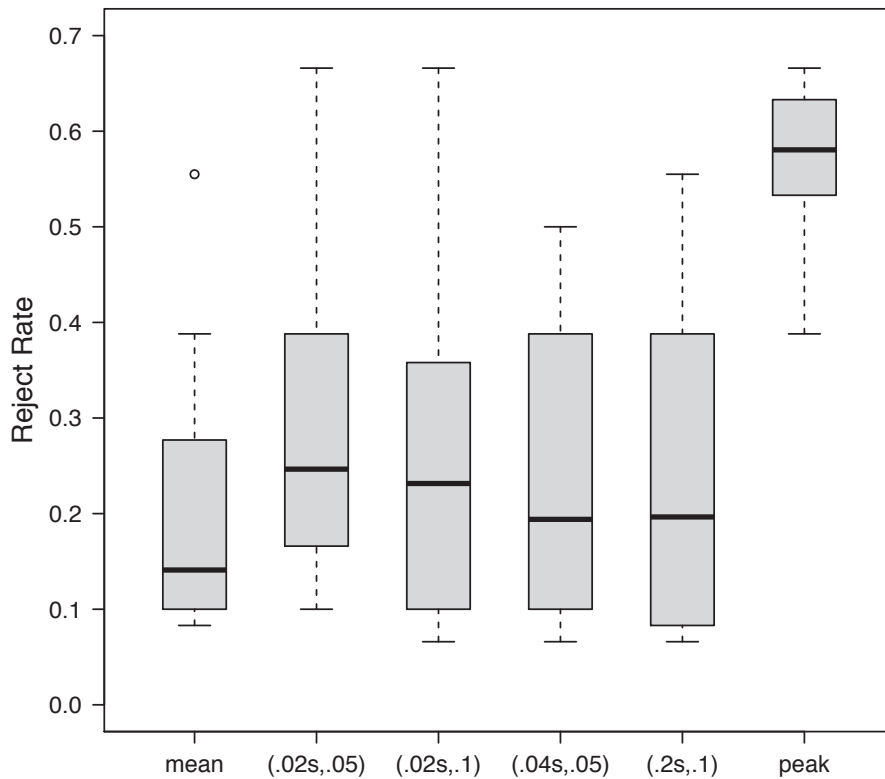


Fig. 5.11 Oktopus (mean) has the lowest rejection rate, while MAPLE's rejection rates are comparable to it, but dependent on the QoS targets. Oktopus (peak) has the highest rejection rate.

value. We investigated the related batches of ensemble requests, and found that is due to MAPLE(0.2s,0.1) was able to accept some requests requesting bigger number of VMs, resulting in subsequently rejecting more requests of smaller number of VMs; whereas MAPLE(0.04s, 0.05) rejected one or two requests of big number of VMs thus was able to accept more requests of small number of VMs.

Overall the rejection rates produced by MAPLE are in between the rejection rates of Oktopus using mean throughput and Oktopus using peak throughput. The values of rejection rates of MAPLE seem more varied, which reflects the dynamics of effective bandwidth requirements needed by the datacenter applications, so that the amount of VMs can be placed is varied. Oktopus variants that always reserve the same bandwidth tend to be more static in terms of rejection rates.

Based on the experimental results on QoS violation rates and request reject rates, overall we can conclude that MAPLE is relatively resource-conserving while providing targeted QoS statistical guarantees.

### 5.5.5 Analysis of MAPLEx

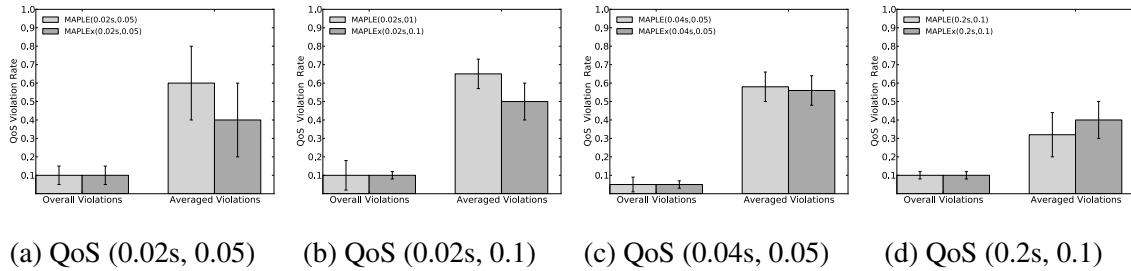


Fig. 5.12 QoS violation rates for different delay based QoS targets. In most cases MAPLE and MAPLEx can keep violations below the specified target level, although some links suffer relatively high violation rates.

To evaluate MAPLEx it is given the same batches of requests that were given to Oktopus and MAPLE in the experiments described above. However, in these batches of requests, we randomly select approximately 70% of the requests to have the anti-colocation constraints, such that 30% of the overall requests have the constraint that VMs from the same ensemble cannot be placed on the same server and 40% of the overall requests have the constraint that the VMs from an ensemble cannot be placed on a server hosting VMs from other ensembles.

Fig. 5.12 compares the QoS violation rates incurred by the placements of MAPLE and MAPLEx for this request pattern. We see that, since placement decisions are based on available effective bandwidth, MAPLEx, like MAPLE, is typically able to control the delay violation rates to meet the given targets. However, as explained earlier, due to the presence of bursty applications, there also exists some link congestion, leading to some undesired QoS violations. For three of the four QoS targets investigated, MAPLEx has a better averaged violation rate, which captures the degree to which violations occur when they do occur. This is due to MAPLEx tending to distribute VMs across more PMs than does MAPLE, due to the presence of the anti-colocation requests (and, if no PMs are available to reject more placement requests). Thus, those links that do suffer violations due to overload generally have lower mean carried load for the MAPLEx case in comparison to the MAPLE case, so that the impact of transient high traffic demands from VMs is lessened in terms of the duration of the overload, leading to a lower level of QoS violation.

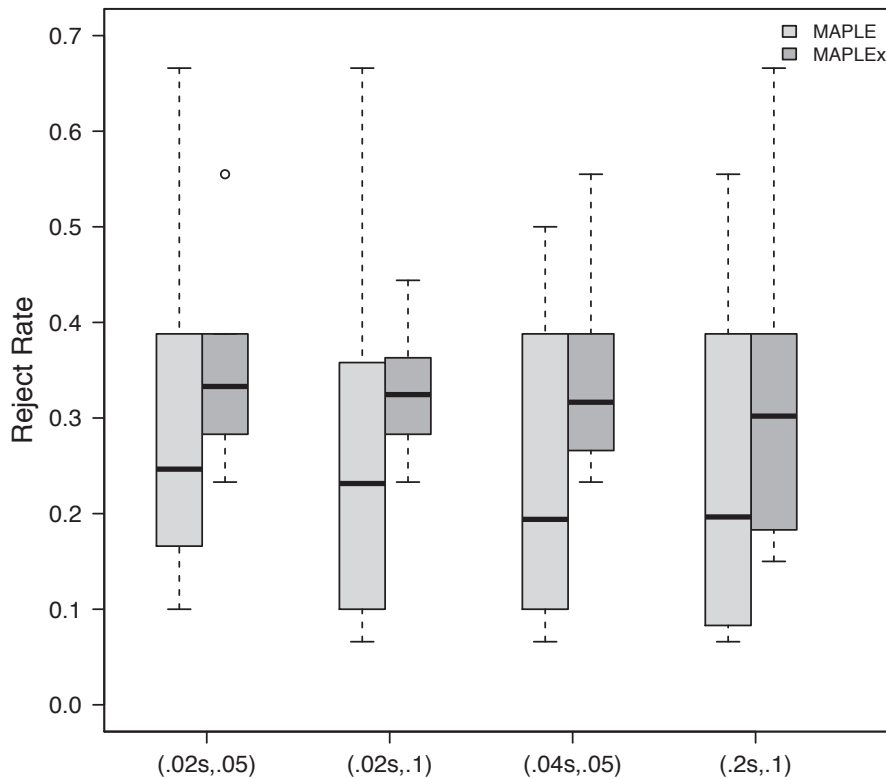


Fig. 5.13 Rejection rates of ensemble placement request—MAPLEx tends to show higher placement rejection rates in comparison to MAPLE. The presence of anti-colocation constraints, particularly those mandating that only VMs from a given ensemble can be placed on a given server effectively reduce the overall capacity of the cluster to accommodate VM ensembles.

Fig. 5.13 compares the request rejection rates; we can observe that anti-colocation constraints supported in MAPLEx frequently result in more requested placements being rejected. In particular, the constraints that VM ensemble does not share server would cause some slots marked occupied but not actually being used—effectively lowering the capacity of the cluster to accommodate VM ensembles. Fig. 5.14 shows that with MAPLEx links generally have lower mean utilization than with MAPLE. This results from MAPLEx tending to place VMs across more PMs (to satisfy anti-colocation constraints) and its tendency to reject more VM ensemble placement requests (because there is less available PM capacity due to some PMs hosting VMs that cannot share resources with VMs from other ensembles).

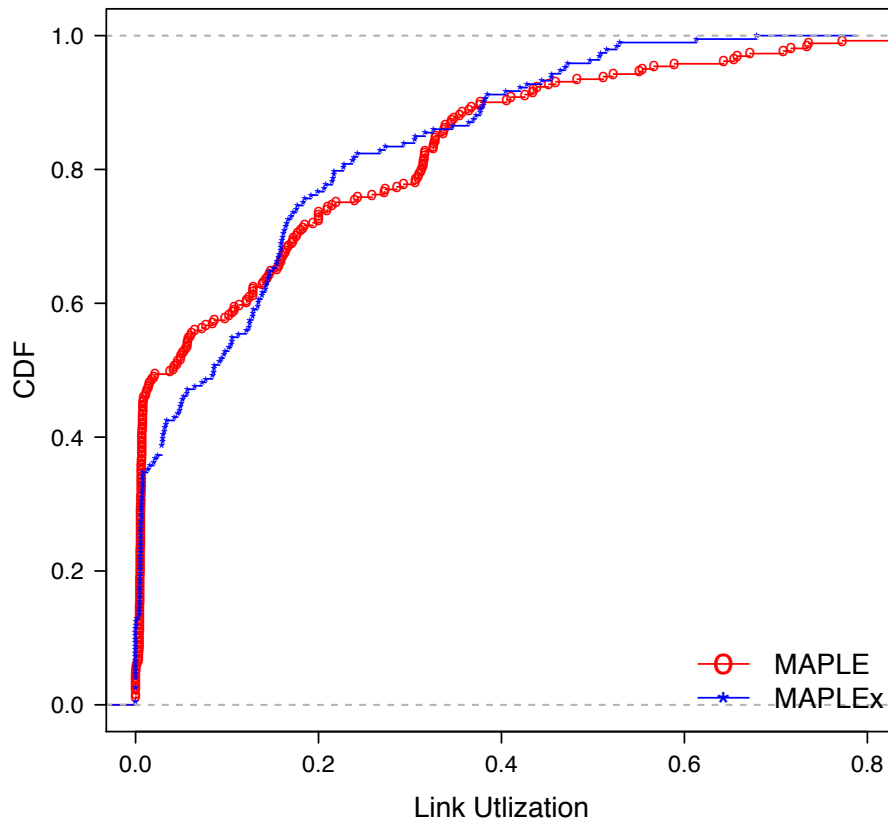


Fig. 5.14 CDF of the mean link utilization of all DC links for which effective bandwidth estimates are generated, averaged over the full duration of an experimental run for MAPLE and MAPLEx. We see that for MAPLEx, links have generally lower mean utilization, resulting from MAPLEx tending to place VMs across more PMs (to satisfy anti-colocation constraints) and its tendency to reject more VM ensemble placement requests than does MAPLE.

## 5.6 Summary

This chapter described the use of a test-bed that utilizes four physical servers to emulate a datacenter cluster having 280 VM slots running three types of data-intensive applications. Using this larger scale of test-bed running different applications, we confirmed the findings of the previous work (Wang et al., 2014) that the Oktopus variant, which allocates bandwidths to tenant VMs based on mean throughput, tends to suffer with low QoS performance, whereas the Oktopus variant based on allocating peak throughput tends to waste resources.

The optimal amount of bandwidth for provisioning should lie between the mean throughput and peak throughput, however existing network-aware VM placement schemes do not determine the optimal value for the bandwidth allocation. They are designed to maximize throughput of datacenter networks, but not to deliver predictable performance in terms of the latency experienced by users of hosted applications. Our proposed system, MAPLE, provides this form of predictability by placing VMs in a tenant application's VM ensemble based on ensuring that there is sufficient effective bandwidth available between all pairs of VMs in the ensemble when they are placed on datacenter servers. Experimental results based on the test-bed have shown that MAPLE typically succeeds in meeting QoS targets whilst simultaneously allocating both computing and network resources in an efficient manner.

By default, MAPLE tends to co-locate the VMs from same ensembles. To cope with the anti-colocation requirements that are often seen in application VM ensemble placement requests, we describe MAPLE<sub>Ex</sub>, a MAPLE extension that accommodates anti-colocation constraints, which in high resource demand scenarios can lead to overprovisioning of server capacity. Our experiments show that, like MAPLE, MAPLE<sub>Ex</sub> was able to provide statistical QoS guarantees through its use of effective bandwidth estimations.

# Chapter 6

## QoS-aware Multipathing in Datacenters Using Effective Bandwidth Estimation and SDN

### 6.1 Introduction

The MAPLE system developed in Chapter 5 proposed a solution that can optimally provision datacenter resources while meeting the QoS constraints. This solution is based on network-aware VM placement, however after the initial placement, the system can hardly control the flows on its VMs, if there are any dramatic changes in their workloads overtime, the applications hosted on those VMs will suffer performance issues. For long-term scenarios, the system may migrate the affected VMs to resolve or mitigate the performance hit, nevertheless this may not be ideal as VM migration is relatively expensive and may not always be feasible for some application VMs. Comparing to VM migration, for handling short-term workload dynamics, a more responsive approach is to deploy a flow scheduler in place to balance the workloads.

There is a gap between existing network-aware VM placement techniques and flow scheduling algorithms. Most network-aware VM placement techniques are designed based on the assumption that network resources can be abstracted as a hose model (Mogul and Popa, 2012), in which an application specifies per-VM bandwidth requirements for all of components deployed in individual VMs. As long as the network requirements of VMs do not exceed the capacity specified, adequate resources can be secured and effectively

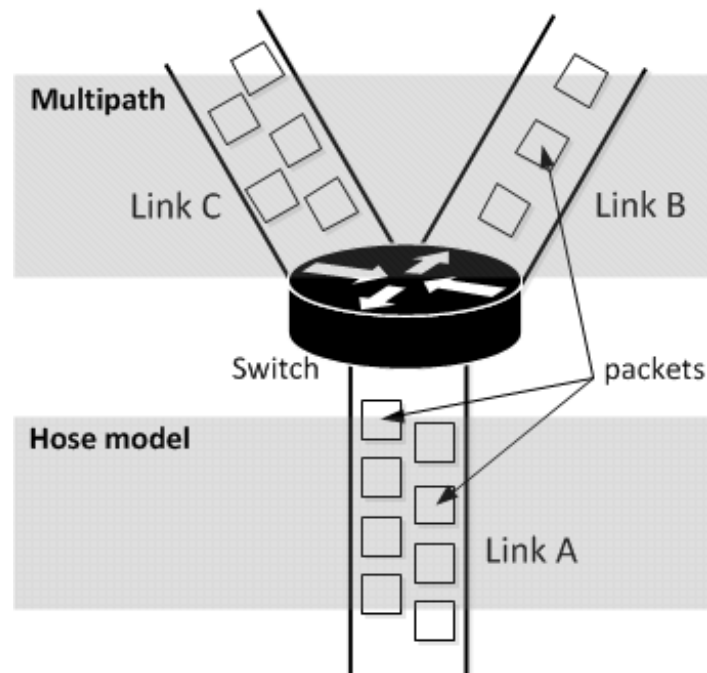


Fig. 6.1 This example shows that measuring at link A is insufficient to estimate the expected throughput of the up links, wherein the traffic packets may not be evenly distributed over the up links.

allocated. Given the multipath technologies commonly used in nowadays datacenters (Bitar et al., 2013), this assumption relies on the presence of a flow scheduler to ensure that every link has equal networking conditions, thus the hose model may not be aware if VMs use the congested links over time. ECMP (Hopps, 2000) is the common protocol used in datacenter networks to map flows into multi-equal paths; existing studies (Al-Fares et al., 2010) have shown its deficiency in the presence of larger flows, which often leads to link congestion onset. On the other hand, most proposed flow scheduling techniques ignore the individual throughput or QoS required by each VM—they only ensure that, overall, the flows are evenly allocated to the given links, but they do not directly address the network resources needed by each VM.

The aforementioned issue is found in MAPLE (Wang et al., 2016a, 2014), as well as it can be found in other resource allocation system that are based on end host measurement and end host congestion feedback mechanisms (LaCurts et al., 2013; Xie et al., 2012), as measuring at the end host level cannot accurately estimate the conditions of the associated up links. Figure 6.1 illustrates how host level traffic measurement based on the hose model

may not capture the network conditions on its up links. Link *B* and *C* represent the equal up links for flows from Link *A*. The current rate estimated on link *A* is 8 packets per second, and it is unevenly distributed to link *B* and *C*. Since host-level measurement is agnostic to the up link utilization status and can potentially lead to allocating new arrived packets to link *C* instead of link *B*, without realizing that link *C* is more congested, the performance of the hosts connected to link *A* may be negatively affected.

To address the issue, we leverage the SDN paradigm (Hu et al., 2014) to develop a flow scheduler, termed the MAPLE-Scheduler, which monitors the network changes in switches and dynamically reschedules flows to meet the QoS requirements, while balancing the link utilization across links. This QoS-aware flow scheduler is designed for incorporation with the MAPLE system. Whilst MAPLE succeeds in finding appropriate, network-aware placements of an application's VMs at the time of initial placement, it does not address the impact of subsequent changes in networking conditions. The addition of the MAPLE-Scheduler seeks to address this shortcoming by harnessing SDN to adjust flow routing in order to avoid QoS violations due to increased latency due to data center link congestion. Crucially, the MAPLE-Scheduler utilizes an empirical effective bandwidth estimation technique (Davy et al., 2008) to assess whether flows need to be re-routing in order to meet QoS targets.

The rest of this chapter is structured as follows. The concept of effective bandwidth coefficient and the motivation to use this coefficient is described in §6.2. In §6.3 and §6.4 the MAPLE-Scheduler is described and its flow scheduling algorithm is specified. In §6.5, MAPLE-Scheduler is evaluated based on a testbed constructed using 4 physical servers. Finally, §6.6 provides concluding remarks of this chapter.

## **6.2 Effective Bandwidth in multipath topologies: Effective Bandwidth Coefficient**

The previous work MAPLE measures effective bandwidth for links by relying on dedicated modules installed on end servers to monitor and regulate their traffic. To cope with multipath topologies, effective bandwidth estimation needs to be revisited in order to take into account that it is practically and computationally hard to measure and run the effective bandwidth estimation algorithm at every single link in the network. We logically divide the network in two parts—the edge and the core—and accordingly use two different approaches to effective



bandwidth estimation. At the edge of the network, where top-of-rack switches (ToR) are, we apply the precise effective bandwidth estimation method for every up link port connecting the ToR switch to the core. Since the ToR switch aggregates traffic coming from the host and the VMs in the same rack, it is practically feasible to make the link selection (flow scheduling) decision informed by an accurate estimation of the effective bandwidth supported by the ToR switch.

For the switching layers forming the core of the datacenter network, we employ the approach of Effective Bandwidth Coefficient (EBC) to estimate the effective bandwidth for the observed links. The Effective bandwidth coefficient is a concept introduced in (Davy et al., 2007), which captures a relationship between estimated effective bandwidth and its corresponding mean throughput of a link regarding a specified QoS target. Let  $R_{eff,i}$  represent the effective bandwidth of a given link  $i$ ,  $mean_i$  is the current mean throughput of the link, the corresponding EBC can be calculated as:

$$EBC_i = \frac{R_{eff,i}}{mean_i} \quad (6.1)$$

The EBC can be calculated offline from collected packet traces and then used to estimate the effective bandwidth required by a given link for a given QoS target (see (Davy et al., 2007) for a full description of this process. Letting  $mean_i$  denote the mean throughput of link  $i$ , the required effective bandwidth  $R_{eff,i}$  for link  $i$  can be calculated as:

$$R_{eff,i} = EBC_i \times mean_i \quad (6.2)$$

The process to capture traffic traversing a switch and subsequently performing effective bandwidth estimation that can be assigned to a dedicated VM. Moreover, given the prevalence of virtual switches in modern datacenters, capturing traffic traversing such virtual switches can be achieved by mirroring network traffic generated by VMs to a dedicated VM to calculate effective bandwidths. As shown in Figure 6.2, this dedicated VM keeps observing and collecting packet traces for each type of traffic and QoS target in order to calculate effective bandwidths and periodically update EBC. In §6.5, we carry out our experiments on an emulated datacenter, where the traffic measurement approach depicted in 6.2 is applied to capture traffic traversing on edge switches instantiated by using open vSwitch (Pfaff et al., 2009).

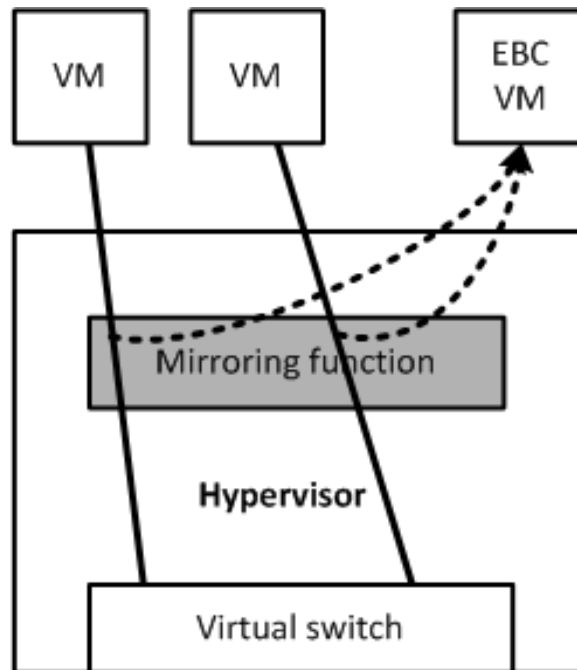


Fig. 6.2 Capturing traffic on virtual switch, the hypervisor can mirror the traffic originated from VMs to a dedicated VM that calculate effective bandwidth.

### 6.3 MAPLE-Scheduler

The MAPLE-Scheduler is a flow scheduling system that monitors flows' QoS performance, and dynamically reschedules flows, aiming to maintain their QoS needs. WE assume that ECMP is used as the default function to map new flows to equal paths, whilst the MAPLE-Scheduler monitors the network status and provide flow rescheduling when it is needed. This means after the initial VM placement, ECMP will manage routing new flows from source VMs to destination VMs, when the flows first arrive to the switches and no associated flow rules can be applied. If existing flow rules can be applied, the switch will apply them to route the flows. Meanwhile, the MAPLE-Scheduler monitors the QoS changing over time and reschedules some flows to mitigate QoS degradation. This rescheduling essentially means that some higher bandwidth flows that are contributing to congestion on a link are reschedule to another link with higher residual bandwidth. This rescheduling is controlled such that flows are re-routed through links with sufficient residual bandwidth to meet their effective bandwidth requirements and in a manner that ensures that there is sufficient capacity on the original link to meet the effective bandwidth requirements of those flows that were not reschedule. The key components of MAPLE-Scheduler are:

- An centralized controller built on SDN technologies (for our prototype the Ryu (Ryu Community, 2014) controller is used);
- A flow scheduling algorithm making schedule decisions based on a combination of different measuring mechanisms at the edge and at the core of the network, both relying on effective bandwidth estimations;
- Distributed monitoring agents residing on servers and edge switches (which do not require any modifications to hardware or protocols in the network devices).

The MAPLE-Scheduler supports fat tree like topologies, which are typically used in datacenter networks (Bitar et al., 2013). In theory, it can be extended to support any kind of multipath network topology, as long as the requirements for network to be composed of OpenFlow enabled switches and for traffic being captured and analyzed are met. The MAPLE-Scheduler leverages the separation of control plane and data plane, as stated by the SDN paradigm: switches only forward packets according to the rules installed by the logically centralized controller. The MAPLE-Scheduler is always aware of the network topology and of the resources consumed by each running flow, by using its distributed monitoring module.

The MAPLE-Scheduler achieves end-to-end dynamic QoS-aware switching by performing a two stage flow scheduling, as shown in Figure 6.3. This figure abstracts datacenter network into two layers: the edge layer and core layer. It schedules flows based on the availability of sufficient residual bandwidth, calculated based on estimations of effective bandwidth. At the edge layer, effective bandwidths are estimated based on captured traces, in which case the residual bandwidth is calculated using Eq. 6.3. For other switches, the MAPLE-Scheduler first queries the flow statistics on switches using the Openflow library, and obtains the effective bandwidth coefficient from the MAPLE system. Once all the required values are available, the residual bandwidth is calculated using Eq. 6.4:

$$\text{residual\_BW} = \text{link\_capacity} - \text{effective\_BW} \quad (6.3)$$

$$\text{residual\_BW} = \text{link\_capacity} - \text{EBC} \times \text{mean\_rate} \quad (6.4)$$

The different approaches for the edge and the core mean that we need only deploy EB agents on edge switches to capture traffic in order to calculate the effective bandwidths. For the other switches that are not associated with EB agents, the effective bandwidths are estimated

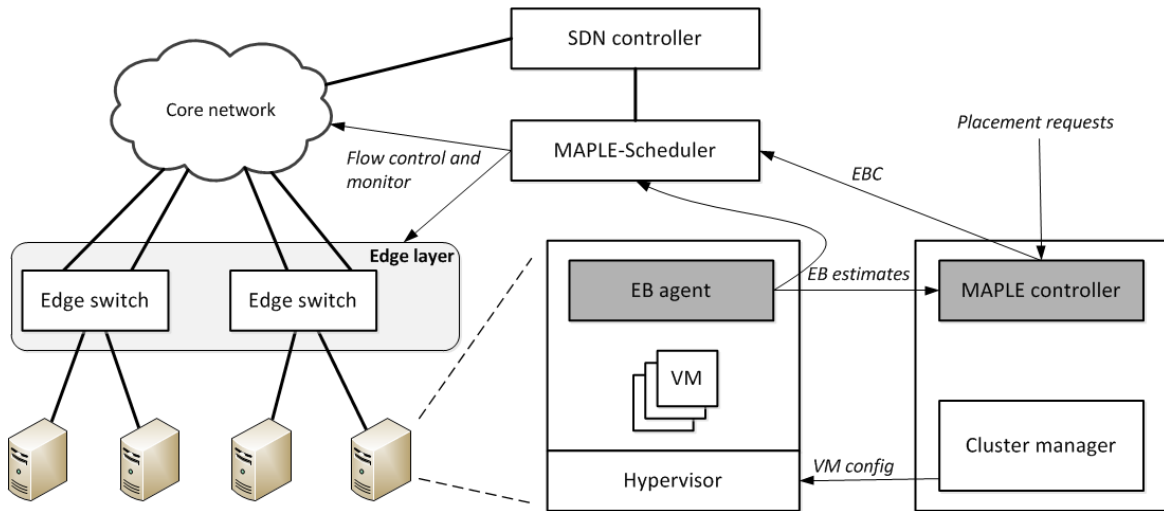


Fig. 6.3 The logical architecture of MAPLE-Scheduler and MAPLE system.

by using Eq. 6.1, where the effective bandwidth coefficient is learned from previous traces, and the mean throughput can be queried from the flow statistics available to SDN controller. We deploy EB agents on Edge switches to calculate the corresponding effective bandwidths in order to obtain more accurate estimations, as they provide first accessing points to the network, which are required to be managed more accurately. A similar design can be found in the Conga architecture (Alizadeh et al., 2014). The module performing EBC estimation is deployed on the MAPLE controller, which implements the techniques described in (Davy et al., 2007). It communicates with switches to collect the related datasets and then perform the related training described in (Davy et al., 2007) to obtain EBC regarding different QoS and applications.

The flow statistics of edge switches are learned by the distributed monitoring agents who capture traffic packets and perform the corresponding analysis. For the other network information, MAPLE-Scheduler builds the related information by using the OpenFlow protocol. Generally there are two types of methods, namely *pull based* and *push based*, for SDN controllers to gather flow statistics from the OpenFlow switches. In the proposal of DevOFlow (Curtis et al., 2011), the technical comparison of these two approaches is discussed. In this work we implemented a trigger based push method for gathering flow statistics, with the aim to reduce the work loads and enable the controller to be timely responsive to state changes in switches. Basically, trigger based method means that the OpenFlow switches update statistics to the central controller only if the switches determined it is necessary to do so. The related function is implemented on the monitoring agents, which monitor if there any large flows appear on the monitored switches and if there occurs any

Table 6.1 Before rescheduling.

Links	Flow Rates (Mbps)	Residual Bandwidth (Mbps)
Link A	[70,40,30,30]	20
Link B	[40,30,20]	100

QoS violations. The monitoring agents will notify the controller to consider rescheduling flows if it observes QoS violation. In other words MAPLE-Scheduler will be only activated for scheduling large flows when QoS violations occur.

## 6.4 MAPLE Flow Scheduling

The flow scheduling algorithm is designed to work on fat tree topologies, but it is agnostic of the topologies as long as the given topology has a hierarchical tree like structure.

The MAPLE-Scheduler works in parallel with ECMP; it only reschedules large flows with the aim to maintain QoS targets. It reschedules a flow based on two conditions: 1) if it will not exceed the selected link's residual bandwidth, based on the estimations using effective bandwidth; and 2) that max-min fairness is met—max-min fairness is applied to achieve fair loads of traffic distributed across a multipath network. By meeting these two conditions, this algorithm in turns provides two advantages over ECMP alone: better QoS and better load balancing. When these two conditions conflict with each other, we choose to meet the QoS requirement over load balancing. Note that, given all the feasible flow schedules, we consider a flow schedule achieve max-min fairness if the flow schedule maximizes the residual bandwidth of the link that has minimum residual bandwidth.

Here we use an example to show MAPLE-Scheduler algorithm attempts to reschedule a flow between two links. Table I presents the current states of Link A and Link B. As shown in the third column on table I, Link B currently has more residual bandwidth. Link A has 4 flows and Link B has 3 flows, the rates of which are respectively given on the second column. We can see the maximum flow rate on Link A is 70 Mbps, for load balance, the algorithm attempts to move this flow to Link B.

Table II presents the expected states of the two links, after moving the flow with rate 70 Mbps to Link B. From the third column on Table II, we see that the residual bandwidth of

Table 6.2 After rescheduling.

Links	Flow Rates (Mbps)	Residual Bandwidth (Mbps)
Link A	[40,30,30]	90
Link B	[70,40,30,20]	30

Link A becomes 90 Mbps, while the residual bandwidth of Link B is 30 Mbps. Now we can see in Table I the minimum residual bandwidth over all links is 20 Mbps, while in Table II the minimum residual bandwidth over all links is 30 Mbps, thus after moving a flow from Link A to Link B, the minimum residual bandwidth is increased, the MAPLE-Scheduler will reschedule the related flow to Link B in order to maximize the minimum residual bandwidth.

Note that the MAPLE-Scheduler algorithm only considers moving large flows, where we apply the definition of large flow from Hedera (Al-Fares et al., 2010), i.e. a flow is large flow if it consumes more than  $\alpha = 10\%$  of its hosting link's capacity.

Algorithm 6 presents the pseudo code of the MAPLE flow scheduling algorithm. The input is a list of links on the switch providing equal paths to flows, with the assumption that all links have the same capacity. Algorithm 6 loops the flows on each links to check if it is suitable to move a given flow from the current link to its equal link.

Line 1 sorts all the links in an increasing order, thus subsequently the links with less residual bandwidth (denoted as *remainBW* in Algorithm 6) will be proceeded first. Line 2 sets  $i = 0$  to select flows to be rescheduled from the link with smallest residual bandwidth at first, while in line 3 setting  $j$  to *Links.length*, attempting to find a feasible place to move in the selected flows into the link that has largest residual bandwidth. *LinkA* represents the link that has flows to be moved out, and *LinkB* represents the link attempted to be moved in flows.

Line 7 makes a condition to stop the algorithm earlier, if the given link has remaining bandwidth greater than  $\beta\%$  of the link capacity (we use  $\beta = 30$  in our experiments). The algorithm can return at this point as all the rest links will only have greater (or at least equal) remaining bandwidths. This is an optional condition that operators can change based on their need, our justification is that it is barely necessary to reschedule any flows for a link that still has  $\beta\%$  capacity left, considering that the possibility of this link to be filled up immediately is low enough (e.g., a large flow generally consumes 10% capacity, and only 20% flows are large flows).

**Algorithm 6** MAPLE Flow Scheduling Algorithm**Input:** *Links*, *Capacity*,  $\alpha$ ,  $\beta$ **Output:** updated links

---

```

1: sort  $\uparrow$  Links based on the remaining bandwidths
2:  $i = 0$ ,
3:  $j = \text{Links.length}$ 
4: for  $i$  in  $[0, \text{Links.length})$  do
5:    $A\text{Link} = \text{Links}[i]$ 
6:    $B\text{Link} = \text{Links}[j]$ 
7:   if  $A\text{Link}.remainBW > \beta * \text{Capacity}$  then
8:     return
9:   end if
10:  for  $flow$  in  $A\text{Link}.flows$  do
11:    if  $flow.rate < \alpha * capacity$  then
12:      continue
13:    end if
14:     $x = A\text{Link}.remainBW - flow.rate$ 
15:     $y = B\text{Link}.remainBW + flow.rate$ 
16:    if  $x < y$  then
17:      move flow to BLink
18:       $B\text{Link}.remainBW = y$ 
19:      break
20:    end if
21:  end for
22:   $j = j - 1$ 
23: end for

```

---

Line 14–16 present the condition that a flow should be moved from its current link to an equal link. This condition require a flow schedule should meet the max-min fairness.

Link 15 updates a link’s residual bandwidth, once it is determined to host a new flow. Thus in a later iteration, it will not be incorrectly added in another flow. However, we do not update the order of the sorted flow list, the iteration will just need to continue on until all the observed flows are revisited.

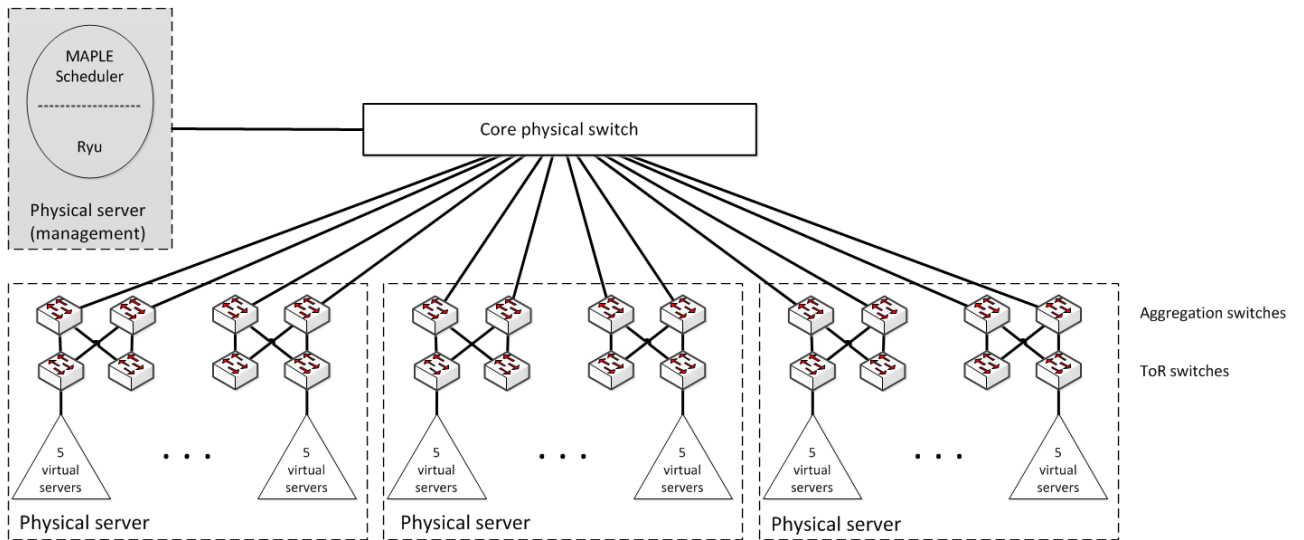


Fig. 6.4 The emulated datacenter has a single root switch connecting 4 physical servers. One server is used as the management server; the rest servers hosting 210 VMs that are connected by 2 levels of vSwitches.

## 6.5 Evaluation

### 6.5.1 Experimental Setup

We emulated a fat tree datacenter network using 4 physical servers deployed as our testbed, as shown in Fig. 6.4. One server is reserved to be the management server where the SDN controller and MAPLE-Scheduler are deployed and connected to a separate management subnet, while each of the rest servers virtualize 2 pods in a fat tree topology. Switches within the pods are Open vSwitch (Pfaff et al., 2009) instances with 1 Gbps upstream and downstream capacity. Each Virtual Switch (vSwitch) is connected to our MAPLE scheduler that is implemented based on the Ryu framework (Ryu Community, 2014). Each server runs Ubuntu 12.04 as the host operating system, with the `libvirt` library (Libvirt Community, 2005) being used to manage up to 70 guest VMs deployed on each physical server. Among these VMs, we group 5 VMs together to form a virtual server (vServer), wherein all VMs are directly connected to a vSwitch, to emulate the scenarios that 5 VMs share one bottleneck link within a physical server.

To emulate bulk data transfers within the network, we used the following data applications: 1) a Hadoop application that runs word count on distributed documents; 2) a data serving application that combines YCSB (data client) and Cassandra (data server), a benchmark



recommended in cloudsuite(Ferdman et al., 2012); finally, 3) a custom data backup application, created using the SCP utility, which simulates the scenario where a data file is copied to a couple of remote nodes for backup purposes. The applications are placed into the testbed with their corresponding VMs by the MAPLE VM placement algorithm, according to some pre-defined VM placement requests.

To simulate large flows that effectively change the QoS performance, we created a cross traffic generation program based on iperf, which runs on a dedicated VMs to simulate background traffic (this generates iperf sessions lasting 10 seconds) to some randomly selected VMs. It is necessary to have this cross traffic generation program, given that in the original setup there are not many flows have size much larger than the others, due to all applications have similar configurations.

In order to have the accurate traffic samples, we operated some preprocessing to the traffic collections. First, we noticed that a long interval of traffic capturing often results in more irrelevant captured traffic, penalizing the subsequent analysis. Thus, we use shorter and more frequent traffic capturing sessions. Second, some captured traces contain small traffic in size or duration, thus we exclude those traffic traces that have sizes smaller than 10 MB or connection duration under 2 seconds, as those traces barely provides valuable information to the analysis.

### 6.5.2 The Analysis of QoS Violations

We analyse QoS violations for the flows generated over the emulated datacenter (EDC). The QoS violation rate is calculated based on individual server's egress links. Namely, the violation rate is the proportion of a server's outgoing packets that experience delays longer than that specified in the QoS target. We employ two metrics: 1) the *overall violation rate* as defined in Eq. 6.5, which measures the overall performance; and 2) the *averaged violation rate* as defined in Eq. 6.6, which measures how strong the violations are on affected links.

$$\text{overall\_violation\_rate} = \frac{\text{sum}(\text{QoS\_violation\_rates})}{|\text{all\_links}|} \quad (6.5)$$

$$\text{averaged\_violation\_rate} = \frac{\text{sum}(\text{QoS\_violation\_rates})}{|\text{links\_with\_violations}|} \quad (6.6)$$

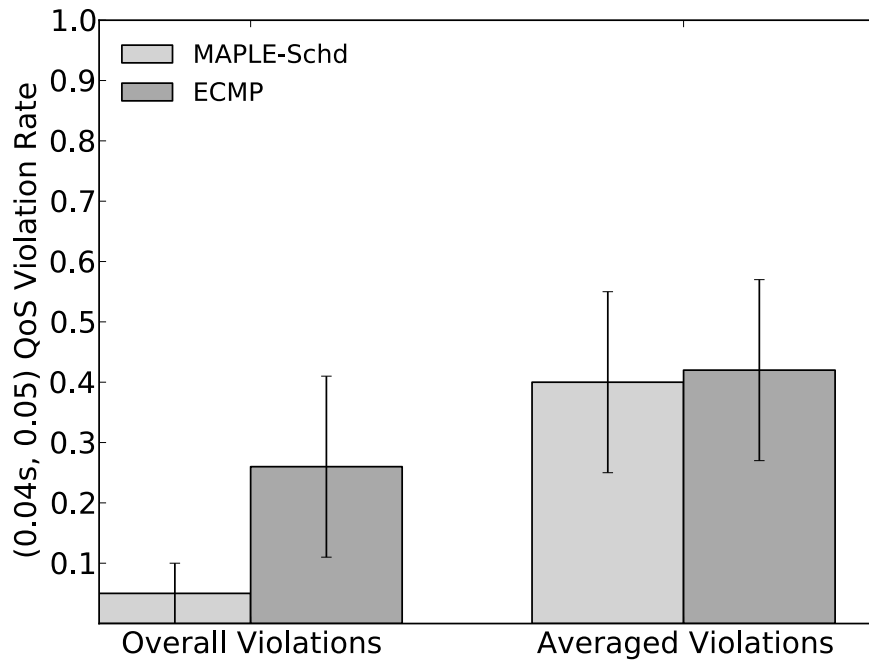


Fig. 6.5 Given the QoS target (0.02s, 0.1), MAPLE-Scheduler generally meets the QoS target, although for the links that are not able to meet the target, they tend to have large violation rates  $\approx 60\%$ . On contrast, ECMP cannot meet the target at all.

Fig. 6.5 depicts the QoS violation results incurred respectively by MAPLE using MAPLE-Scheduler and MAPLE using ECMP for the QoS target (0.02s, 0.1). Initially, VMs are placed in the EDC based on the QoS requirement: the initial QoS measurement for this stage is not shown here, MAPLE could achieve the QoS guarantee. However, once we turned on the cross traffic generation program, MAPLE using ECMP suffered increasing packet delays, as shown in Fig. 6.5, leading to an overall violation rate up to  $\approx 35\%$ , exceeding the requirement that only 10% can be delayed more than 0.02s. On the contrary, MAPLE-Scheduler once detected the QoS degradations, it managed to reschedule the flows to keep overall violation rates within the targets. The difference is also evident in the averaged violation rates, where MAPLE using ECMP shows *slightly* higher violation rates on the affected links.

Fig. 6.6 presents the second set of experiments comparing MAPLE using MAPLE-Scheduler and MAPLE using ECMP based on a new QoS target (0.04s, 0.05). This set of experiments show similar results, MAPLE-Scheduler is able to keep the overall violation rates within required range, where ECMP overall has approximately 30% traffic suffered violations. Regarding the links do suffer violations, their averaged rates come close, yet

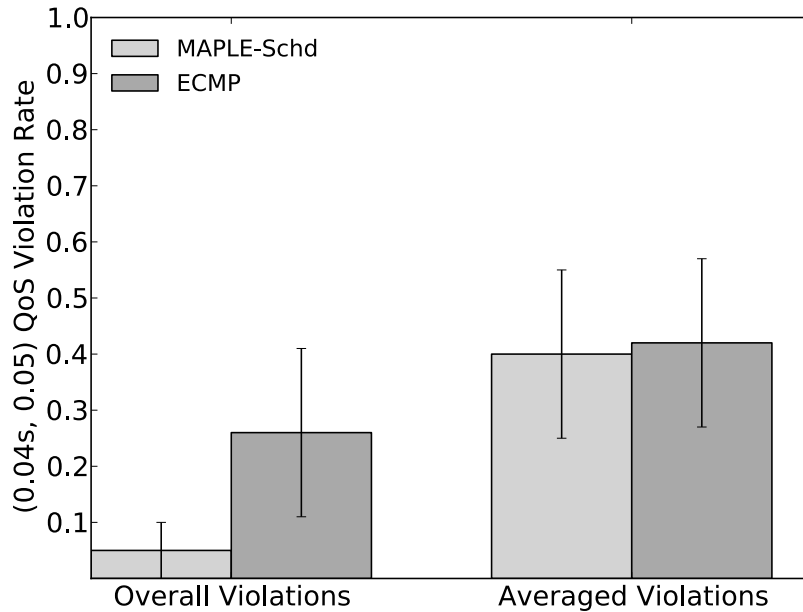


Fig. 6.6 Given the QoS target (0.04s, 0.05), MAPLE-Scheduler generally meets the QoS target; for the links that suffer QoS violation have 45% violation rate on average. On contrast, ECMP cannot meet the target at all.

MAPLE-Scheduler still has *slightly* smaller violation rates. Comparing to the previous Fig. 6.5, all the involved techniques received relatively less violation samples, though which is mainly due to the relaxed delay requirement.

### 6.5.3 Analysis of Throughput Performance and Link Utilization

To evaluate the efficiency of MAPLE-Scheduler's flow scheduling, we analyze the throughput performance of the servers and the link utilization. Note that MAPLE-Scheduler reschedules flows in order to maintain the delay QoS targets, delivering better throughput or better link utilization is not the main goal, though it is interesting to see MAPLE-Scheduler's performance on these metrics. Again, all the throughput are evaluated based on the servers' egress traffic, thus only the VM-to-VM traffic traversing inter-servers or inter-racks are evaluated, i.e., the traffic between VMs within same servers are excluded.

Fig. 6.7 presents the throughput performance for the test cases related to first QoS target (0.02s, 0.1). In general, MAPLE with MAPLE-Scheduler shows significantly better throughput performance comparing to MAPLE with ECMP. Though in terms of the

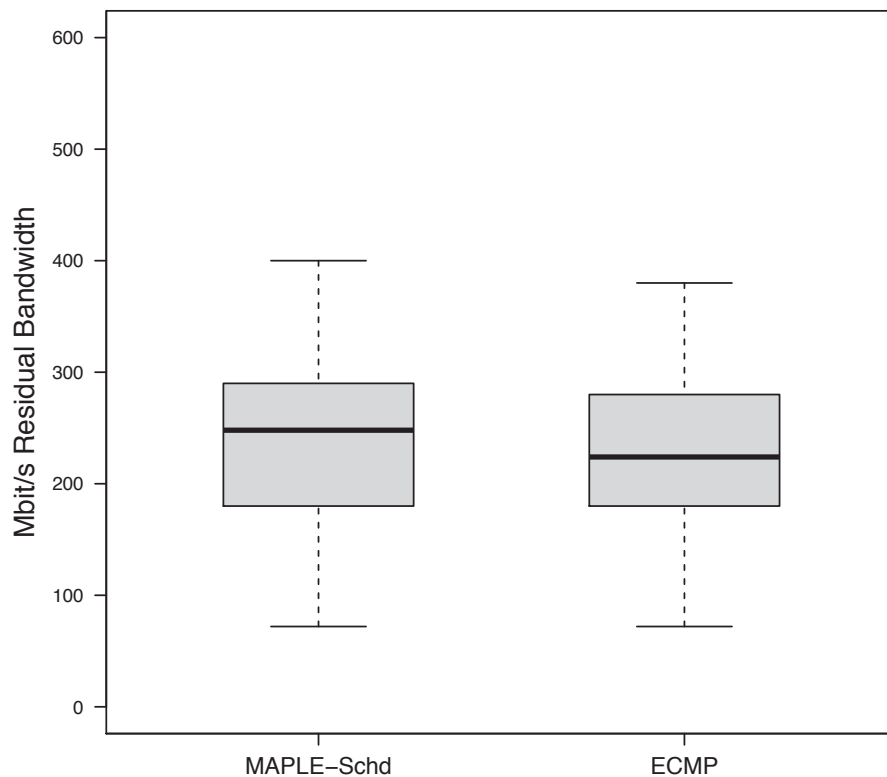


Fig. 6.7 Given the QoS target (0.02s, 0.1), MAPLE-Scheduler achieves to meet the QoS target as discussed previously, while also delivers significantly better throughput comparing to ECMP.

minimum throughput, there is no difference, they all down to 80 Mbps, while the median throughput for MAPLE-Scheduler is approximately 260 Mbps and for MAPLE using ECMP is approximately 240 Mbps. The CDF of mean link utilization levels averaged across the duration of an experimental run, presented in Fig. 6.9, indicates that the comparing techniques also show similar link utilizations, yet MAPLE-Scheduler shows slightly lighter loads on its links.

Fig. 6.8 presents the throughput performance for the test cases related to first QoS target (0.04s, 0.05). Similar to previous experiments, MAPLE using MAPLE-Scheduler shows significantly better throughput performance. The minimum throughput is consistent, in that for all cases they all go down to approximately 80 Mbps. In Fig. 6.8, the difference in median throughput is larger, MAPLE-Scheduler produced approximately 260 Mbps as the

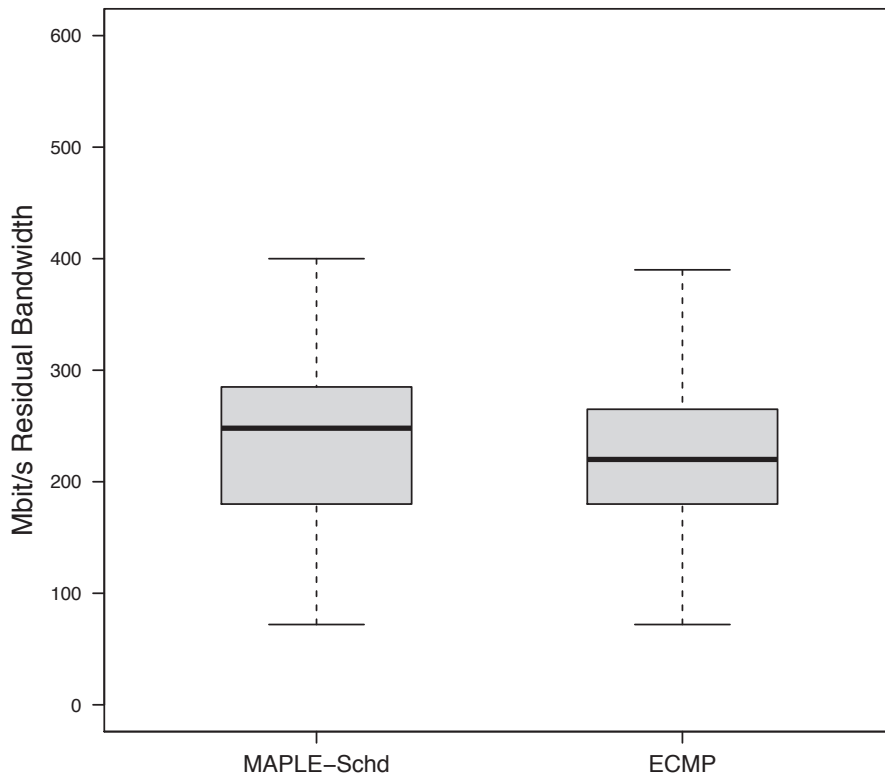


Fig. 6.8 Given the QoS target (0.04s, 0.05), MAPLE-Scheduler achieves to meet the QoS target as discussed previously, while also delivers significantly better throughput, compared to ECMP.

median, while MAPLE using ECMP produced approximately 220 Mbps. The CDF of mean link utilization levels, averaged across the duration of an experimental run, presented in Fig. 6.10, shows that a relatively bigger gap in the utilization rates up to 40%, regarding MAPLE-Scheduler, there is about 70% of links have utilization rates smaller than 40%, while regarding MAPLE using ECMP, there is about 59% of links have utilization rates smaller than 40%.

## 6.6 Summary

The technique proposed in this chapter addresses the issue that the QoS and network aware VM placement system like MAPLE has no control to the VMs' flows after initial placement

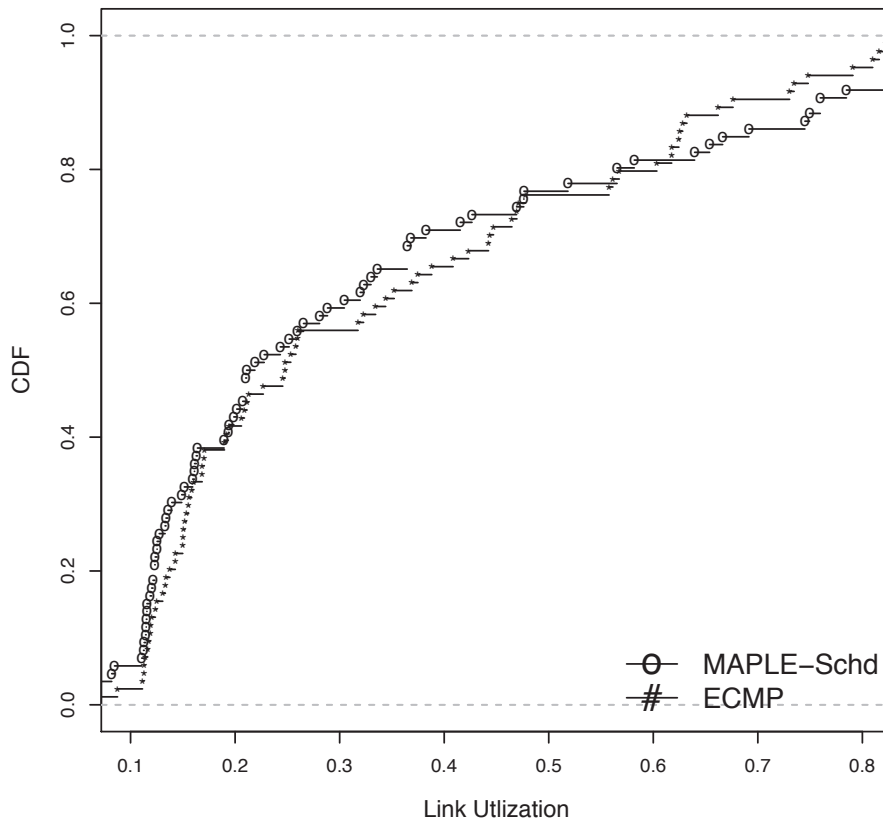


Fig. 6.9 Both MAPLE-Scheduler and ECMP show similar link utilization, as shown by their CDF of the mean link utilization, where they all have 80% of links that have utilization lower than 60%, wherein MAPLE-Scheduler has slightly lighter loads.

of those VMs. Common load balancing mechanism (e.g., ECMP) that perform static mapping of flows to multi equal paths does not consider link utilization status and QoS requirements, where large flows can collide on same paths, making network links congested and not suitable for latency sensitive flows. In order to take into account of QoS targets, this work designed MAPLE-Scheduler built on SDN controller, which can effectively schedule flows to enforce QoS guarantee in multipathing datacenters. In the experiments based on our SDN testbed consisting of 6 pods of multipaths, the experimental results show that, when large flows arriving into the networks, MAPLE using ECMP cannot maintain the delay requirement, while MAPLE using MAPLE-Scheduler can effectively reschedule some appropriate flows to keep the delay performance within the QoS targets.

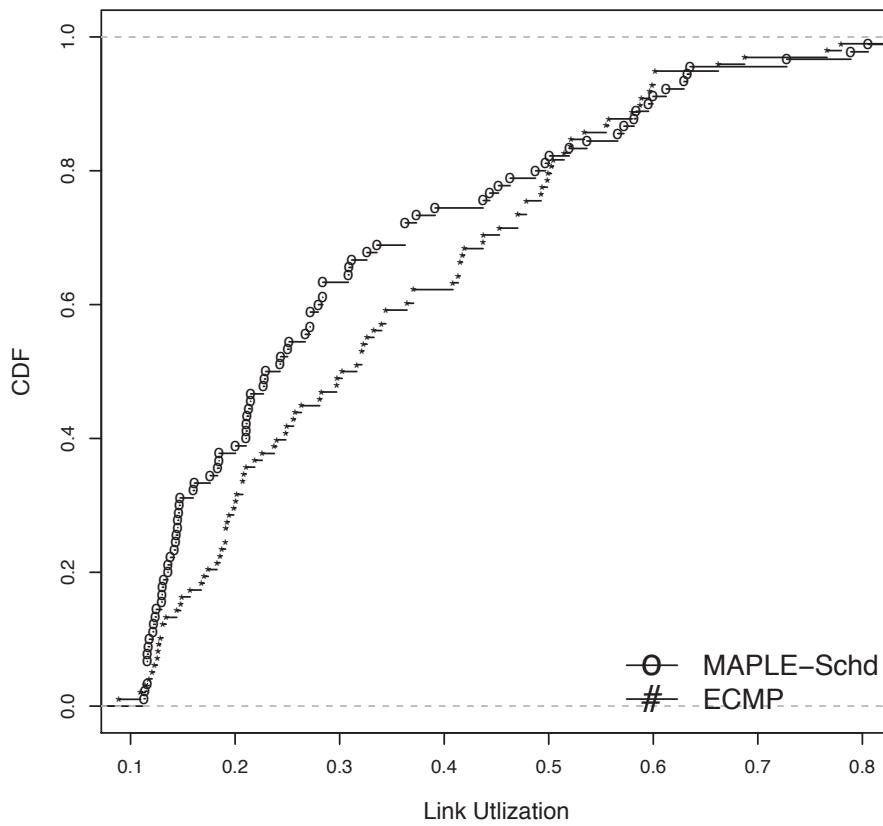


Fig. 6.10 The gap between two lines on the range of link utilization between 0.2 to 0.4 that indicates MAPLE-Scheduler achieve lighter loads across its links, comparing to ECMP.

# Chapter 7

## Conclusions & Future Work

This thesis addressed how to perform effective network management through using traffic analysis. The previous chapters demonstrated the benefits of using traffic analysis through addressing four specific problems in traffic classification and resource allocation in datacenters. The traffic classification chapters considered grouping network packets into flows, so that useful statistics on flow level can be learnt. These flow statistics can then be used with appropriate analytical techniques that can provide effective predictions and estimations to leverage the involved network management applications. In traffic classification, we see that the use of advanced traffic analysis techniques leads to more accurate prediction of the source applications of traffic, even in challenging conditions. In resource allocations, we see that the use of traffic analysis that provides estimation of resource demands can lead to optimal resource allocation while maintaining QoS targets. We now summarize the main contributions presented in this thesis.

Chapter 3 studied the flexibility of preparing flow statistics to implement traffic classification. Comparing to the existing techniques relying on flow statistics learned from first 5 packets or sets of consecutive packets, we investigated the flexibility of using arbitrary packet sets to train traffic classifiers. Our proposed classifiers could be used as auxiliary classifiers that are able to function in cases where first 5 packets or consecutive packets are not available, due to possible packet losses/retransmissions. This proposed technique also mitigates the issue that payload mutation techniques are used to evade classification.

Chapter 4 studied the issue that flow statistics can be changed over time. Due to the rapid development on today's network based applications, traffic classifiers accurately trained at a particular time can suffer degraded accuracy subsequently. We proposed an adjustable traffic



classification system, in which the key technique is an ensemble learning technique, assisted by a change detection method. This proposal enables traffic classifiers to be effectively updated in response to the changing traffic distributions.

Chapter 5 presented the MAPLE system, a network-aware VM placement system for datacenters hosting multi-tenant applications. MAPLE applies the empirical effective bandwidth estimation technique to optimally provision datacenter-hosted applications. We observed that the optimal amount of bandwidth for provisioning datacenter applications should lie between the mean throughput and peak throughput. Existing network-aware VM placement techniques do not determine this optimal value for the bandwidth allocation. They are designed to maximize throughput of datacenter networks, but not to deliver predictable performance in terms of the latency experienced by users of hosted applications. In contrast, MAPLE provides this form of predictability by placing VMs in a tenant application's VM ensemble based on ensuring that there is sufficient effective bandwidth available between all pairs of VMs in the ensemble when they are placed on datacenter servers.

Chapter 6 presented MAPLE-Scheduler, a latency-aware flow scheduling system that schedules flows based on QoS requirements from tenant applications. MAPLE-Scheduler is developed based on the use of an SDN controller. By using effective bandwidth, it can effectively schedule flows to enforce QoS guarantee in multipathing datacenters. The MAPLE system typically as a VM placement system has no control to the VMs' flows after initial placement of those VMs. Common load balancing mechanisms that perform static mapping of flows to multi equal paths do not consider link utilization status and QoS requirements, so that large flows can collide on same paths, making network links congested and not suitable for latency-sensitive flows. MAPLE-Scheduler is a complement to MAPLE that ensures the QoS requirements can be met over time while the workload changed after the initial placement.

## 7.1 Future Work

The key question driving this research is how to utilize traffic analysis techniques to realize effective network management. Previous chapters have shared the lessons and experiences of using and deploying traffic analysis techniques in enterprize and datacenter networks. On one hand, we believe there are some aspects that can be improved for the work described in this thesis; on the other hand, from a longer term prospective, we see potential to develop a more complete conceptual framework for traffic analysis based network management. It will

be useful to identify the reusable models for network data collection and analysis, and provide technical guidelines to address potential issues could arise from practical applications for network operators who are interested to apply traffic analysis to manage their networks.

In the near future, we seek to enhance the current proposed techniques from the below directions:

- **Traffic Classification Accuracy.** Some traffic of particular applications are more difficult to be identified than the others; our proposed traffic classifiers presented in chapter 3 and chapter 4 respectively show relatively lower accuracies on those applications. In addition, the two different issues are addressed respectively in different proposals presented in separate chapters, however in practice the two issues could co-exist. It will be interesting to implement integrated techniques that can address the presented issues and evaluate the accuracy on some more complicated scenarios.
- **Traffic Classification on SDN.** Some experimental research of integrating traffic classification with SDN has been reported in the literature (Ng et al., 2015). The programmable controller in SDN provides a flexible way to implement traffic classification. In particular regarding the traffic flows that are hard to be classified, controller can keep observing the flows and re-classify them once sufficient information obtained. The SDN controller also enables possibility to implement hybrid traffic classifier, where classification are dynamically made using different classification models, and deep packet inspection classifiers and port based classifiers can be together used to accelerate the classification for some traffic classes.
- **Elastic Network-Aware VM Placement.** For the MAPLE system described in chapter 5 and 6, we plan to extend MAPLE to support elastic VM ensembles in which constituent VMs can be instantiated and/or re-provisioned on-the-fly to handle growing application demands. One limitation of MAPLE is that the VM placements it makes, whilst being appropriate at the time the placements are made, may lead to sub-optimal placements over time, especially if VM ensembles are deployed for extended time periods. We plan to explore how MAPLE could be extended to periodically identify VM migrations that would result in more optimal global placements. We also intend to explore the use of effective bandwidth estimation in the placement of more complex ensembles which include (virtual) network appliances (for

example, load balancers and intrusion detection appliances) as well as application VMs.

- **Effective flow scheduling.** MAPLE-Scheduler schedules flows based on the use of ECMP before the flows become elephant flows. Is there a more proactive way to allow bandwidths for elephant flows? Accurate classification of datacenter flows can provide good indications to whether given flows would become elephant flows, since some applications (e.g., big data applications) tend to have higher possibility to consume more bandwidth and last longer. Flow scheduling based on application types is an interesting research direction, for which good accuracy of flow identifications is required.

# Bibliography

- (2010). Scalable flow-based networking with DIFANE. In *Proceedings of the ACM SIGCOMM*, pages 351–362.
- Al-Fares, M., Radhakrishnan, S., Raghavan, B., Huang, N., and Vahdat, A. (2010). Hedera: dynamic flow scheduling for data center networks. In *Proc. NSDI*, pages 11–19.
- Alizadeh, M., Yadav, N., Varghese, G., Edsall, T., Dharmapurikar, S., Vaidyanathan, R., Chu, K., Fingerhut, A., Lam, V. T., Matus, F., and Pan, R. (2014). Conga. *Proceedings of the 2014 ACM conference on SIGCOMM - SIGCOMM '14*, pages 503–514.
- Amazon Inc. (2006). Amazon EC2. <http://aws.amazon.com/ec2/instance-types/>.
- Ballani, H., Costa, P., Karagiannis, T., and Rowstron, A. (2011). Towards predictable datacenter networks. In *Proc. ACM SIGCOMM*, pages 242–253.
- Benson, T., Akella, A., and Maltz, D. A. (2010). Network traffic characteristics of data centers in the wild. In *Proc. ACM IMC*, pages 267–280.
- Benson, T., Akella, A., Shaikh, A., and Sahu, S. (2011a). CloudNaaS: a cloud networking platform for enterprise applications. In *Proc. ACM Symposium on Cloud Computing (SOCC)*, pages 8:1–8:13.
- Benson, T., Anand, A., Akella, A., and Zhang, M. (2011b). Microte: Fine grained traffic engineering for data centers. In *Proc. CoNEXT*, pages 8:1–8:12.
- Bernaille, L., Teixeira, R., Akodkenou, I., Soule, A., and Salamatian, K. (2006a). Traffic classification on the fly. *SIGCOMM Comput. Commun. Rev.*, 36(2):23–26.
- Bernaille, L., Teixeira, R., and Salamatian, K. (2006b). Early application identification. In *Proc. ACM CoNEXT*, pages 6:1–6:12.
- Bin, E., Biran, O., Boni, O., Hadad, E., Kolodner, E., Moatti, Y., and Lorenz, D. (2011). Guaranteeing High Availability Goals for Virtual Machine Placement. In *Proc. ICDCS*, pages 700–709.
- Bitar, N., Gringeri, S., and Xia, T. (2013). Technologies and protocols for data center and cloud networking. *IEEE Comm. Mag.*, 51(9):24–31.
- Breitgand, D. and Epstein, A. (2012). Improving consolidation of virtual machines with risk-aware bandwidth oversubscription in compute clouds. In *Proc. IEEE INFOCOM*, pages 2861–2865.

- Callado, A. C., Kamienski, C. A., Szabo, G., Gero, B. P., Kelner, J., Fernandes, S. F. L., and Sadok, D. F. H. (2009). A Survey on Internet Traffic Identification. *IEEE Communications Surveys and Tutorials*, 11(3):37–52.
- Cheng, T., Lin, Y., Lai, Y., and Lin, P. (2012). Evasion techniques: Sneaking through your intrusion detection/prevention systems. *IEEE Communications Surveys and Tutorials*, pages 1011–1020.
- Cho, K., Mitsuya, K., and Kato, A. (2000). Traffic data repository at the WIDE project. In *Proc. annual conference on USENIX Annual Technical Conference*.
- Chowdhury, M., Zaharia, M., Ma, J., Jordan, M., and Stoica, I. (2011). Managing data transfers in computer clusters with orchestra. In *Proc. ACM SIGCOMM*, pages 98–109.
- Chowdhury, M., Zhong, Y., and Stoica, I. (2014). Efficient coflow scheduling with Varys. *Proceedings of the 2014 ACM conference on SIGCOMM - SIGCOMM '14*, pages 443–454.
- Chrysos, N., Gusat, M., Neeser, F., Minkenberg, C., Denzel, W., and Basso, C. (2014). High Performance Multipath Routing for Datacenters. pages 70–75.
- Cisco Online Document (2006). Understanding Delay in Packet Voice Networks. <http://www.cisco.com/c/en/us/support/docs/voice/voice-quality/5125-delay-details.html>.
- Claffy, K. C. (1994). *Internet Traffic Characterization*. PhD thesis.
- Courcoubetis, C., Siris, V. A., and Stamoulis, G. D. (1998). Application and evaluation of large deviation techniques for traffic engineering in broadband networks. In *Proc. ACM SIGMETRICS*, pages 212–221.
- Crotti, M., Gringoli, F., and Salgarelli, L. (2009). Impact of asymmetric routing on statistical traffic classification. In *Proc. IEEE Globecom*, pages 655–662.
- Curtis, A. R., Mogul, J. C., Tourrilhes, J., Yalagandula, P., Sharma, P., and Banerjee, S. (2011). Devoflow: scaling flow management for high-performance networks. In *Proc. ACM SIGCOMM*, pages 254–265.
- Dainotti, A., Donato, W., and Pescapè, A. (2009). Tie: A community-oriented traffic classification platform. In *Proc. TMA Workshop*, pages 64–74.
- Dainotti, A., Pescapè, A., and Claffy, K. C. (2012). Issues and future directions in traffic classification. *IEEE Network*, 26(1):35–40.
- Dainotti, A., Pescapè, A., and Sansone, C. (2011). Early classification of network traffic through multi-classification. In *Proc. TMA Workshop*, pages 122–135.
- Davy, A., Botvich, D., and Jennings, B. (2007). On the use of accounting data for qos-aware ip network planning. In *Managing Traffic Performance in Converged Networks*, chapter 33, pages 348–360.
- Davy, A., Botvich, D., and Jennings, B. (2008). Revenue optimized iptv admission control using empirical effective bandwidth estimation. *IEEE Transactions on Broadcasting*, 54(3):599–611.

- Este, A., Gringoli, F., and Salgarelli, L. (2009). Support Vector Machines for TCP traffic classification. *Comput. Netw.*, 53(14):2476–2490.
- Ferdman, M., Adileh, A., Kocberber, O., Volos, S., Alisafae, M., Jevdjic, D., Kaynak, C., Popescu, A. D., Ailamaki, A., and Falsafi, B. (2012). Clearing the clouds: a study of emerging scale-out workloads on modern hardware. In *Proc. ACM ASPLOS*, pages 37–48.
- Finamore, A., Mellia, M., Meo, M., and Rossi, D. (2010). KISS: stochastic packet inspection classifier for udp traffic. *IEEE/ACM Trans. Netw.*, 18(5):1505–1515.
- Finamore, A., Mellia, M., Meo, M., Torino, P., and Rossi, D. (2011). Experiences of internet traffic monitoring with tstat. *IEEE Network*, 25(3):8–14.
- Gill, P., Jain, N., and Nagappan, N. (2011). Understanding network failures in data centers: Measurement, analysis, and implications. *SIGCOMM Comput. Commun. Rev.*, 41(4):350–361.
- Greenberg, A., Hamilton, J. R., Jain, N., Kandula, S., Kim, C., Lahiri, P., Maltz, D. A., Patel, P., and Sengupta, S. (2009). VI2: a scalable and flexible data center network. In *Proc. ACM SIGCOMM*, pages 51–62.
- Gringoli, F., Salgarelli, L., Dusi, M., Cascarano, N., Risso, F., and claffy, k. c. (2009). Gt: picking up the truth from the ground for internet traffic. *SIGCOMM Comput. Commun. Rev.*, 39(5):12–18.
- Guerin, R., Ahmadi, H., and Naghshineh, M. (1991). Equivalent capacity and its application to bandwidth allocation in high-speed networks. *IEEE Journal on Selected Areas in Communications*, 9(7):968–981.
- Guo, C., Lu, G., Wang, H., Yang, S., Kong, C., Sun, P., Wu, W., and Zhang, Y. (2010). Secondnet: a data center network virtualization architecture with bandwidth guarantees. In *Proc. ACM CONEXT*, pages 15:1–15:12.
- Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18.
- He, K., Rozner, E., Agarwal, K., Felter, W., Carter, J., and Akella, A. (2015). Presto: Edge-based Load Balancing for Fast Datacenter Networks. *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication - SIGCOMM '15*, pages 465–478.
- Hong, C., Kandula, S., and Mahajan, R. (2013). Achieving high utilization with software-driven WAN. *Sigcomm 2014*, pages 15–26.
- Hopps, C. (2000). Analysis of an equal-cost multi-path algorithm. Internet Requests for Comments.
- Hu, F., Hao, Q., and Bao, K. (2014). A Survey on Software Defined Networking (SDN) and OpenFlow: From Concept to Implementation. *IEEE Communications Surveys & Tutorials*, 16(c):1–1.
- IANA (1972). IANA Internet Assigned Numbers Authority. <http://www.iana.org/>.

- Ihaka, R. and Gentleman, R. (1993). R programming language. <http://www.R-project.org>.
- Ishimori, A., Farias, F., Cerqueira, E., and Abelem, A. (2013). Control of Multiple Packet Schedulers for Improving QoS on OpenFlow/SDN Networking. *2013 Second European Workshop on Software Defined Networks*, pages 81–86.
- Jain, S., Zhu, M., Zolla, J., Hölzle, U., Stuart, S., Vahdat, A., Kumar, A., Mandal, S., Ong, J., Poutievski, L., Singh, A., Venkata, S., Wanderer, J., and Zhou, J. (2013). B4. *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM - SIGCOMM '13*, page 3.
- Jayasinghe, D., Pu, C., Eilam, T., Steinder, M., Whally, I., and Snible, E. (2011). Improving Performance and Availability of Services Hosted on IaaS Clouds with Structural Constraint-Aware Virtual Machine Placement. In *Proc. 2011 IEEE International Conference on Services Computing*, pages 72–79.
- Jiang, J. W., Lan, T., Ha, S., Chen, M., and Chiang, M. (2012). Joint vm placement and routing for data center traffic engineering. In *Proc. IEEE INFOCOM*, pages 2876–2880.
- Karagiannis, T., Papagiannaki, K., and Faloutsos, M. (2005). Blinc: multilevel traffic classification in the dark. In *Proc. ACM SIGCOMM*, pages 229–240.
- Kelly, F. (1996). Notes on effective bandwidths. *Stochastic Networks: Theory and Applications*, 4:141–168.
- Kim, H., Claffy, K. C., Fomenkov, M., Barman, D., Faloutsos, M., and Lee, K. (2008). Internet traffic classification demystified: myths, caveats, and the best practices. In *Proc. ACM CoNEXT*, pages 11:1–11:12.
- Kim, W., Sharma, P., Lee, J., Banerjee, S., Tourrilhes, J., Lee, S.-J., and Yalagandula, P. (2010). Automated and scalable QoS control for network convergence. *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, page 1.
- Knuth, D. E. (1997). *The art of computer programming, volume 2: seminumerical algorithms*. Addison-Wesley Longman Inc.
- Kreutz, D., Ramos, F., Verissimo, P., Rothenberg, C., and Azodolmolky, Siamakabd Uhlig, S. (2015). Software-Defined Networking: A Comprehensive Survey. *Proceedings of the IEEE*, 103(1):14–76.
- LaCurts, K., Shuo, D., Goyal, A., and Balakrishnan, H. (2013). Choreo: Network-aware task placement for cloud applications. In *Proc. ACM IMC*, pages 191–204.
- Lam, V. T., Radhakrishnan, S., Pan, R., Vahdat, A., and Varghese, G. (2012). Netshare and stochastic netshare: predictable bandwidth allocation for data centers. *SIGCOMM Comput. Commun. Rev.*, 42(3):5–11.
- Lara, A., Kolasani, A., and Ramamurthy, B. (2014). Network Innovation using OpenFlow: A Survey. *IEEE Communications Surveys Tutorials*, 16(1):493–512.
- Lee, S., Kim, H., Barman, D., Lee, S., Kim, C., Kwon, T., and Choi, Y. (2011). Netramark: a network traffic classification benchmark. *SIGCOMM Comput. Commun. Rev.*, 41:22–30.

- Lee, Y. and Lee, Y. (2012). Toward Scalable Internet Traffic Measurement and Analysis with Hadoop. *SIGCOMM Comput. Commun. Rev.*, 43(1):5–13.
- Libvirt Community (2005). Libvirt The virtualization API. <http://libvirt.org/>.
- Lim, Y., Kim, H., Jeong, J., Kim, C., Kwon, T., and Choi, Y. (2010). Internet traffic classification demystified: on the sources of the discriminative power. In *Proc. ACM CoNEXT*, pages 9:1–9:12.
- Mann, H. B. and Whitney, D. R. (1947). On a test of whether one of two random variables is stochastically larger than the other. *Annals of Mathematical Statistics*, 18:50–60.
- McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). Openflow: enabling innovation in campus networks. *ACM Comput. Commun. Rev.*, 38(2):69–74.
- Mellia, M. and Meo, M. (2010). Measurement of IPTV traffic from an operative network. *European Transactions on Telecommunications*, 21(4):324–336.
- Meng, X., Pappas, V., and Zhang, L. (2010). Improving the scalability of data center networks with traffic-aware virtual machine placement. In *Proc. IEEE INFOCOM*, pages 1154–1162.
- Mogul, J. C. and Popa, L. (2012). What we talk about when we talk about cloud network performance. *SIGCOMM Comput. Commun. Rev.*, 42(5):44–48.
- Moore, A. W. and Zuev, D. (2005). Internet traffic classification using bayesian analysis techniques. In *Proc. ACM SIGMETRICS*, pages 50–60.
- Mysore, N. R., Pamboris, A., Farrington, N., Huang, N., Miri, P., Radhakrishnan, S., Subramanya, V., and Vahdat, A. (2009). Portland: a scalable fault-tolerant layer 2 data center network fabric. In *Proc. ACM SIGCOMM*, pages 39–50.
- Nechay, D., Pointurier, Y., Coates, M., and Quebec, M. (2009). Controlling false alarm/discovery rates in online internet traffic flow classification. In *Proc. IEEE INFOCOM*.
- Ng, B., Hayes, M., and Seah, W. (2015). Developing a traffic classification platform for enterprise networks with SDN: Experiences & lessons learned. In *Proc. IFIP Networking*, pages 1–9.
- Nguyen, T. T. and Armitage, G. (2006). Training on multiple sub-flows to optimise the use of machine learning classifiers in real-world ip networks. In *Proc. IEEE LCN*.
- Openvswitch Community (2014). Rate-Limiting VM Traffic Using QoS Policing. <http://openvswitch.org/support/config-cookbooks/qos-rate-limiting/>.
- Ostermann, S. (2003). Tcptrace Homepage. <http://www.tcptrace.org/>.
- Paxson, V. and Floyd, S. (1995). Wide area traffic: the failure of poisson modeling. *IEEE/ACM Trans. Netw.*, 3(3):226–244.



- Pfaff, B., Pettit, J., Koppern, T., Amidon, K., Casado, M., and Shenker, S. (2009). Extending networking into the virtualization layer. In *Proc. ACM HotNets Workshop*.
- Popa, L., Krishnamurthy, A., Ratnasamy, S., and Stoica, I. (2012). Faircloud: sharing the network in cloud computing. In *Proc. ACM SIGCOMM*, pages 187–198.
- Popa, L., Yalagandula, P., Banerjee, S., Mogul, J. C., Turner, Y., and Santos, J. R. (2013). Elasticswitch: practical work-conserving bandwidth guarantees for cloud computing. In *Proc. ACM SIGCOMM*, pages 351–362.
- Popek, G. J. and Goldberg, R. P. (1974). Formal requirements for virtualizable third generation architectures. *Commun. ACM*, 17(7):412–421.
- Quinlan, J. R. (1993). *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc.
- Rackspace Inc. (2006). Rackspace Cloud. <http://www.rackspace.com/>.
- Raiciu, C., Barre, S., Pluntke, C., Greenhalgh, A., Wischik, D., and Handley, M. (2011). Improving datacenter performance and robustness with multipath tcp. In *Proc. ACM SIGCOMM*, pages 266–277.
- Rodrigues, H., Santos, J. R., Turner, Y., Soares, P., and Guedes, D. (2011). Gatekeeper: Supporting bandwidth guarantees for multi-tenant datacenter networks. In *Proc. USENIX WIOV*, pages 6–6.
- Roughan, M., Sen, S., Spatscheck, O., and Duffield, N. (2004). Class-of-service mapping for QoS: a statistical signature-based approach to IP traffic classification. In *Proc. ACM IMC*, pages 135–148.
- Ryu Community (2014). Ryu SDN framework. <http://osrg.github.io/ryu/>.
- Sheikhalishahi, M., Wallace, R. M., Grandinetti, L., Vazquez-Poletti, J. L., and Guerriero, F. (2014). A Multi-capacity Queuing Mechanism in Multi-dimensional Resource Scheduling. In *Adaptive Resource Management and Scheduling for Cloud Computing*, number 8907 in Lecture Notes in Computer Science, pages 9–25.
- Shi, L., Butler, B., Botvich, D., and Jennings, B. (2013a). Provisioning of Requests for Virtual Machine Sets with Placement Constraints in IaaS Clouds. In *Proc. IFIP/IEEE International Conference on Integrated Network Management (IM 2013)*, pages 499–505. IEEE.
- Shi, L., Butler, B., Wang, R., Botvich, D., and Jennings, B. (2012). Optimal placement of virtual machines with different placement constraints in IaaS clouds. In *China Ireland Symposium on ICT and Energy Efficiency (CICT)*.
- Shi, L., Furlong, J., and Wang, R. (2013b). Empirical evaluation of vector bin packing algorithms for energy efficient data centers. In *Proc. IEEE Management of Cloud Systems Workshop (MoCS)*, pages 9–15.
- Shieh, A., Kandula, S., Greenberg, A., and Kim, C. (2010). Seawall: Performance isolation for cloud datacenter networks. In *Proc. ACM HotCloud*, pages 1–1.

- Szabó, G., Orincsay, D., Malomsoky, S., and Szabó, I. (2008). On the validation of traffic classification algorithms. In *Proc. PAM*, pages 72–81.
- VMware Inc. (2014). VMware vCenter.
- Wang, H., Fan, W., Yu, P. S., and Han, J. (2003). Mining concept-drifting data streams using ensemble classifiers. In *Proc. ACM KDD*, pages 226–235.
- Wang, M., Meng, X., and Zhang, L. (2011). Consolidating virtual machines with dynamic bandwidth demand in data centers. In *Proc. IEEE INFOCOM*, pages 71–75.
- Wang, R., Araujo Wickboldt, J., Pereira Esteves, R., Shi, L., Jennings, B., and Zambenedetti Granville, L. (2016a). Using empirical estimates of effective bandwidth in network-aware placement of virtual machines in Datacenters. *IEEE Transactions on Network and Service Management*.
- Wang, R., Esteves, R., Shi, L., Wickboldt, J., Jennings, B., and Granville, L. (2014). Network-aware placement of virtual machine ensembles using effective bandwidth estimation. In *Proc. IFIP/IEEE CNSM*, pages 100–108.
- Wang, R., Mangiante, S., Davy, A., Shi, L., and Jennings, B. (2016b). Qos-aware multipathing in datacenters using effective bandwidth estimation and sdn. In *Proc. International Workshop on Management of SDN and NFV Systems (ManSDN/NFV)*.
- Wang, R., Shi, L., and Jennings, B. (2013a). Ensemble classifier for traffic in presence of changing distributions. In *Proc. IEEE ISCC*.
- Wang, R., Shi, L., and Jennings, B. (2013b). Training traffic classifiers with arbitrary packet sets. In *Proc. IEEE Workshop on Traffic Identification and Classification for Advanced Network Services and Scenarios (TRICANS)*.
- Wireshark (1998). Wireshark Homepage. <http://www.wireshark.org/>.
- Wuhib, F., Yanggratoke, R., and Stadler, R. (2015). Allocating Compute and Network Resources Under Management Objectives in Large-Scale Clouds. *Journal of Network and Systems Management*, 23(1):111–136.
- Xie, D., Ding, N., Hu, Y. C., and Kompella, R. (2012). The only constant is change: incorporating time-varying network reservations in data centers. In *Proc. ACM SIGCOMM*, pages 199–210.
- Yue, M. (1991). A simple proof of the inequality  $\text{ffd}(l) \leq 11/9 \text{opt}(l) + 1$ , for the ffd bin-packing algorithm. *Acta Mathematicae Applicatae Sinica*, 7(4):321–331.
- Zander, S. and Armitage, A. (2011). Practical machine learning based multimedia traffic classification for distributed qos management. In *Proc. IEEE LCN*.

# Appendix A

## List of Publications

- Runxin Wang, Simone Mangiante, Alan Davy, Lei Shi, Brendan Jennings. **QoS-aware Multipathing in Datacenters Using Effective Bandwidth Estimation and SDN.** *Proc. International Workshop on Management of SDN and NFV Systems (ManSDN/NFV)*, 2016.
- Runxin Wang, Juliano Wickboldt, Rafael Esteves, Lei Shi, Brendan Jennings, and Lisandro Granville. **Using empirical estimates of effective bandwidth in network-aware placement of virtual machines in datacenters.** *In IEEE Transactions on Network and Service Management*, vol. 13, pages 267-280, 2016.
- Runxin Wang, Rafael Esteves, Lei Shi, Juliano Wickboldt, Brendan Jennings, and Lisandro Granville. **Network-aware placement of virtual machine ensembles using effective bandwidth estimation.** *In Proc. IFIP/IEEE CNSM*, pages 100-108, 2014.
- Runxin Wang, Lei Shi, and Brendan Jennings. **Ensemble classifier for traffic in presence of changing distributions.** *In Proc. IEEE ISCC*, pages 629-635, 2013.
- Runxin Wang, Lei Shi, and Brendan Jennings. **Training traffic classifiers with arbitrary packet sets.** *Workshop on Traffic Identification and Classification for Advanced Network Services and Scenarios (TRICANS)*, pages 1314-1318, 2013.
- Lei Shi, John Furlong, and Runxin Wang. **Empirical evaluation of vector bin packing algorithms for energy efficient data centers.** *In Proc. IEEE Management of Cloud Systems Workshop (MoCS)*, pages 9-15, 2013.

- Lei Shi, Bernard Butler, Runxin Wang, Dmitri Botvich and Brendan Jennings.  
**Optimal Placement of Virtual Machines with Different Placement Constraints in IaaS Clouds.** *In Proc. China Ireland Symposium on ICT and Energy Efficiency (CICT)*, 2012.

