Software-Defined Networking Architecture Framework for Multi-Tenant Enterprise Cloud Environments

by

Aryan TaheriMonfared

A dissertation submitted in partial satisfaction of
the requirements for the degree
PHILOSOPHIAE DOCTOR (PhD)

University of
Stavanger

Faculty of Science and Technology
Department of Electrical Engineering and Computer Science
October 2015

University of Stavanger
N-4036 Stavanger
NORWAY
`www.uis.no`

ISBN 978-82-7644-621-0
ISSN 1890-1387

PhD Thesis UiS no. 269

Abstract

The Information and Communications Technology (ICT) infrastructure of a large-scale enterprise consists of a wide variety of computing, storage, and networking hardware and software. The demand for new services and business models is growing rapidly; however, traditional operating mechanisms and computing models can not cope with the trend.

Particularly, network infrastructure and services are complex and hard to manage. The reasons are various, including rudimentary interfaces and vertically integrated networking planes. The situation is more critical when network resources are off-premises. These resources have limited functionality, while offering less control.

The contribution of this thesis is twofold. First, several architectural improvements are proposed for network monitoring services. These proposals take advantage of the data-intensive computing model and SDN mechanisms to advance the state-of-the-art in monitoring backbone and data center networks. Second, various components of an SDN architecture framework are designed that enhance the efficacy, reliability, and manageability of a large-scale cloud infrastructure. The enhancements are particularly made to network virtualization techniques, which are the critical building blocks in the cloud service delivery. This thesis specifically tackles five challenges:

The capacity and traffic volume of backbone networks are increasing rapidly. However, monitoring solutions have not evolved at the same pace and can not cope. Thus, a data-intensive computing platform is designed that significantly improves the scalability and efficiency of network monitoring services. The platform stores and processes large data-sets on commodity hardware and handles the data in real-time. The prototype provides the response three order of magnitude faster than traditional tools.

The multi-tenancy aspect of the cloud model is unknown to most network monitoring solutions. The SDN controller's global view of the network and the cloud controller's knowledge of the resource distribution and allocation are the key for improving monitoring tools. The results show the effectiveness and robustness of this approach, even in heterogeneous infrastructure where all resources are not visible to both controllers.

Packet payload analysis of traffic flows in a data center network is costly and can cause a complete network meltdown. The approach, to solve this problem, has two phases. First, finding a combination of monitoring hosts and switches for the designated traffic flows, such that the network will not become congested, while minimizing the service overhead. Second, enforcing the solution on the network. The combinatorial optimization problem is solved with the help of an SDN controller. The controller provides the network information (e.g. utilization, topology, paths) and programs the network. The implementation proves the feasibility of payload analysis in large-scale data centers. For instance, 10% of traffic flows can be analyzed with less than 1% of data center hosts configured as commodity processing nodes.

Large-scale enterprises are reluctant to utilize cloud resources, because they lose control. The lack of control is more critical in virtual network resources. A novel and elegant network virtualization architecture is introduced to rectify the problem. The architecture proposes mechanisms for establishing virtual networks that are under the complete control of tenants. The architecture evaluation reports negligible overhead, while the offered functionality is considerably important.

In the cloud model, virtual machines can be customized, and the set of properties that define a virtual machine is called flavor. In contrast, virtual networks are established with the same specifications. A centralized SDN controller is used to customize virtual networks according to the tenant's requirements, and deliver virtual network flavors.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Chunming Rong, whose encouragement, guidance, and support from the initial to the final step, enabled me to develop the idea and write the thesis.

I am also grateful to Fernando M. V. Ramos for his constructive and encouraging comments. In addition to research guidance, he kindly provided me with the opportunity to stay at the University of Lisbon.

I thank Martin Gilje Jaatun, Hein Meling, and Antorweep Chakravorty for their insightful feedback, and Theodor Ivesdal for his technical assistance in setting up the infrastructure.

Throughout my studies I was fortunate enough to work with a number of talented people from UNINETT. I would like to thank Olav Kvittem and Otto Wittner for this opportunity and their support.

Special thanks to my parents for their unconditional love and sacrifices they make. I would also like to thank my friends for helping me get through the difficult times and the great time we shared.

Finally, I owe Noora a great deal of gratitude for her patience and support over the years it took me to conceive and execute this work. I simply could not have managed without her understanding and encouragement.

Aryan TaheriMonfared, October 2015

Contents

List of Papers

The following papers are included in this thesis:

- **Paper 1**

  Multi-tenant Network Monitoring Based on Software Defined Networking
  A. TaheriMonfared, C. Rong
  Secure Virtual Infrastructures (DOA-Trusted Cloud), 2013 International Conference on

- **Paper 2**

  Real-Time Handling of Network Monitoring Data Using a Data-Intensive Framework
  A. TaheriMonfared, T.W. Wlodarczyk, C. Rong
  Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on

- **Paper 3**

  Flexible Building Blocks for Software Defined Network Function Virtualization
  A. TaheriMonfared, C. Rong
  Heterogeneous Networking for Quality, Reliability, Security and Robustness (QShine), 2014 10th International Conference on

- **Paper 4**

Virtual Network Flavors: Differentiated Traffic Forwarding for Cloud Tenants
A. TaheriMonfared, C. Rong
Wired & Wireless Internet Communications (WWIC), 2015 13th International Conference on

- **Paper 5**

  On the Feasibility of Deep Packet Inspection for Multi-Tenant Data Center Networks
  A. TaheriMonfared, F. Ramos, C. Rong
  To be submitted

Chapter 1

Introduction

The Information and Communications Technology (ICT) infrastructure of a large-scale enterprise or a national research and development institute consist of a wide variety of computing, storage, and networking hardware and software. The architecture complexity and new service demands are growing significantly, however, traditional configuration and operation mechanisms are not scalable and flexible to cope with the trends. Moreover, the incurred Capital Expenditures (CAPEX) and Operational Expenditures (OPEX) costs make the infrastructure unsustainable in the long run. Therefore, new models and approaches are required to deliver services and maintain the infrastructure.

Cloud computing is emerging as a model that reduces the capital expenses and delivers new types of services. Customers (also known as tenants) provision and release resources on-demand from a shared pool of resources and access them via the network. The reduction in the capital expenses and the scalability of resources decreases the barriers for the introduction of a new application or business [4].

Traditional networks have fundamental challenges, and they demand for an improved architecture. Software Defined Networking (SDN) is a network architecture that breaks the vertical integration of networking planes such as data forwarding, control, and management planes. The architecture benefits from a logically centralized controller which is physically separated from the data plane; while, the controller is capable of managing and programming several data planes. The SDN mechanisms take advantage of abstractions, that extract simplicity from low level networking substrates, to facilitate an efficient control over the network [61].

This thesis studies various aspects of the SDN architecture and large-scale processing frameworks to improve the enterprise ICT infrastructure, including cloud platforms and backbone networks.

1.1 Enterprise Infrastructure

A typical large-scale enterprise infrastructure consists of a backbone network, and several campus area networks. The backbone interconnects campus networks and plays a critical role in the reliable operation of the infrastructure. Each campus may have multiple data centers, and within data centers, logical networks interconnect server farms. An abstract view of an enterprise architecture is depicted in Figure 1.1. The view is a simplified version of the Norwegian National Education and Research Network (NREN) topology[1].

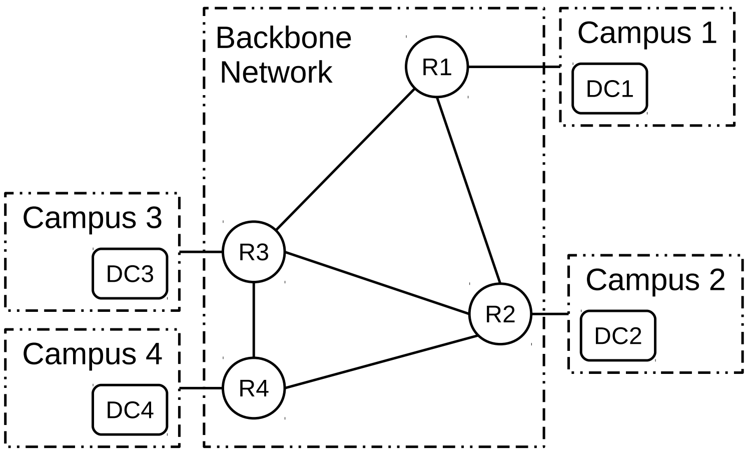


Figure 1.1: A High-Level View of an Enterprise Infrastructure (R: router, DC: Data Center).

A data center, or a subset of resources in it, can be tailored to deliver a specific type of services. For instance, the cloud Infrastructure as a Service (IaaS) model requires distinct components such as cloud platform controllers, network gateway nodes, and compute nodes (Figure 1.2). The components communicate with each other for different purposes, and each traffic workload is handled by a separate logical network. The open-source OpenStack [53] cloud platform is used as a reference implementation for the cloud service delivery model.

[1]https://drift.uninett.no/

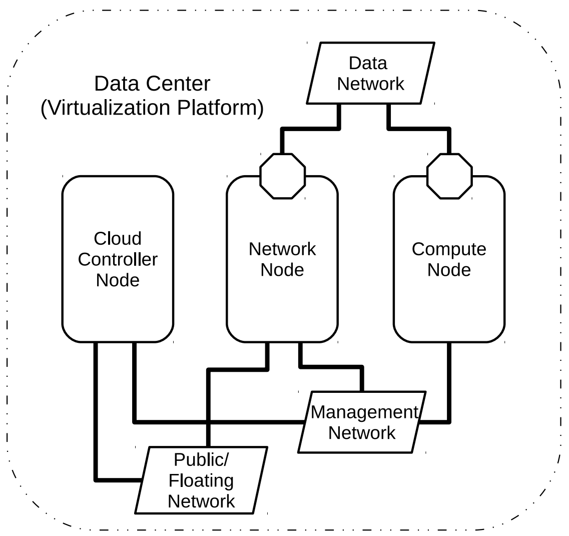


Figure 1.2: A High-level view of the logical components in a cloud infrastructure

One of the logical networks in the cloud infrastructure is the Data Network. It is the underlay for tenants virtual networks. A virtual network provides connectivity between provisioned resources of a tenant, moreover, it makes them accessible from outside.

The data center network plays a crucial role in the reliable operation of cloud services and has several challenging characteristics. Multi-tenancy, volatile configurations, and diverse requirements are parts of the cloud data center network characteristics, which are explained next.

The underlay serves several overlay networks and should isolate overlays' traffic. The multi-tenancy aspect requires a spectrum of isolation techniques, such as a simple subnet traffic isolation or a more sophisticated performance isolation.

The overlay networks are highly dynamic and make the configurations volatile. For instance, multiple changes in the configuration are required whenever tenants join and leave the platform, or provisioned resources migrate.

In addition, overlay networks impose a diverse set of requirements on the underlay (e.g. fine-grained QoS for each overlay).

1.2 Motivations

This section briefly explains some of the challenges in operating a large-scale ICT infrastructure, which are the motivations for this work.

1.2.1 Lack of scalable and fine-grained network monitoring

Monitoring solutions have not evolved at the same pace as the computing models and network architectures. Therefore, many aspects of the new models and architectures are not covered by these solutions. Moreover, the increased capacity of the infrastructure has put a significant pressure on traditional monitoring techniques, and rendered them inefficient. Some of these limitations and challenges are discussed in the following.

Unscalable Storage and Processing

The backbone network plays an important role in a reliable operation of a large-scale infrastructure, and network monitoring is a fundamental service for network management. Monitoring engines receive a high-volume of data streams from monitoring sources. The lack of scalable and efficient monitoring data processing techniques makes real-time querying infeasible. In addition, searching for new features in the archived monitoring data requires significant storage and processing capacity, as well as considerable time. This makes the data exploration intractable.

Coarse-Grained Monitoring

The cloud model introduces a complex provider-customer relationship. A tenant allocates resources and can provide services to other tenants. In addition, the cloud platform can be built in a heterogeneous infrastructure with varying components' features and vendors. Therefore, legacy network monitoring solutions have several shortcomings, in the context of cloud computing: (i) They capture the aggregation and are unable to distinguish between tenants' provisioned resources and resource utilization, due to the lack of knowledge about multi-tenancy and entities' relationships. (ii) They are incompatible with the new characteristics of the cloud model (e.g. on-demand resource provisioning and releasing). (iii) They are unreliable in a heterogeneous infrastructure.

Fine-Grained Monitoring

Deep Packet Inspection (DPI) and payload analysis are common services in the network [62]. Regularly, these services are deployed at the network borders (i.e. choke-points), to facilitate access to the egress and ingress traffic. The nature of traditional architectures and attack vectors allow choke-points to be effective.

However, in the presence of the multi-tenant data center infrastructure, several challenges are introduced in delivering these services, including (i) distribution of monitoring framework components (e.g. mirroring switch, collecting node, and processing node) for a complete visibility into the inter-tenant traffic, (ii) orchestration of components, (iii) programming of the underlay network for an efficient monitoring message transportation, (iv) incurred cost on the underlay network (e.g. link utilization, switching overhead, host processing overhead), and (v) tenant discovery and identification in the underlay network.

Monitoring messages are exchanged between monitoring components, and these messages may have the same frequency and volume of the monitored traffic (e.g. in case of traffic mirroring). The frequency and volume of the monitoring messages can degrade the performance or cause a complete meltdown. Thus, there should be reliable and dynamic network paths for delivering these messages, such that the network capacity is not overloaded.

In addition, a robust monitoring framework consists of multiple services and each service is replicated. The redundancy has several benefits, including (i) service high-availability, (ii) workload distribution, (iii) cost efficiency through commoditization, and (iv) better coverage. The redundancy demands for an orchestrator to coordinate the tasks among services.

However, traditional monitoring architectures can not control the transport network, nor do they have an orchestrator.

1.2.2 Lack of flexible and programmable virtual networks

Lack of virtual network Quality of Service (QoS)

A cloud tenant can specify the properties of a requested virtual machine (VM) based on its business requirement. For instance, the properties can determine the number of virtual CPUs, memory volume, storage capacity, virtual NIC speed, etc. The set of properties, that describes a VM, is commonly known as the VM flavor.

However, cloud platforms do not support a similar feature for virtual networks (i.e. virtual network flavor). In other words, a tenant can not specify the characteristics of its virtual network, such as link speed, maximum and minimum rate, end-to-end QoS, etc. The lack of a mechanism for enforcing QoS limits the cloud computing efficacy and scalability. The inflexibility is not satisfactory for the large-scale tenants and those with a special networking requirements.

Lack of control over virtual networks

Enterprises use cloud computing to benefit from the new business and service models. For instance, Amazon serves a significant number of customers with its cloud services, and large-scale service providers, such as Netflix and Dropbox, are among them [1].

The customers provision a large number of virtual machines, and have a complete control over them. The great level of control is possible, because of the resource type (i.e. virtual machine) and the way the access is provided (i.e. a privileged user in the operating system). Typically, Configuration Management (CM) utilities are employed to operate and maintain these resources. The CM automates the configuration process and tunes the resources to support the business, efficiently.

However, virtual networks are not as accessible as virtual machines. Cloud providers offer virtual network as a mean for interconnecting provisioned resources of a tenant. Although the tenant can control all aspects of a virtual machine operating system, there is no mechanism to provide the same level of control for virtual networks.

The lack of such a mechanism is a tremendous burden on the adoption of the new computing model. Enterprises, with a large number of provisioned resources across multiple cloud data centers, should be able to modify their network configurations and enhance their services (e.g. routing algorithm, networking access control lists).

1.3 Proposed Enterprise Infrastructure

Large-scale enterprises are lacking control and services in the cloud model. The deficiency is more critical in the network domain. In addition, the operation and maintenance of the traditional network is challenging [6, 41], whether physical (e.g. on-premises) or virtual (e.g. off-premises) network. Therefore, new mechanisms are required to reduce the costs and improve the control over the network.

Moreover, the volume and complexity of data, that should be processed in an enterprise, are growing. For instance, the network monitoring data is increasing and various protocols with different data structures are providing monitoring services. The traditional data processing frameworks do not cope with the trend. Thus, new paradigms are emerging that facilitate distributed processing of huge data-sets in real-time.

In this thesis, the SDN architecture and large-scale data processing frameworks are used to tackle some of the aforementioned challenges. A high-level view of an improved enterprise infrastructure is depicted in Figure 1.3.

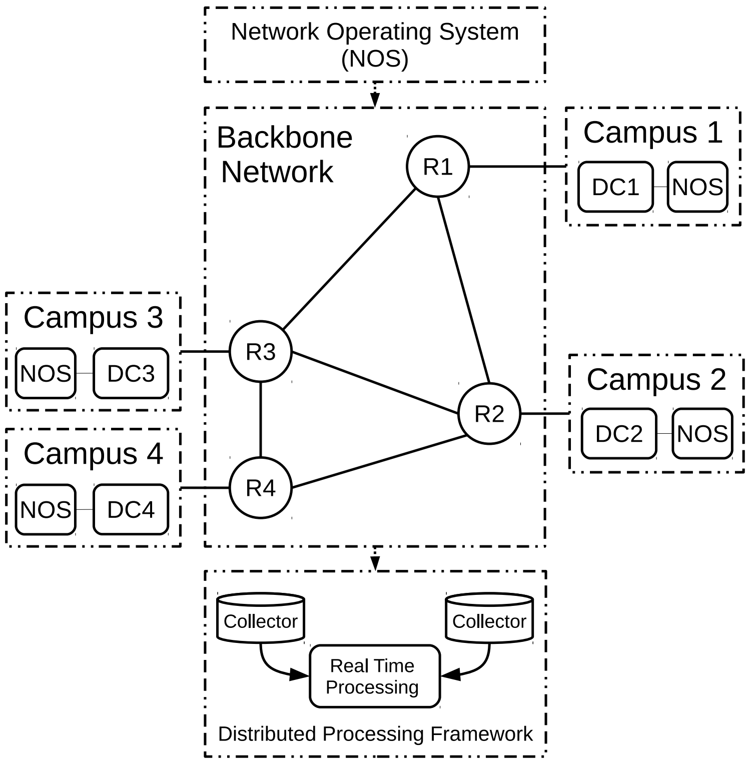


Figure 1.3: A High-Level View of an Improved Enterprise Infrastructure (R: router, DC: Data Center, NOS: Network Operating System/SDN Controller)

The recent advances in Software Defined Networking (SDN) [61, 46] simplify the network configuration and operation [38]. The SDN architecture can significantly improve the efficiency and flexibility of the cloud model.

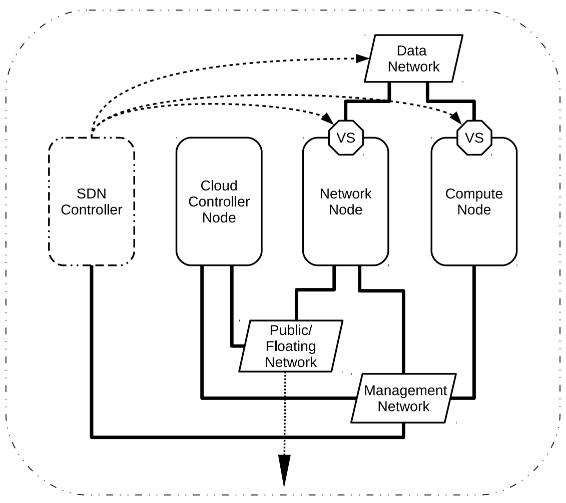


Figure 1.4: High-level view of the logical components in a cloud infrastructure with a SDN controller

Figure 1.4 depicts a cloud platform which utilizes an SDN controller[2] for managing the Data Network and virtualized network components. When the cloud controller makes changes to the platform (e.g. provisioning new resources for a tenant), the SDN controller is asked to update the network configuration and program the data plane across the network to serve the new request's objectives. In addition, the SDN controller can take a proactive approach and update the network by learning from the past trends.

Data center networks have multiple layers with redundant switches and links. A typical three-tiered data center network has the edge tier, which connects the servers to the network fabric; the aggregation tier, which interconnects the edge switches to the core switches; and the core layer, which interconnects aggregation switches and can connects the network to the WAN [19]. Data center network architecture is further explained in Section 2.3.2.

A three-tiered data center network is depicted in Figure 1.5. This represents the Data Network of a cloud infrastructure, where all network devices are controlled by an SDN controller. In addition, for a better integration with the cloud platform, the SDN controller manages the virtual switches in the compute nodes and maintains overlay virtual networks.

[2]The terms SDN controller and Network Operating System (NOS) are used interchangeably.

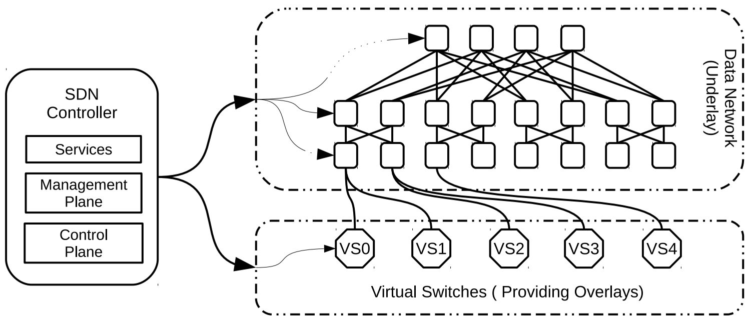


Figure 1.5: SDN controlled Data Network and virtual components (virtual switches) which are necessary for building a cloud data center.

1.4 Contributions

This thesis proposes mechanisms for an efficient operation, maintenance, and monitoring of a large-scale network infrastructure. Primarily, it focuses on the backbone network across campuses and data center networks in each campus. The data center, studied here, is tailored for cloud services.

1.4.1 Network Monitoring

This section summarizes the contributions that tackle the challenges of network monitoring services.

Scalable Storage and Processing

The advent of Big Data [65] processing techniques can be used to address the scalability issues of data storage and processing in network monitoring services. Paper 2 proposes a novel approach for scalable storage and real-time processing of network monitoring data. It takes advantage of data-intensive frameworks for storing data in a flexible and protocol-agnostic schema. The designed schema and data access model facilitate low latency responses to exploratory and ad-hoc queries, while preserving the accuracy of long-term planned queries. The solution provides a querying mechanism which is three order of magnitude faster than the traditional solutions.

Fine-Grained Extraction and Collection

Paper 1 discusses two types of data extraction methods, which fulfill the monitoring requirements of a heterogeneous infrastructure. The first type benefits from the cloud platform APIs to learn about multi-tenancy, tenants' relationship, and their utilized resources. Then, it processes the network monitoring data, which are collected by legacy tools, to build per-tenant traffic statistics. The second type takes advantage of the unified network view of an SDN controller to derive the resource mapping information, which are used for further processing of legacy tools' monitoring data.

Distributed Monitoring and Framework Orchestration

As explained in Section 1.2.1, the monitoring service should be distributed and an orchestrator should coordinate the task among service nodes. In addition, the orchestrator is responsible for instructing the network for delivering monitoring messages between responsible nodes.

Paper 5 introduces a novel design for distribution and transportation of the monitoring messages and orchestration of framework service nodes, such that the cost of monitoring is low and reliability is high. The design takes advantage of the unified view of an SDN controller over the network and the knowledge of a cloud controller about resource distribution. The orchestrator learn about the network topology, monitoring components, and target traffic from these two controllers and find a configuration with the right balance between the cost and accuracy. Then, it asks the SDN controller to configure the monitoring switches and program the path for messages.

1.4.2 Flexible and Programmable Virtual Networks

The flexibility and control are the missing features of virtual networks for enterprise and large-scale tenants. The following contributions improve these aspects of the network virtualization significantly.

Virtual Network Flavors

The lack of a mechanism, for customizing virtual networks and delivering flavors (e.g. QoS), limits the efficacy and scalability of cloud providers (Section 1.2.2). Paper 4 proposes an approach for defining flavors for virtual networks and a mechanism for differentiated forwarding of overlay traffic in the underlay network. A flavor specifies the network properties such as the coarse-grained network class, end-to-end priority, and traffic rate bounds.

The SDN controller receives the specifications of a virtual network and configure all paths for that virtual network such that they satisfy the requested QoS. Moreover, the design is not limited to an SDN only network where all devices support SDN protocols. As long as edge switches (physical or virtual) are SDN capable, overlays can be established such that the underlay understands the different classes and honor them.

Programmable Virtual Networks

The limited functionality of today's virtual networks are perceived as a burden on the cloud adoption by the large-scale enterprises (Section 1.2.2). Paper 3 presents new building blocks for network virtualization and proposes a mechanism for establishing virtual networks that are controlled and programmed by tenants. Tenant-controlled virtual network facilitates the decoupling of network functions from the hardware and simplifies the implementation in software. In addition, the off-premises and on-premises resources of a tenant can be integrated seamlessly, which foster the cloud model adoption.

1.5 Methodology

Several methodologies have been used to tackle the research questions within each topic. These methodologies facilitate domain exploration, problem description, solution realization, system modeling, and evaluation. The following sections explain each methodology and employed methods to perform the research.

1.5.1 Implementation

All proposed ideas throughout this thesis are implemented, evaluated, and compared with available baselines. Cloud solutions are integrated with OpenStack [53], and SDN modules are developed as a set of projects in the OpenDaylight SDN controller [43].

The implementations have a modular design with reusable components, which are made open-source.

1.5.2 Evaluation

The experimental methodology plays an important role in exploring the discipline, finding research questions, and evaluating the proposed solutions.

The evaluations are performed in several research phases, including exploration and evaluation phases. The exploratory experiments assist in understanding the system characteristics and building a baseline for the solution assessment. The other group of experiments evaluates the proposed solutions and studies their side-effects.

Record Keeping

The experiments should be conducted and reported such that they can be reproduced by the community. Therefore, all the steps and procedures are logged, which are required for reproducing an experiment. Apart from the exploratory experiment, which has an ad-hoc nature, the rest of experiments are automated. The automation of an experiment has several benefits, including a reduction in the time and complexity of rerunning the experiment by the authors, and making the process feasible for the community.

Although the experiments are automated and easy to reproduce, some researchers may be more interested in the raw data and do not want to spend resources on the re-execution. Therefore, the inputs and outputs of each execution are stored in a clear structure that facilitate offline processing by the community.

In addition, the data from a production system (e.g. backbone network traces) can contain sensitive information. Thus, proper measures, such as anonymization, are taken to preserve the privacy of users. For instance, Paper 2 uses a one-to-one and prefix-preserving technique [18] to anonymize IP header records before storing them in the processing framework.

Testbeds

The initial versions of an implementation is evaluated for the functional correctness in a virtualized testbed. When the functional aspects of the implementation are verified, the solution is deployed on a physical testbed and evaluated. The second evaluation studies the performance characteristics of the solution at scale, as well as its impacts on uncontrolled variables of the ecosystem.

	Testbed 1	Testbed 2	Testbed 3
Purpose	Data Processing	Cloud Platform	Cloud Platform
Nodes	20	8	5
CPU	AMD Opteron 4180	AMD Opteron 4180	Intel Core i5-4250U
Cores	6	6	4
Memory	32 GB	32 GB	16 GB
NIC	2x Gigabit Ethernet	2x Gigabit Ethernet	2x Gigabit Ethernet
Connectivity	HP ProCurve 5406zl	HP ProCurve 5406zl	Pica8 P-3290

Table 1.1: Testbeds' purposes and specifications.

Monitoring

Studying the side-effects of a particular solution is as important as measuring the solution's achievement. For instance, the throughput measurement for a new network virtualization method should happen in parallel with the CPU utilization measurement. Therefore, a variety of metrics in testbeds are continuously monitored, such as CPU usage, network utilization, memory usage, error rates, temperature, etc.

1.6 Organization

The remaining parts of the thesis is organized as follows. Chapter 2 discusses some background to the large-scale ICT infrastructure, including computing models, enterprise network infrastructure, Software-Defined Networking, network virtualization, and network monitoring. Finally, the papers, which are the basis of this thesis, are included chronologically.

Chapter 2

Background

This thesis tackles operational challenges of enterprise networking infrastructures. Therefore, it is crucial to understand the various aspects of such an environment, including computing models, network infrastructure, new architectures, and network services.

First, the cloud and data-intensive computing models are discussed. The cloud computing model is widely used in the enterprise infrastructure and the data-intensive computing model facilitates large-scale, distributed storage and processing. Second, the data center and backbone networks are reviewed and their challenging characteristics are explained. Third, to deal with these challenges an emerging network architecture (i.e. SDN) is explained, that improves the network management and configuration. Fourth, network services are presented that take advantage of the new network architecture and computing models to improve the efficiency and efficacy of the enterprise network infrastructure.

(i) *Computing models*: cloud computing (§ 2.1) and data-intensive computing (§ 2.2).

(ii) *Network infrastructure*: backbone network (§ 2.3.1) and data center network (§ 2.3.2).

(iii) *Network architecture*: software-defined networking (§ 2.4).

(iv) *Network services*: network virtualization (§ 2.5) and network monitoring (§ 2.6).

2.1 Cloud Computing

Cloud computing is an emerging computing model, which has gained considerable traction. The model provides a shared pool of elastic resources, that can be provisioned and released on-demand [48]. Customers of a cloud platform (also known as tenants) utilize resources from this pool. Typically, the cloud platform is deployed in large-scale data centers, and benefits from economies of scale.

2.1.1 Service Models

Cloud providers deliver different resource types according to their service models. The three main service models are Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) [48]. The IaaS model offers basic computing resources, storage services, and network connectivity. The resources can be virtual or physical, with different levels of isolation between tenants (e.g. different resource pools). The PaaS provides the hosting framework for deploying tenant-specific applications, that need to be compatible with the provider's platform. The SaaS offers software services that are hosted and managed by the provider, with the tenants having limited control over the software configurations.

2.1.2 Deployment Models

Cloud services are provided and utilized by various organizations. The organizations' types and relations determine the deployment model. The private, public, community, and hybrid models are the most common deployments [48]. The private cloud serves a single organization and it can be operated by the organization or a third-party. The public cloud serves the general public and can be provided by a variety of organizations. The community cloud deliver services to a group of organizations with similar objectives. When two or more deployment models are composed for delivering cloud services, a hybrid cloud is formed.

2.1.3 Characteristics of an IaaS Cloud

A typical IaaS data center has several types of service nodes, including platform controller, network node, compute node, storage node (Figure 2.1). The platform controller manages the cloud infrastructure and provides a communication interface for the end users. The network node facilitates virtual network services for tenants. The compute node is either provisioned by a tenant on-demand, or hosts virtual machines for multiple tenants. The storage node delivers storage services such as block and object storage. Scalability and reliability are important aspects of a cloud platform. Thus, each service can scale horizontally by adding more service nodes.

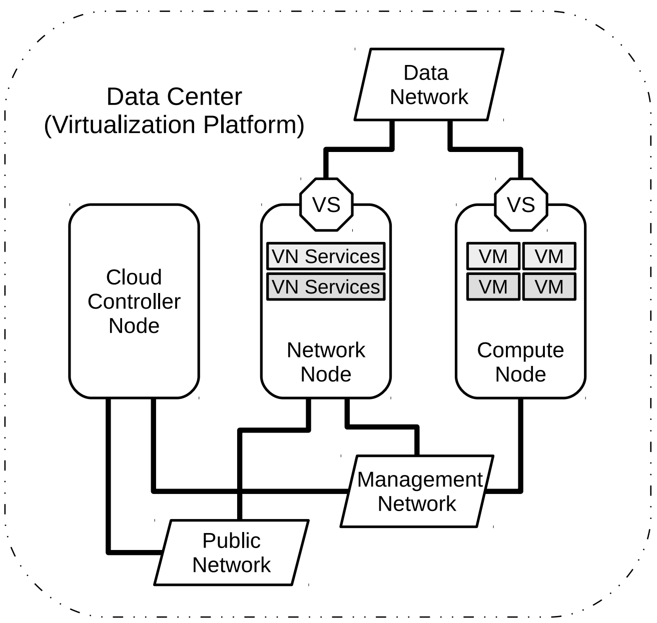


Figure 2.1: An IaaS Cloud Platform

In addition, multiple logical or physical networks fulfill diverse requirements. The management network handles the control traffic, the data network provides the underlay substrate for tenants' virtual networks, the public network makes the public interfaces of the cloud platform reachable by end-users.

Virtualization is the fundamental requirement of cloud service delivery. Server (i.e., computing resources) virtualization is a prominent technique for emulating physical hardware [5]. This technique allows coexistence of multiple virtual machines on top of a single physical machine and is the key to resource sharing in data centers.

Storage and network virtualization complement server virtualization, but they are still in an early stage when compared with server. They facilitate a full-fledged cloud platform with the aim to provide tenants with the same functionality as their own on-premises resources. Network virtualization is discussed in Section 2.5.

Moreover, existing data center solutions are not completely applicable to the cloud infrastructure. The issue resides in the new properties of the cloud model such as mandatory automation for scalability (1 person: 1000 servers), the size of cloud data centers (reaching 100,000 servers), and the horizontal scaling (scale-out) for increasing the service capacity [26].

2.2 Data-Intensive Computing

Data-intensive computing is introduced to address the fundamental requirements and mechanisms for a timely processing of large data-sets (i.e. Big Data) [25, 63].

The characteristics of Big Data make the existing processing tools (e.g. Relational Database Management Systems (RDBMS)) not able to cope. For example, traditional tools are incapable of handling the data in a reasonable time [65, 36]. Three characteristics define Big Data: volume, variety, and velocity [65]. Volume refers to the size of data. The data-sets are large enough to make the traditional processing mechanisms inefficient and unpractical. Variety captures the data structure heterogeneity of the data-sets, which is a typical property of them. Finally, velocity refers to the high frequency of data generation and the demand for fast processing.

Various frameworks have been proposed for data-intensive processing. The common properties of these frameworks are the distributed processing and horizontal scalability (i.e. scale out) using commodity hardware. The first system of such kind was MapReduce [15], a programming model introduced by Google. These frameworks are available in different forms such as open-source projects, proprietary software, and commercial services. For instance, Apache Hadoop [2], Spark [3], Google File System (GFS) [23], BigTable [9], and Microsoft Dryad [35] address different aspects of data-intensive processing.

2.2.1 Network Monitoring Data

This thesis pays particular attention to monitoring data. The data from network monitoring probes have the characteristics of Big Data. They have a high volume, are generated frequently with a variety of probes and protocols, and are processed for both exploratory (i.e. real-time) and planning (i.e. batch) purposes [13].

The operators demand fast responses from exploratory queries and accurate outputs from batch queries. However, the data characteristics make the existing tools unpractical, and require new approaches. Data-intensive processing frameworks are promising alternatives to tackle the scalability challenges of monitoring data storage and processing.

Paper 2 proposes a novel approach for storage and processing of network monitoring data, using data-intensive processing frameworks.

2.3 Network Infrastructure

An enterprise infrastructure may have various types of network. This section briefly explains two types: backbone and data center networks.

2.3.1 Backbone Networks

A backbone network interconnects various parts of network with different spatial scopes such as local area, campus, and data center networks. The backbone networks are expected to have high capacity and availability, insignificant congestion, and low delay [54, 22, 64].

Although these networks are well-designed and engineered, frequent failures are inevitable [32]. Therefore, it is crucial to have a comprehensive understanding of the network characteristics and detailed operational data [45].

2.3.2 Data Center Networks

A Data Center (DC) is a large set of computer clusters that is operated by an organization [7]. Data centers have various sizes and purposes, which are determined by the organizations' requirements. Large universities, private enterprises, and cloud service providers are common organizations with data centers.

A typical data center network has a hierarchical structure (Figure 2.2). Devices at the higher levels have more capabilities and capacities to provide interconnections to the lower levels, and consequently are more expensive. The available bandwidth between two hosts depends on the topology bisection capacity and hosts' locations. The bisection capacity of a tiered topology is the aggregate capacity of the links at the core level. A topology with full bisection bandwidth provides uniform bandwidth between any two hosts, regardless of their locations [7].

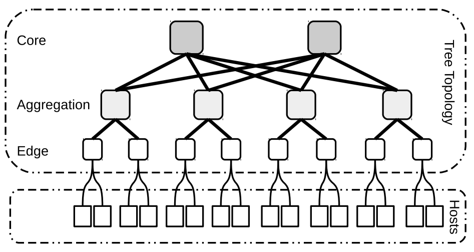


Figure 2.2: Typical data center tree topology. Inter-switch links are 10 GigE and switch-host links are GigE.

However, a traditional tree topology can not provide full bisection with a reasonable cost. Achieving full bisection bandwidth for 1 Gbps Ethernet is cost prohibitive, and for 10 Gbps Ethernet with current switching capacity is infeasible [19].

Over-subscription in the network is introduced to decrease the data center cost. It is the ratio of the highest aggregate bandwidth between the hosts at the edge to the total bisection capacity [19]. However, the over-subscription limits the network capacity between some hosts and leads to non-uniform bandwidth between them. Non-uniform bandwidth makes service performance a function of hosts' locations, degrading service performance and reducing data center agility [26].

Therefore, there is an increasing effort to design network architectures which have full bisection bandwidth and low cost [19, 27]. These architectures are based on a Clos topology [12], and benefits from a large number of commodity switches for building a high capacity network.

For instance, the data center network architecture proposed in [19] takes advantage of k-port commodity switches to build a Fat-Tree [42] topology. It uses $(5 * k^2)/4$ switches to provide full bisection capacity for $(k^3)/4$ hosts at the edge. The architecture is cost efficient for a data center with more than 10,000 hosts and it does not over-subscribe. Table 2.1 lists the network size and the number of supported hosts for a different number of switch ports.

A Fat-Tree topology with 4-port switches is depicted in Figure 2.3. It has 4 pods, each with 4 switches, and 4 core switches that interconnect the pods. A pod forms a logical container for the edge and aggregation switches, which are interconnected. This topology supports 16 hosts at the edge layer. Regardless of the number of ports in the switch (k), a path length (i.e. the number of links) between two hosts depends on the relative location of the hosts, and it is 2 for hosts on the same edge, 4 for

K	Core	Aggr./P	Edge/P	Aggr.	Edge	Pod Sw	Hosts	Nodes
4	4	2	2	8	8	16	16	36
8	16	4	4	32	32	64	128	208
16	64	8	8	128	128	256	1024	1344
32	256	16	16	512	512	1024	8192	9472
48	576	24	24	1152	1152	2304	27648	30528
64	1024	32	32	2048	2048	4096	65536	70656

Table 2.1: The impact of k on the topology size. The columns are the number of ports on the switch, the number of core switches, aggregation switches per pod, edge switches per pod, aggregation layer size, edge layer size, total number of switches in all pods, supported hosts, and the total nodes in the architecture.

same pod but different edges, and 6 for different pods.

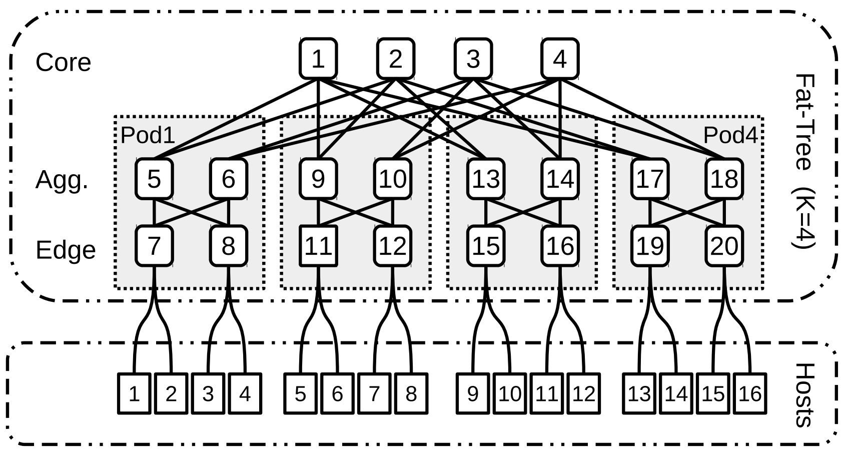
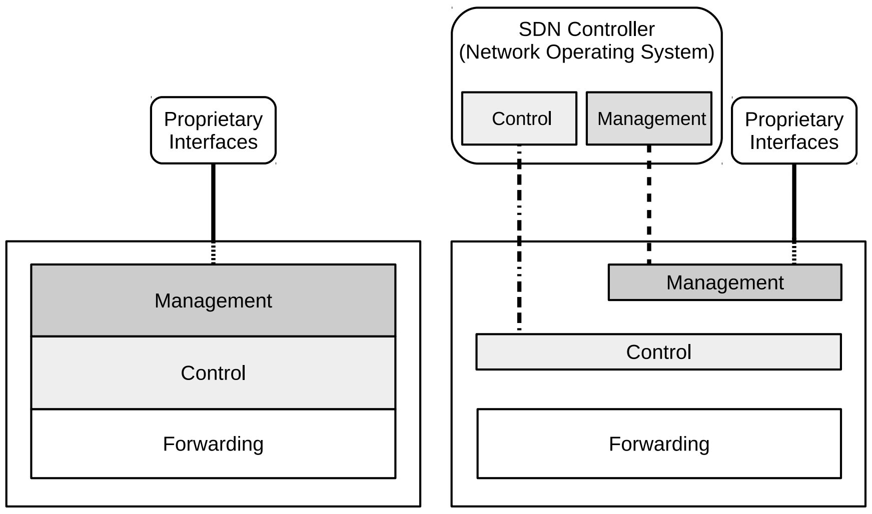


Figure 2.3: Fat-Tree with 4-port switches and hosts at the edge layer. All links are GigE.

Data center traffic has unique characteristics. A recent empirical study on data center network traffic [7] reveals that (i) most flows within a data center are small; (ii) the packet arrival process at the edge exhibits an ON/OFF nature; (iii) most traffic within a cloud data center stays in the rack; (iv) link utilization at the edge and aggregation are low and steady, but the loss rate is high; (v) link utilization at the core is high and fluctuating, but the loss rate is low. These empirical observations are used throughout this work to assist the design of the proposed solutions.



(a) A traditional switch **(b)** A switch with separated planes

Figure 2.4: An abstract model of a traditional switch and a switch with separated planes.

2.4 Software-Defined Networking

IP networks are complex and hard to manage [6]. Their devices have several issues, including vendor-specific and primitive configuration interfaces, tight coupling of networking planes, direct and indirect costs.

First, network devices are configured through vendor-specific interfaces, whether being Command Line Interfaces (CLIs), Application Programming Interfaces (APIs), or high-level languages for describing policies. The lack of a standard and unified protocol, that is also reliable and flexible, hinders network configuration and programming.

Second, network planes are vertically integrated. Management, control, and data forwarding planes are tightly coupled inside the network device. The exposed functionality to the user is limited and available through rudimentary vendor-specific interfaces. Therefore, introducing a new control logic or management approach is a daunting task. Figure 2.4a depicts a traditional switch, where all planes are vertically integrated.

Third, the network cost (i.e. direct) can be considerably high in a large-scale data center. The high cost is caused by the natural port speed and switching capacity limitations [19]. Although networking does not corresponds to the largest cost in a data center, it has a key role in reducing the overall cost (i.e. indirect) and increasing efficiency [26].

These challenges hinder innovation and make evolution a daunting task in networking infrastructure. For instance, a new routing protocol requires 5 to 10, after its design, to become mature for deployment [41]. Similarly, a clean-slate change [21] in the Internet architecture is considered practically infeasible [59, 24].

Therefore, new approaches are required for building networks and interacting with them.

2.4.1 Definition

Software Defined Networking (SDN) [46, 61] is an emerging network architecture that facilitates efficiency, scalability, and security in the network. The SDN architecture is commonly defined by four principles [41]:

(i) The separation of the control and data planes.

(ii) Flow-based forwarding in the data plane.

(iii) Logically centralized SDN controller (also know as Network Operating System (NOS)) that instructs multiple forwarding planes.

(iv) Network programmability via the SDN controller.

In the SDN architecture the control plane is physically decoupled from the data forwarding plane. The data plane's forwarding behavior is determined by a set of flow tables, and the forwarding decisions are flow-based. A logically centralized SDN controller implements the control plane logic and remotely controls several data planes. The SDN controller builds a global view of the network, and offers a control logic that programs an abstract model of this view. Figure 2.4b shows a switch with separated data and control planes.

SDN provides mechanisms for a centralized network operation and configuration, that facilitate the development of networking solutions. These solutions can be categorized into two groups: (i) legacy network functions that used to be vertically integrated to the devices (ii) new network services that were not feasible before.

Although the SDN architecture achieves more control, provides better guarantees, and extracts simplicity [38], the complexity of network devices are not removed. The SDN abstractions *extract simplicity* and shield the complexity of low-level components, however, their realizations are not necessarily simple [60].

2.4.2 SDN Abstractions

SDN proposes three abstractions in the control plane: ”forwarding model abstraction”, ”distributed network state abstraction”, and ”specification abstraction”. These abstractions separate concerns, when solving network challenges, and are fundamental parts of the SDN architecture [60, 41].

The forwarding abstraction provides a flexible and vendor-neutral mechanism for instructing the forwarding device about the control logic decisions (e.g. forward flows from port a to port b). These instructions are not dependent on a particular hardware or software implementations.

The distributed network state abstraction facilitates a global view of the network. The SDN controller builds the global view using the messages received from the forwarding devices and controls the devices using this view. This abstraction, along with the achieved global view, masks the changes in the network state. Therefore, the distributed systems problem of the network is transformed into a logically centralized graph problem.

The specification abstraction provides mechanisms for expressing a high-level task (e.g. access control). The implementation of a specific task is not the responsibility of these mechanisms. These mechanisms build an abstract view of the network. This view captures just enough details, that are required for expressing a high-level task.

2.4.3 SDN Protocols

The SDN realization requires various protocols to vertically separate the network planes. Each protocol addresses different aspects of the architecture and several protocols exist with similar purposes.

OpenFlow Protocol

OpenFlow [47] is the fundamental SDN protocol, realizing the forwarding abstraction. It is an interface for the communication between the SDN controller and network forwarding layer [52]. OpenFlow rules in a switch flow table are analogous to the x86 instruction set. In the same way as the x86 instruction set programs a computer, the OpenFlow rules programs a network forwarding plane.

The OpenFlow Switch specifications [51] describe the requirement of an OpenFlow Logical Switch as well as the OpenFlow protocol. The OpenFlow protocol describes the messages that are exchanged between the controller and the forwarding plane of an OpenFlow Logical Switch.

OpenFlow Config Protocol

OpenFlow Config (OF-CONFIG) [50] is a configuration and management protocol for OpenFlow Capable Switches. While OpenFlow instructs the switch about forwarding actions, OF-CONFIG enables remote configuration of various aspects of the switch (e.g. IP addresses, port status, etc.). OpenFlow Capable Switch is the operation context of OF-CONFIG, which can have one or more OpenFlow Logical Switches.

Open vSwitch Database Management (OVSDB) Protocol

Open vSwitch [57] is a multi-layer virtual switch that was designed from scratch for virtual environments. The OVSDB management protocol [56] was designed to address the management plane of the switch. The protocol is implemented in the switch and the controller, and provides an interface for switch management. OVSDB uses JSON [8] for encoding protocol messages.

Despite its name, OVSDB is not limited to virtual switches and there are other fabrics (Pica8 [58], HP [30]) that support the protocol.

OVSDB and OF-CONFIG are both management protocols and the latter can be implemented on top of the former. Figure 2.5 presents an OpenFlow Capable Switch which is managed by an SDN controller and has vertically separated planes.

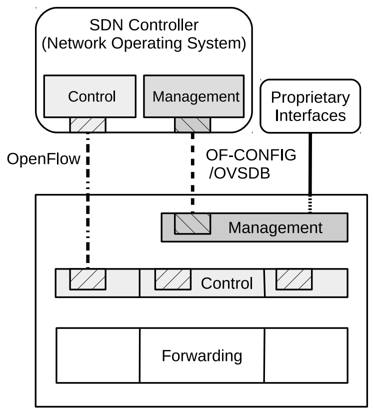


Figure 2.5: An abstract model of an OpenFlow Capable Switch.

Network Configuration (NETCONF) Protocol

NETCONF [16] is a protocol that defines mechanisms to manage the network. The mechanisms are capable of installing, modifying, and deleting the configuration of network devices. In NETCONF, the configuration data and the protocol messages are encoded in XML.

NETCONF is used as the transport protocol for OF-CONFIG and fulfills its operational requirements[50].

2.4.4 SDN Controllers

In an SDN architecture, the SDN controller is responsible for building a global view of the network and providing mechanisms for network management and control. A schematic view of an SDN controller is depicted in Figure 2.6.

The southbound interfaces are protocol drivers for the communication with network devices. Devices may support one or more protocols and protocols can address different aspects of a device (e.g. control plane, management plane). The service layer builds the global view and applies instructions to the network through southbound interfaces. The instructions can be from other internal entities or external applications via the northbound interfaces. The management and control logic contain basic centralized functions for management and control planes of the network. Network functions cover legacy network services, such as routing, that are now logically centralized. The northbound interfaces expose the SDN controller services to applications allowing the network to be programmed. The east-west interfaces facilitate inter-controller communication, that is crucial for the controller horizontal scalability and workload distribution.

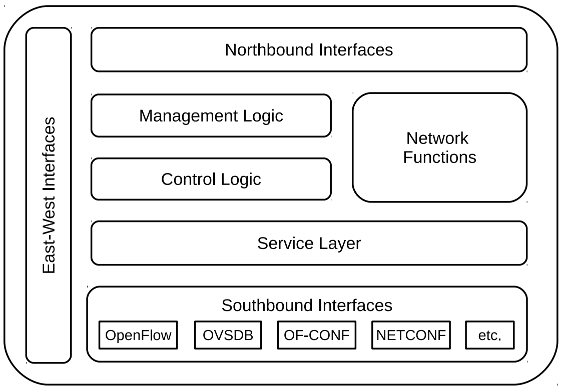


Figure 2.6: A schematic view of an SDN Controller.

The SDN architecture is endorsing a logically centralized model. However, it does not contend for a physically centralized implementation [40]. Indeed, production-level SDN controller requires horizontal distribution to scale out and a reliable network operation is dependent on the redundancy in the controller layer [47, 40]. The SDN architecture with a controller cluster is presented in Figure 2.7.

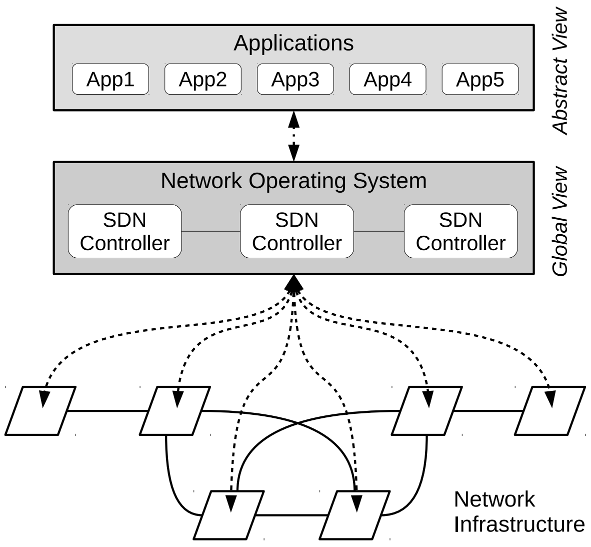


Figure 2.7: SDN Architecture.

2.4.5 SDN Adoption

The Open Networking Foundation (ONF), a nonprofit organization responsible for SDN standard development, is promoting SDN adoption [49] and has over 140 members[1].

SDN is emerging and large-scale enterprises have already adopted this architecture to improve their networks and offer new network services.

Google data centers are connected by B4, a private Wide Area Network (WAN). This network takes advantage of the SDN architecture and uses simple OpenFlow switches. The network utilization has reached the maximum, since a centralized traffic engineering mechanism was introduced [37].

SWAN [31] is the Microsoft Software-Driven WAN. The SWAN's centralized SDN controller configures network devices and determines the traffic rates of services. It reserves a small portion of the network capacity (i.e. link bandwidth and flow table capacity) to exchange control messages.

VMware's NVP is a network virtualization platform for multi-tenant data centers [39]. NVP's goal is to build virtual networks that are faithful reproductions of physical networks. The platform uses SDN principles to control software switches in compute nodes. The faithful virtual networks enable tenants to move their workloads from on-premises to off-premises resources without modifications.

2.5 Network Virtualization

Virtualization is an emerging technique for separating workloads from hardware in data centers [7]. It allows sharing of data center resources among tenants. The resources are of various types, such as compute, network, and storage.

Multi-tenancy at the network level is achieved by establishing virtual networks on top of the network infrastructure. In the cloud model, the resources provisioned for a tenant are connected to the tenant's virtual networks. The challenge is that tenants demand a seamless migration from on-premises to off-premises resources, and that requires flexible and controllable virtual networks [39]. Moreover, today's physical networks have difficulties in supporting workloads' heterogeneous service, topology, and addressing requirements.

Therefore, network virtualization [10] should offer a faithful reproduc-

[1]`https://www.opennetworking.org/our-members`

tion of physical networks and services that are completely isolated and independent of the physical network state and location [39].

2.5.1 Definition

Network virtualization facilitates virtual networks for various types of workloads, such that each one has an independent service model, topology, and addressing architecture, while sharing the same physical network (i.e. underlay) [39]. The recent trends in network virtualization endorse the use of a logically centralized controller that is responsible for the management and configuration of virtual networks.

2.5.2 Techniques

A virtual network is realized on top of a set of overlays and has a dedicated set of network services. The overlays isolate virtual networks traffic and facilitate the topology and addressing architecture independence, on top of a shared underlay.

The overlays can be achieved by VLAN tagging [33], IP block splitting, and tunneling [20, 44, 14]. The dedicated network services are platform dependent, for instance, network namespaces isolate network services on GNU/Linux platforms.

VLAN tagging and IP block splitting are the most prominent isolation techniques in production platforms, because of the low overhead and high performance. However, they have flexibility and scalability issues.

Traditional tunneling techniques have a higher overhead and lower performance compared to VLAN tagging and IP block splitting. Therefore, they were not widely used in production environment, despite their flexibility and scalability. However, the advent of efficient tunneling protocols (e.g. Stateless Transport Tunneling (STT) [14]) has relaxed these barriers, significantly.

The following explains the VLAN tagging technique and its challenges, as the first step for establishing virtual networks.

VLAN Tagging

Figure 2.8 presents the configuration details for establishing two virtual networks using VLAN tagging technique. Each virtual switch allocates a VLAN ID (i.e. internal VID) for each tenant. The internal VID of a virtual network may vary across virtual switches. The internal VID is translated to the data network VID, for that particular tenant, when packets leave the virtual switch. The reverse translation happens for ingress packets. Therefore, each switch (or an agent in the compute node) knows the binding between the virtual network ID, internal VID, and external VID.

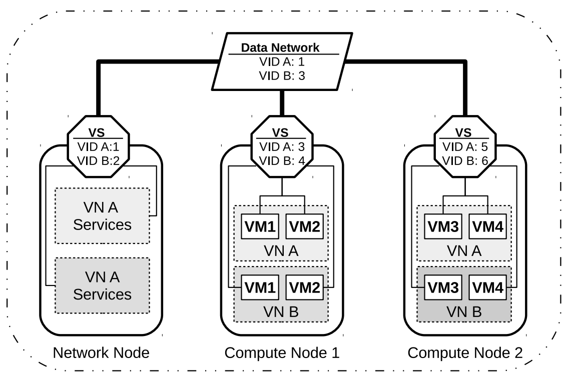


Figure 2.8: The complexity of isolation techniques. VS: Virtual Switch, VID: VLAN ID, VN: Virtual Network

Regardless of the chosen isolation technique, network virtualization is a complex task. Maintaining this level of complexity, in addition to the dynamics of the cloud environment, makes the current approaches intractable. Therefore, network virtualization is a niche for SDN mechanisms.

2.5.3 SDN mechanisms for NV

Tunneling is the most flexible and scalable technique for establishing overlays, therefore, this technique is used throughout the thesis. Tunnels of a virtual network form a mesh overlay as shown in Figure 2.9. They connect any two compute nodes, that are hosting provisioned resources of a particular tenant. The tunnel set is dynamic and it may change by provisioning and releasing resources. Moreover, a tunnel can be dedicated to a specific virtual network, or shared among multiple virtual networks which are present in a compute node. In OpenFlow 1.3, the former is called ”port-based” tunnel, and the latter is ”flow-based” [55].

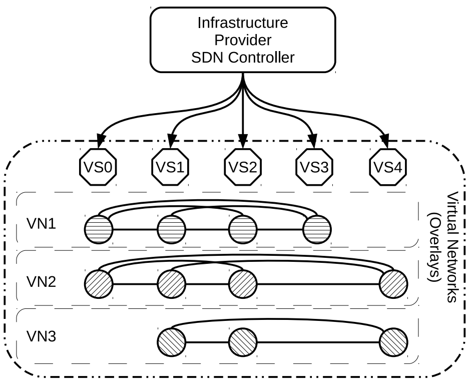

Infrastructure Provider SDN Controller
VS0
VS1
VS2
VS3
VS4
VN1
VN2
VN3
Virtual Networks (Overlays)

Figure 2.9: An SDN controller maintains virtual networks.

Current techniques connect all virtual resources to the same virtual switch on a compute node, regardless of their tenants (Figure 2.10). Although sharing a virtual switch can be perceived as an efficient approach, it limits the virtual network flexibility and constrains a faithful reproduction of physical networks (i.e. a network virtualization goal).

The virtual switches are controlled by the provider's SDN controller and tenants do not have any direct control on their virtual networks. A tenant can use provider's controller northbound APIs to send instructions and achieve a desired configuration. However, this interaction is limited by the controller interfaces offerings. In addition, the lack of a standard northbound API poses a significant vendor lock-in risk, and it can make the tenant application incompatible with other providers. Finally, the provider's controller failure disrupts the tenant applications.

Considering these challenges, a new mechanism for network virtualization is proposed in Paper 3, that fulfills the requirement of a faithful network reproduction. The idea is to create dedicated virtual switches for each tenant, such that they can represent on-premises network substrates. Moreover, the tenants can use any SDN controller to directly interact with provisioned network resources.

Although tenants are in control of the virtual networks, in this architecture, they do not have any control on the underlay network (e.g. data center network in Figure 2.3). In addition, all the virtual networks are established with the same specifications and lack flexibility. Paper 4 introduces a mechanism for enforcing QoS and customizing virtual networks.

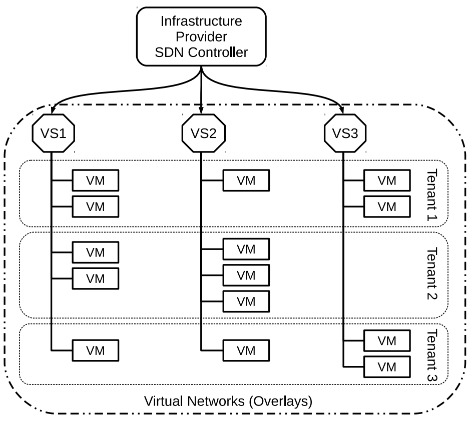

Infrastructure Provider SDN Controller
VS1
VS2
VS3
VM
Tenant 1
Tenant 2
Tenant 3
Virtual Networks (Overlays)

Figure 2.10: VMs placement with respect to the infrastructure components, in an SDN controlled platform.

2.5.4 Network Function Virtualization

Network Function Virtualization (NFV) [17] is an effort to decouple the network function logic from the underlying hardware, such that the network devices are consolidated onto regular servers, switches, and storage. This goal is achieved by implementing the network function in software, and running it on commodity servers and switches.

NFV decreases costs by consolidating network services onto commodity hardware, taking advantage of the economies of scale, and exploiting the hardware homogeneity. In addition, the software and hardware have independent life-cycle and they can evolve separately [17].

It should be noted that SDN and NFV do not compete but complement each other. NFV can be achieved without SDN mechanisms, but combining the two paradigms significantly improve future network architectures and foster innovation.

2.6 Network Monitoring

A robust monitoring framework plays a critical role in the proper operation and maintenance of the network [28, 22, 34]. Such a framework provides visibility into the network traffic and facilitates network research and exploration.

Network monitoring work-flow can be divided into several steps (Figure 2.11), including Monitoring Data Generation, Data Collection and

Storage, Data Processing, Transport and Orchestration.

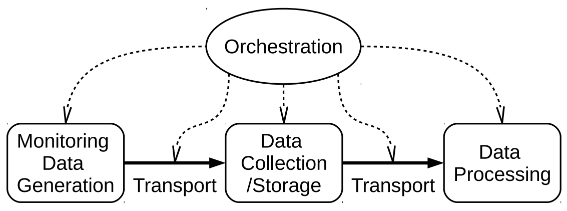


Figure 2.11: Abstract work-flow for network monitoring.

2.6.1 Monitoring Data Generation

Monitoring data have different sources and their origins depend on the network device capability, data type, the extraction method, and the processing purpose. Traffic statistics can be generated in each device, locally. Simple Network Management Protocol (SNMP) [29] agents send device statistics, in the form of messages, to the manager. NetFlow [11] uses flow exporter on the device to aggregate packets and build flow records. Payload analysis (e.g. Deep Packet Inspection) goes further and inspects the complete packet. Therefore, a copy of the target traffic should be sent to the collector. The copy can be made at any nodes on the traffic path.

Paper 1 discusses several methods for generating per-tenant monitoring statistics from the aggregated underlay network monitoring data.

2.6.2 Data Collection and Storage

This phase consists of collecting the exported data and storing them. Prior to the storage, there can be limited processing, such as aggregation and anonymization, to make the process efficient and policy compliant.

2.6.3 Data Processing

Processing network monitoring data has various objectives such as trend discovery, capacity planning, and real-time querying for data exploration and decision making. The monitoring data come in a great volume, and the expansion of the infrastructure increases the volume significantly.

However, today's techniques for network data processing are not scalable to the ever-grown datasets and do not fulfill timing requirements. Paper 2 proposes an efficient framework for the storage and processing of a large volume of monitoring data from backbone network.

2.6.4 Transport

An essential part of a monitoring framework is the transport mechanisms. Monitoring data traverse the network to reach the collectors and processors. The data volume, time constraints, sensitivity, and incurred network overhead (in terms of link utilization and switching fabric capacity) are important factors in designing an efficient transport mechanisms.

Paper 5 takes advantage of the SDN controller to program the network for an efficient delivery of monitoring messages.

2.6.5 Orchestration

As explained earlier, a robust monitoring framework consists of several service nodes, that cooperate to achieve a common goal. An orchestrator is responsible for (i) choosing a right combination of service nodes for a specific monitoring task; (ii) finding a suitable path between the nodes in conjunction with the routing mechanisms; (iii) enforcing the path.

The right set is chosen such that the resource utilization and task cost is minimized. The optimization problem requires knowledge about resource distribution and utilization, monitoring target (e.g. a specific traffic flow), and the impact of a monitoring task on the platform (e.g. resource usage).

In addition, there can be several paths that visit all the employed components. Therefore, the orchestrator is responsible for enumerating these paths, and selecting the most suitable one. When the path is chosen, the transport network should be programmed to forward monitoring messages accordingly.

Paper 5 uses the global view of the network and service node distribution to coordinate monitoring tasks and deliver an efficient monitoring service.

References

[1] *Amazon Web Services Case Studies*. 2015. URL: `http://aws.amazon.com/solutions/case-studies/`.

[2] *Apache Hadoop*. URL: `https://hadoop.apache.org/`.

[3] *Apache Spark*. URL: `https://spark.apache.org/`.

[4] Michael Armbrust et al. "A view of cloud computing." In: *Communications of the ACM* 53.4 (Apr. 2010), p. 50.

[5] Paul Barham et al. "Xen and the art of virtualization." In: *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*. SOSP'03. 2003, p. 164.

[6] Theophilus Benson, Aditya Akella, and David Maltz. "Unraveling the Complexity of Network Management." In: *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*. NSDI'09. 2009, pp. 335–348.

[7] Theophilus Benson, Aditya Akella, and David A. Maltz. "Network traffic characteristics of data centers in the wild." In: *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*. IMC'10. 2010, p. 267.

[8] T. Bray. *The JavaScript Object Notation (JSON) Data Interchange Format*. RFC 7159 (Proposed Standard). Internet Engineering Task Force, Mar. 2014. URL: `http://www.ietf.org/rfc/rfc7159.txt`.

[9] Fay Chang et al. "Bigtable: A Distributed Storage System for Structured Data." In: *ACM Transactions on Computer Systems* 26.2 (June 2008), pp. 1–26.

[10] N.M. Mosharaf Kabir Chowdhury and Raouf Boutaba. "A survey of network virtualization." In: *Computer Networks* 54.5 (Apr. 2010), pp. 862–876.

[11] B. Claise. *Cisco Systems NetFlow Services Export Version 9*. RFC 3954 (Informational). Internet Engineering Task Force, Oct. 2004. URL: `http://www.ietf.org/rfc/rfc3954.txt`.

[12] Charles Clos. "A Study of Non-Blocking Switching Networks." In: *Bell System Technical Journal* 32.2 (1953), pp. 406–424.

[13] Chuck Cranor et al. "Gigascope: A Stream Database for Network Applications." In: *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*. SIGMOD'03. 2003, pp. 647–651.

[14] B. Davie and J. Gross. *A Stateless Transport Tunneling Protocol for Network Virtualization*. Internet Draft (Informational). Network Working Group, Oct. 2014. URL: `https://tools.ietf.org/html/draft-davie-stt-06`.

[15] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters." In: *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*. OSDI'04. 2004, pp. 10–10.

[16] R. Enns et al. *Network Configuration Protocol (NETCONF)*. RFC 6241 (Proposed Standard). Internet Engineering Task Force, June 2011. URL: `http://www.ietf.org/rfc/rfc6241.txt`.

[17] ETSI. "Network Functions Virtualisation." In: Oct. 2012. URL: `http://portal.etsi.org/NFV/NFV_White_Paper.pdf`.

[18] Jinliang Fan et al. "Prefix-preserving IP address anonymization: measurement-based security evaluation and a new cryptography-based scheme." In: *Computer Networks* 46.2 (Oct. 2004), pp. 253–272.

[19] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. "A scalable, commodity data center network architecture." In: *ACM SIGCOMM Computer Communication Review* 38.4 (Oct. 2008), p. 63.

[20] D. Farinacci et al. *Generic Routing Encapsulation (GRE)*. Request for Comments 2784. IETF, Mar. 2000. URL: `http://www.ietf.org/rfc/rfc2784.txt`.

[21] Anja Feldmann. "Internet clean-slate design: what and why?" In: *ACM SIGCOMM Computer Communication Review* 37.3 (July 2007), p. 59.

[22] C. Fraleigh et al. "Packet-level traffic measurements from the sprint IP backbone." In: *IEEE Network* 17.6 (Nov. 2003), pp. 6–16.

[23] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. "The Google File System." In: *ACM SIGOPS Operating Systems Review* 37.5 (Dec. 2003), p. 29.

[24] Ali Ghodsi et al. “Intelligent design enables architectural evolution.” In: *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*. HotNets-X. 2011, pp. 1–6.

[25] Ian Gorton et al. “Data-Intensive Computing in the 21st Century.” In: *Computer* 41.4 (Apr. 2008), pp. 30–32.

[26] Albert Greenberg et al. “The cost of a cloud: research problems in data center networks.” In: *ACM SIGCOMM Computer Communication Review* 39.1 (Dec. 2008), p. 68.

[27] Albert Greenberg et al. “VL2: a scalable and flexible data center network.” In: *ACM SIGCOMM Computer Communication Review* 39.4 (Aug. 2009), p. 51.

[28] Andreas Hanemann et al. “PerfSONAR: A Service Oriented Architecture for Multi-domain Network Monitoring.” In: *Proceedings of the Third International Conference on Service-Oriented Computing*. Ed. by David Hutchison et al. Vol. 3826. ICSOC’05. 2005, pp. 241–254.

[29] D. Harrington, R. Presuhn, and B. Wijnen. *An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks*. RFC 3411 (INTERNET STANDARD). Updated by RFCs 5343, 5590. Internet Engineering Task Force, Dec. 2002. URL: `http://www.ietf.org/rfc/rfc3411.txt`.

[30] Hewlett-Packard Development. *HP-VMware Networking Solution Technical Brief*. 2014. URL: `http://h17007.www1.hp.com/docs/sdn/4AA5-4271ENW.pdf`.

[31] Chi-Yao Hong et al. “Achieving high utilization with software-driven WAN.” In: *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*. SIGCOMM’13. 2013, p. 15.

[32] Gianluca Iannaccone et al. “Analysis of link failures in an IP backbone.” In: *Proceedings of the 2Nd ACM SIGCOMM Workshop on Internet Measurment*. IMW’02. 2002, p. 237.

[33] “IEEE Standard for Local and Metropolitan Area Networks Virtual Bridged Local Area Networks.” In: *IEEE Std 802.1Q-2005 (Incorporates IEEE Std 802.1Q1998, IEEE Std 802.1u-2001, IEEE Std 802.1v-2001, and IEEE Std 802.1s-2002)* (2006), pp. 1–285.

[34] Marios Iliofotou et al. “Network monitoring using traffic dispersion graphs (tdgs).” In: *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*. IMC’07. 2007, p. 315.

[35] Michael Isard et al. “Dryad: distributed data-parallel programs from sequential building blocks.” In: *Proceedings of the 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*. EuroSys’07. 2007, p. 59.

[36] Adam Jacobs. “The pathologies of big data.” In: *Communications of the ACM* 52.8 (Aug. 2009), p. 36.

[37] Sushant Jain et al. “B4: experience with a globally-deployed software defined wan.” In: *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*. SIGCOMM’13. 2013, p. 3.

[38] Hyojoon Kim and Nick Feamster. “Improving network management with software defined networking.” In: *IEEE Communications Magazine* 51.2 (Feb. 2013), pp. 114–119.

[39] Teemu Koponen et al. “Network Virtualization in Multi-tenant Datacenters.” In: *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*. NSDI’14. 2014, pp. 203–216.

[40] Teemu Koponen et al. “Onix: A Distributed Control Platform for Large-scale Production Networks.” In: *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*. OSDI’10. 2010, pp. 1–6.

[41] Diego Kreutz et al. “Software-Defined Networking: A Comprehensive Survey.” In: *Proceedings of the IEEE* 103.1 (Jan. 2015), pp. 14–76.

[42] Charles E. Leiserson. “Fat-trees: Universal networks for hardware-efficient supercomputing.” In: *IEEE Transactions on Computers* C-34.10 (Oct. 1985), pp. 892–901.

[43] Linux Foundation. *OpenDaylight*. URL: `http://www.opendaylight.org/`.

[44] M. Mahalingam et al. *Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks*. RFC 7348 (Informational). Internet Engineering Task Force, Aug. 2014. URL: `http://www.ietf.org/rfc/rfc7348.txt`.

[45] A. Markopoulou et al. “Characterization of failures in an IP backbone.” In: *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*. Vol. 4. INFOCOM’04. 2004, pp. 2307–2317.

[46] Nick McKeown. *How SDN will shape networking*. 2011.

[47] Nick McKeown et al. “OpenFlow: enabling innovation in campus networks.” In: *ACM SIGCOMM Computer Communication Review* 38.2 (Mar. 2008), p. 69.

[48] Peter Mell and Timothy Grance. *The NIST Definition of Cloud Computing*. Tech. rep. SP 800-145. National Institute of Standards and Technology, Information Technology Laboratory, Sept. 2011.

[49] Open Networking Foundation. *ONF Overview*. 2015. URL: `https://www.opennetworking.org/about/onf-overview`.

[50] Open Networking Foundation. *OpenFlow Management and Configuration Protocol*. 2014. URL: `https://www.opennetworking.org/sdn-resources/technical-library`.

[51] Open Networking Foundation. *OpenFlow Switch Specification*. Apr. 2015. URL: `https://www.opennetworking.org/sdn-resources/technical-library`.

[52] Open Networking Foundation. *Software-Defined Networking: The New Norm for Networks*. Apr. 2012. URL: `https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf`.

[53] *OpenStack*. URL: `https://www.openstack.org`.

[54] K. Papagiannaki et al. “Measurement and analysis of single-hop delay on an IP backbone network.” In: *IEEE Journal on Selected Areas in Communications* 21.6 (Aug. 2003), pp. 908–921.

[55] Ben Pfaff. *Open vSwitch Manual*. URL: `http://benpfaff.org/~blp/ovs-fields.pdf`.

[56] Ben Pfaff and Bruce Davie. *The Open vSwitch Database Management Protocol*. Request for Comments 7047. IETF, Dec. 2013. URL: `http://www.ietf.org/rfc/rfc7047.txt`.

[57] Ben Pfaff et al. “The Design and Implementation of Open vSwitch.” In: *12th USENIX Symposium on Networked Systems Design and Implementation*. NSDI’15. May 2015, pp. 117–130. URL: `https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/pfaff`.

[58] *Pica8 Inc.* URL: `http://www.pica8.com/`.

[59] Barath Raghavan et al. “Software-defined internet architecture: decoupling architecture from infrastructure.” In: *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*. HotNets-XI. 2012, pp. 43–48.

[60] Scott Shenker. *Network Management and Software-Defined Networking (SDN)*. Dec. 2013. URL: `http://www-inst.eecs.berkeley.edu/~ee122/fa13/lectures/lec22.pdf`.

[61] Scott Shenker et al. “The future of networking, and the past of protocols.” In: *Open Networking Summit* (2011). URL: `http://www.opennetsummit.org/archives/apr12/site/talks/shenker-tue.pdf`.

[62] Justine Sherry et al. “Making middleboxes someone else’s problem: network processing as a cloud service.” In: *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. SIGCOMM’12. 2012, p. 13.

[63] Lizhe Wang et al. “G-Hadoop: MapReduce across distributed data centers for data-intensive computing.” In: *Future Generation Computer Systems* 29.3 (Mar. 2013), pp. 739–750.

[64] Rui Zhang-Shen and Nick McKeown. “Designing a Predictable Internet Backbone with Valiant Load-balancing.” In: *Proceedings of the 13th International Conference on Quality of Service*. IWQoS’05. 2005, pp. 178–192.

[65] Paul Zikopoulos and Chris Eaton. *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data*. 1st. McGraw-Hill Osborne Media, 2011.

Paper 1: Multi-tenant Network Monitoring Based on Software Defined Networking

Multi-tenant Network Monitoring Based on Software Defined Networking

A. TaheriMonfared[1], C. Rong[1]

[1] Department of Electrical Engineering and Computer Science, University of Stavanger
4036 Stavanger, Norway

Abstract:
In any cloud service model multiple stakeholders are involved with different roles in the service provider-customer relationship. One service provider can use other services at the same time. It is vital to distinguish between stakeholders activities such as their communication in the network. Thus, there should be a monitoring system, which knows about stakeholders, their characteristics, and provisioned resources by them at any point of time. In most cases, traditional monitoring solutions does not have the capability to understand the difference between involved parties, while they're orchestrated to achieve a desired result (e.g. delivering a service for the end user).
In this study an improved architecture of the networking service for a cloud platform is introduced. Software Defined Networking (SDN), Network Virtualization, and traditional methods are adapted for gathering evidence and auditing activities on a per-tenant basis. Four approaches are investigated for monitoring and identifying tenants' traffic in an infrastructure. They are discussed and implemented to fulfill the cloud's network monitoring requirements. The focus is on the effectiveness of software defined networking for monitoring tenants' virtual networks.
Keywords: Software Defined Networking; Network Virtualization; OpenFlow; Network Monitoring; Multi-Tenant

1 Introduction

In a virtualized environment, physical network resources can provide several virtual networks (VN). Traditional Internet Service Providers (ISPs) have two major roles, consisting of: physical infrastructure management and end-to-end service delivery. Network virtualization is realized by separating these roles and assigning them to different entities. In such an environment, an Infrastructure Provider (InP) is responsible for the physical infrastructure management, and a Service Provide (SP) is responsible for delivering the end-to-end network service [7]. Functionality decoupling leads to co-existence of multiple virtual networks which can be created by different SPs. These virtual networks are isolated and may span multiple InPs [8].

Network virtualization can be seen as a utility for studying new networking architecture or an important attribute of the new architecture [1]. The main idea behind network virtualization can be implemented using multiple technologies [7], such as: Virtual Local Area Network (VLAN) [15], Virtual Private Network (VPN) [14], active and programmable networks [6], overlay networks, and Software Defined Networking (SDN) [18]. SDN is a platform for network virtualization [12] that abstracts the hardware from the programming interfaces [19].

In the Infrastructure as a Service (IaaS) model, a customer has a set of virtual machine instances distributed over geographical regions. Instances' connectivity can be realized using Network Virtualization. A proper network monitoring solution must distinguish between customers' activities in the infrastructure's network. This is challenging due to the inherited properties of virtualization (e.g. utilizing shared pool of resources). Observation points should be placed in multiple layers with the functionality to monitor in different granularities. This study investigates applicability of existing monitoring techniques to the new computing model. Traditional techniques are improved using a network controller (e.g. an OpenFlow controller), which provides a unified view of networking substrates as well as the information about provisioned resources by each tenant.

1.1 Related Works

There has been a few efforts on the monitoring aspect of SDN, either using SDN for monitoring an infrastructure, or evaluating new characteristics of SDN enabled networks. Jose et al. [16] propose a low overhead measurement framework for commodity network switches that support OpenFlow. The solution is tuned to identify large traffic aggregates. Yu et al. [24] designed a push-based monitoring system using control messages sent to the OpenFlow controller by switches. It estimates links utilization in a passive way by analysing the stream of PacketIn and FlowRemoved messages. Ballard et al. [3] introduce OpenSAFE as a reliable and high performance approach of routing traffic for security monitoring purposes. In addition, a language (ALARMS [3]) is defined for management of network monitoring equipments. Braga et al. [5] presented a low overhead detection method for DDoS flooding attacks, which uses self organizing maps. The method periodically polls flow records from all OpenFlow enabled switches, and extract DDoS related features. Tootoonchian et al. [22] designed an OpenTM for traffic matrix (TM) estimation using capabilities in OpenFlow switches. TMs are crucial for anomaly detection in a network. Based on the routing information provided by the OpenFlow controller, OpenTM chooses a set of switches for querying. An efficient querying strategy lowers the overhead in network equipment.

The rest of the paper is structured as follows: Section 2 explains a common cloud platform architecture. Section 3 gives an overview of the network monitoring in a cloud platform with visibility into tenants' activities. Section 4 discusses challenges of designing such an awareness and monitoring system, as well as shortcomings of existing techniques. Then, Section 5 proposes four techniques to overcome challenges and shortcomings. Finally, Section 7 concludes the paper and compares proposed techniques.

2 Abstract Cloud Platform Architecture

In the following, an abstract architecture of a cloud environment is explained. The architecture is the basis for our testbed. Later, the architecture is improved by delivering the networking service using SDN.

Three types of nodes are introduced in the architecture (Figure 2.1), comprising: Platform Controller, Network Gateway, and Compute Node. The platform controller runs core services, such as: inter-module message passing, authentication and authorization, virtual machine image management, basic networking service, job scheduling, and database service. Although core services are centralized in a single node, each service can be replicated on multiple nodes. The network gateway provides core networking services, including: DHCP service, L3 connectivity to the external network. The compute node hosts virtual machine instances and handles their network connectivity through a dedicated virtual switch.

From the networking perspective, four types of logical network are available that serves diverse requirements: infrastructure management, VM's communications, API access, and VMs access to the external networks.

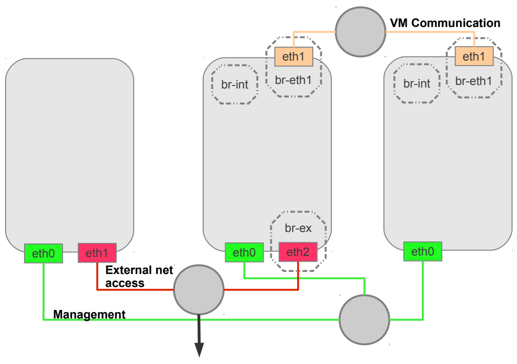


Figure 2.1: Abstract Architecture

Figure 2.2 gives more details about the network gateway and the compute node. The compute node has two and the network gateway has three virtual switches. Virtual switches are responsible for inter-VM connectivity. For the sake of simplicity one tenant (i.e. *Tenant A*) in a single compute node is depicted. Each tenant in the cloud platform has a dedicated virtual domain in the network gateway. The domain consists of virtual routers and networking services required for VMs connectivity and operation.

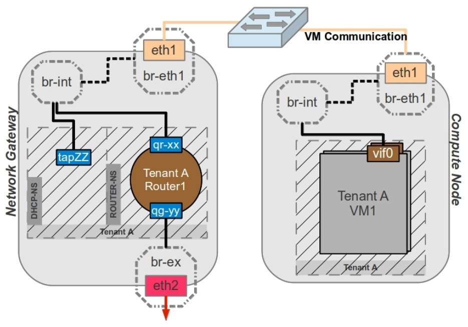


Figure 2.2: Architecture Networking Details

As depicted in Figure 2.3, *Tenant A* has several VMs distributed over multiple compute nodes. It has dedicated VLAN IDs (VIDs) on the physical switch (i.e. VM Switch 1), and virtual switches. VIDs on the physical and virtual switches may not be identical, since they're in different domains. The virtual switch in the compute node is responsible for translating VIDs between the physical and virtual domain. The mapping information is provided by the controller node, and is dependent on the tenant. All VMs, belonging to a particular tenant, are in the same broadcast domain, although they might be in multiple compute nodes.

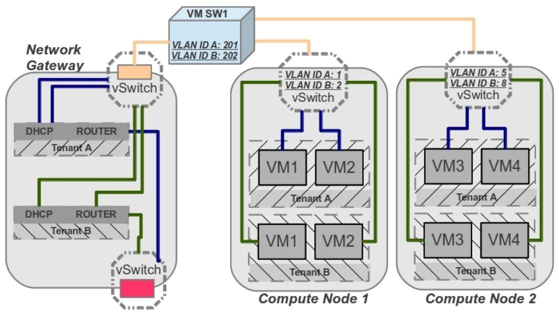


Figure 2.3: VLAN Configuration Details

In the testbed in Figure 2.1, the OpenStack Compute project (Nova[2]) provides the compute virtualization platform, and the networking project (Quantum[3]) is responsible for the networking services. Virtual switches are Open vSwitch[4] instances controlled by the Quantum server and they support the OpenFlow standard [20]. This feature facilitates building a unified view of our network and fulfilling the requirements of our monitoring solution (5.2). In addition, virtual switches are capable of exporting NetFlow[10], and sFlow[21] data, which are used for maintaining network awareness.

3 Tenant Based Network Monitoring Overview

In a distributed environment such as a cloud several stakeholders are involved. Stakeholders can be both service providers and consumers at the same time. Complexity of these interactions and the new characteristics of the networking architecture hinder the adaptation of traditional monitoring

[2]`http://www.openstack.org/software/openstack-compute/`
[3]`http://www.openstack.org/software/openstack-networking/`
[4]http://openvswitch.org/

approaches. The following section explains our approach for monitoring a networking environment that supports network virtualization.

Monitoring can be done in different layers. By separating the layers based on the provider type, there will be three layers: service provider layer, platform provider layer, and infrastructure provider layer. There is no clear cut between these types and a single provider can cover several responsibilities.

- **The Infrastructure provider** is responsible for providing underlying substrates, such as physical compute nodes, networking equipment, cooling facilities, etc. (e.g. a data center provider). These components might be heterogeneous and distributed over a variety of geographic locations, under multiple jurisdictions.

- **The Platform provider** delivers functionalities which utilizes underlying components. It builds a unified view of distributed substrates for the upper layer entities. As an example a platform provider is responsible for deployment, maintenance, and operation of a cloud platform (e.g. OpenStack[5]), or a network virtualization platform (e.g. an SDN environment using standards such as OpenFlow [20])

- **The Service provider** uses functionalities exposed by the platform provider to build a variety of services. The provider can deliver one or more services from the cloud service models (e.g. Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Software as a Service (SaaS) [2]).

However, in a cloud environment where several stakeholders are involved, fine-grained monitoring capability is crucial. It should be possible to monitor all resources used by a specific tenant (e.g. service customer). As an example, in a security incident the responsible Computer Emergency Response Team (CERT) should be able to monitor resources which are or have been provisioned by a tenant. Switches, routers, and interfaces (either physical or virtual) are considered to be resources, and a proper set of observation points should be identified for fine-grained monitoring.

This study concentrates on the Infrastructure as a Service model of Cloud. Therefore, network monitoring approaches for auditing low level resources (e.g. virtual machines, block storage) are investigated.

[5]`http://www.openstack.org`

4 Tenant Based Network Monitoring Challenges

New generations of cyber-attacks use sophisticated communication methods. The communication period is short and the traffic may not be significant. As an example Stuxnet uses five methods to propagate itself: peer-to-peer communication, infecting WinCC machine, network shares, MS10-061 Printer Spooler, and MS08-067 Windows Server Service [13]. None of these methods employ heavy scanning or long period communication. Thus, it is crucial to have a complete view of flows, and active services. Berthier et al. [4] proposes a solution for maintaining such a view, and we will investigate its application in a virtualized environment.

NetFlow services [10] provide information on IP flows from network elements, such as routers and switches. Flow data are sent from data networks to collectors. The data has detailed information about resource usage and can be helpful for obtaining network awareness. According to [10], a flow is considered to be unidirectional and bidirectional flows (Biflows) is a sequence of packets that pass through a network in both directions between two endpoints [23]. Thus, Biflows are more informative for security purposes. Although it is possible to determine Biflows in the collector using NetFlow data, it's more efficient to have it as an extension in the protocol [23]. The IP Flow Information Export (IPFIX) protocol [11] exports Biflows, though it is not widely used in substrates and is not supported by Open vSwitch. Thus, analytic tools are developed for identifying Biflows.

NetFlow is considered to be the most suitable protocol because of its accuracy. Virtual switches are configured to export their NetFlow data to a collector for further analysis. Nfcapd, nfdump[6], nfsen[7], and NFSight [4] are used for the initial collection and processing of data.

Monitoring the virtual switch in each node provides better visibility of the VMs communication. There are two virtual switches in each compute node, `br-eth1` and `br-int`. `br-eth1` is our observation point, since it is connected to the Top of the Rack (ToR) switch and handles all VMs' traffic inside the physical machine. On the network gateway, three virtual switches are configured, and two of them are monitored, `br-eth1`, and `br-ex` (external network access for VMs).

As depicted in Figure 2.4, the aggregated traffic on a virtual switch is visible. However, we can not differentiate tenants' traffic yet. Since

[6]`http://nfdump.sourceforge.net/`
[7]`http://nfsen.sourceforge.net/`

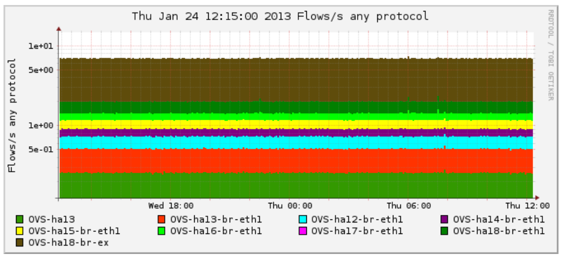


Figure 2.4: Network traffic observed by a monitoring tool without the knowledge about tenants' co-existence

tenants may have used overlapping IP addresses, traffic classification based on network addresses is not sufficient. Based on the current architecture, tenants can be distinguished from each other using their VIDs, which have been assigned by virtual and physical switches. These IDs are not essentially identical among switches. Moreover, this technique is useful when VLAN tagging has been employed for tenants' isolation. VLAN tagging may not be implemented everywhere, due to its disadvantages (e.g. VLAN table scalability) for an environment with a large number of customers. For instance, the FloodLight OpenFlow controller does not use VLAN tagging for creating isolated logical layer 2 networks.

The following part summarize challenges of network awareness in a cloud platform, when multiple tenants coexist.

Elasticity and Dynamicity of The Environment: Tenants provision resources on demand and release them when there is no more need. Provisioned resources can scale up or down to meet the requirements, and can be migrated to other physical substrates or regions. Scalability, elasticity, and migration of virtual resources in the cloud, make monitoring much harder. A tenant is not bounded to a particular region or hardware. Therefore, the monitoring solution must be able to provide the tenant based statistics, independent of the physical location.

Complex Stakeholder Relations: In the new computing model, several stakeholders are involved and their interactions are complicated. Traditional monitoring solutions are not adapted to this model, and they can not logically distinguish between tenants in a cloud environment.

Incident Handling and Network Forensics: In an incident, the forensic team asks for relevant information. If the incident happens in a tenant domain, only data related to that specific tenant should be exposed to the forensic team. Proper clustering and anonymization of data become crucial, due to the privacy and security of other tenants.

Reliable Monitoring and Non-Repudiability: A malicious tenant may try to forge its networking specifications. The motivation can be concealing its own identity or impersonating other tenants. Monitoring approaches should not solely rely on the information provided by tenants, but provide mechanisms for assessing the originality of data. For instance, IP addresses can be spoofed by a tenant, however, VIDs are assigned by switches out of tenants' domains.

Hardware and Software Limitations: Existing hardware and software may not fully support new protocols and standards (e.g. NetFlow v9 and OpenFlow are not supported widely). Proposed techniques for the new model should not rely on standards or protocols, which are not mature yet.

Invisibility of tenants' internal communications: Instances in a tenant communicate with each other or with external entities (e.g. other tenants, off-premises entities). Observation points must be chosen such that they cover all types of traffic.

Tenants' Interests for Monitoring Services: Some companies are interested in using external services and moving their resources off-premises. A big obstacle on the migration is the lack of visibility and control over resources. Offering tenant based monitoring services can relax the issue, and satisfies more customers.

5 Tenant Based Network Monitoring Solutions

There should be multiple techniques to fulfill requirements of a heterogeneous infrastructure. Four approaches in two categories are proposed for a tenant's network monitoring in a cloud infrastructure:

- Non-OpenFlow enabled components

  - Tenants monitoring using IP datagram header.
  - Tenants monitoring using data link frame header.
  - Tenants monitoring using virtual components in the network gateway.

- OpenFlow enabled components

Following sections discuss each approach and the advantages and disadvantages.

5.1 Non-OpenFlow enabled components

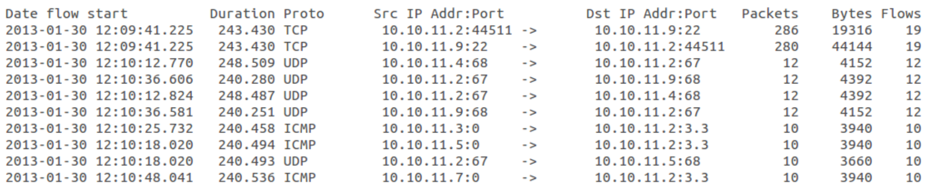
SDN is in its early stages, and OpenFlow capable components are the realization tools. Thus, we can not assume that all equipments in an infrastructure will support the standard in a near future. Three techniques are explained for answering the network monitoring demands. Although the technology behind these techniques are not new, they should be adapted to the new model.

Tenants monitoring using IP datagram header

Each tenant in a cloud platform can have several networks and subnets. The tenant's traffic can be identified using the IP addresses assigned to its subnets. The virtual switch (i.e. Open vSwitch) supports both sFlow and NetFlow v5 monitoring protocols. Most physical switches supports at least one of these protocols as well. Given that NetFlow is supported by all switches in an infrastructure, we can collect raw data and extract tenant's statistics from them.

Tenant's networks and subnets list can be retrieved from the networking service (e.g. OpenStack Quantum). Assigned IP addresses to the tenant is in the subnets information. The information can be used for processing the monitoring data that are collected from observation points. The analysis output shows the statistics for all instances which belongs to a particular tenant. For illustration, *Tenant A* is attached to two subnets, and we are interested in the subnet 10.10.11.0/24. Figure 2.5 shows *Tenant A*'s flows in all switches.

The implementation is simple, and requires several queries to the networking service and the monitoring data collector. Despite its simplicity, it's quite useful as it will preserve all capabilities provided by sFlow or NetFlow, as well as the visibility into tenants' traffic and behavior.



```
Date flow start          Duration Proto      Src IP Addr:Port          Dst IP Addr:Port   Packets    Bytes Flows
2013-01-30 12:09:41.225   243.430 TCP         10.10.11.2:44511 ->       10.10.11.9:22          286    19316    19
2013-01-30 12:09:41.225   243.430 TCP         10.10.11.9:22    ->       10.10.11.2:44511       280    44144    19
2013-01-30 12:10:12.770   248.509 UDP         10.10.11.4:68    ->       10.10.11.2:67           12     4152    12
2013-01-30 12:10:36.606   240.280 UDP         10.10.11.2:67    ->       10.10.11.9:68           12     4392    12
2013-01-30 12:10:12.824   248.487 UDP         10.10.11.2:67    ->       10.10.11.4:68           12     4392    12
2013-01-30 12:10:36.581   240.251 UDP         10.10.11.9:68    ->       10.10.11.2:67           12     4152    12
2013-01-30 12:10:25.732   240.458 ICMP        10.10.11.3:0     ->       10.10.11.2:3.3          10     3940    10
2013-01-30 12:10:18.020   240.494 ICMP        10.10.11.5:0     ->       10.10.11.2:3.3          10     3940    10
2013-01-30 12:10:18.020   240.493 UDP         10.10.11.2:67    ->       10.10.11.5:68           10     3660    10
2013-01-30 12:10:48.041   240.536 ICMP        10.10.11.7:0     ->       10.10.11.2:3.3          10     3940    10
```

Figure 2.5: Tenants monitoring using IP datagram header fields

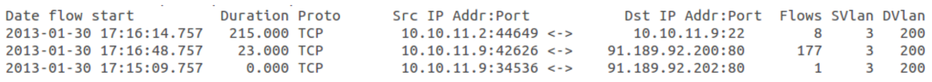
However, the statistics may not be accurate or reliable. Because tenants may have overlapping IP address or a nefarious tenant can forge IP addresses. Thus, a tenant can impersonate another one during an attack. Auditing mechanisms fail to distinguish between the attacker and the impersonated tenant. It leads to accountability issues and contradicts non-repudiability principle. The same scenario can be imagined for billing issues, when tenants are charged according to their traffic. Overlapping IP addresses lead to inconsistent billing information and traffic measurement. So, the solution should be extended to cover these shortcomings.

Tenant monitoring using data link frame header

Monitoring tenants' traffic using VLAN information is more robust. VLAN tags are located in the data link layer frame header, and tenants can not forge them. Since, the virtual switch maintains a binding between instances in a particular tenant and the VID. In addition, using data link layer frame information makes the solution independent of network layer protocols. Thus, it will not be limited to IP.

As it was explained earlier, the virtual switch supports sFlow and NetFlow v5. NetFlow v5 is not suitable, as it doesn't deliver VLAN information, as shown in Table 2.1. For this approach sFlow or NetFlow v9/IPFIX must be used.

The networking service stores the mapping of tenants and physical switches VIDs, but tenants' VIDs in virtual switches are not known to it. An agent in each compute node should expose the VID mapping (i.e. tenant : VID : virtual switch ID). Then, the mapping can be used for processing sFlow monitoring data. The implementation is not trivial, because tenants' VIDs may not be identical among switches. First, we identify tenants' VIDs in each switch. Then, a query is created for aggregating the statistics from all virtual switches. As an example, *Tenant A* is using VID 3 in the virtual switch OVS-ha13 and VID 200 on the physical switch (Figure 2.6).



```
Date flow start          Duration Proto      Src IP Addr:Port          Dst IP Addr:Port  Flows SVlan DVlan
2013-01-30 17:16:14.757   215.000 TCP         10.10.11.2:44649 <->      10.10.11.9:22          8     3   200
2013-01-30 17:16:48.757    23.000 TCP         10.10.11.9:42626 <->    91.189.92.200:80       177     3   200
2013-01-30 17:15:09.757     0.000 TCP         10.10.11.9:34536 <->    91.189.92.202:80         1     3   200
```

Figure 2.6: Tenant monitoring using data link frame header fields

Tenant monitoring using virtual components in the network gateway

According to the architecture, tenants have their own dedicated network name-spaces in the network gateway. Tenant's virtual routers and interfaces are visible in the its name-space. Components in the tenant's name-space are responsible for delivering networking services such as DHCP and layer 3 connectivity.

These components (in the hatched area inside the network gateway in Figure 2.2) can be considered as observation points (i.e. `Route1, tapZZ, qr-xx, qg-yy`). But only a part of the tenant traffic is passing through these points, including: DHCP traffic, ingress/egress traffic from/to tenant's network. In other words, internal communication of instances inside a tenant is not observable. Therefore, it has several disadvantages: *a)* monitoring these points will not provide complete visibility of the tenant's traffic, *b)* tenant's traffic in each virtual switch is not distinguishable, *c)* deploying multiple redundant network gateways can disrupt the visibility and will require additional data correlations.

Although the approach has multiple drawbacks, its benefits make it interesting to be offered as a service. The implementation is simple and scales well. It has low overhead and the resource consumption is considerably less than other solutions. A tenant can use the service for monitoring communications with other networks. Networks can be either other tenants' networks or destinations outside the cloud platform. This is an efficient solution when all internal entities inside a tenant are trusted, or when there is no interest in the internal communication.

5.2 OpenFlow enabled components

The networking service is provided by the Quantum project, that means all virtual switches are configured by the Quantum. Although Open vSwitch supports OpenFlow, activating the controller in the virtual switch overwrites the previous configuration. Thus, we need to commit Quantum's command through an OpenFlow controller. In this architecture, the controller is connected to all capable switches and has a unified view

Bytes	*0-3*	*4-7*	*8-11*	*12-13*	*14-15*	*16-19*	*20-23*	*24-27*
Contents	srcadd	dstaddr	nexthop	input	output	dPkts	dOcts	first
Bytes	*28-31*	*32-33*	*34-35*	*36*	*37*	*38*	*39*	*40-41*
Contents	last	srcport	dstport	pad1	tcp_flags	prot	tos	src_as
Bytes	*42-43*	*44*	*45*	*46-47*				
Contents	dst_as	src_mask	dst_mask	pad2				

Table 2.1: NetFlow version 5 Flow Record Format [9]

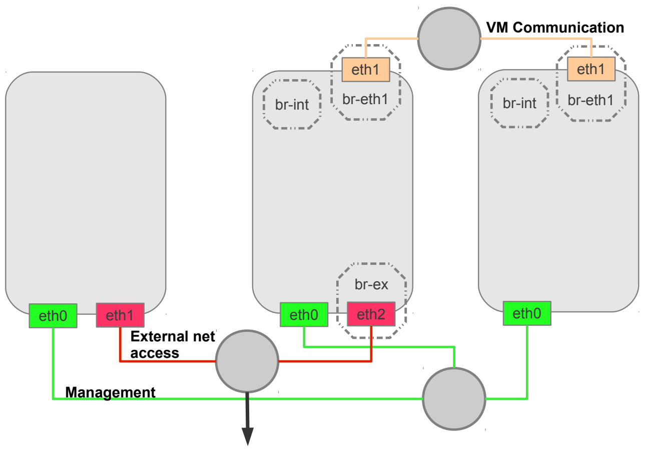


Figure 2.7: OpenFlow enabled Lab Architecture

of the network. Therefore, the monitoring node communicates with the controller to build per-tenant view of the network and generates monitoring information for each tenant, Figure 2.7.

There are multiple Quantum plug-ins providing this functionality, such as: Ryu OpenFlow Controller plugin[8], NEC OpenFlow plugin[9], Floodlight OpenFlow Controller[10] plugin. The most suitable controller, for our use-case in this period, is Floodlight controller. Floodlight has an application module called Virtual Network Filter (VNF) that provides virtualized layer 2 networks over a single layer 2 domain. VNF listens for PacketIn messages, and verifies the source and destination MAC addresses. If they belongs to the same virtual network, the packet will pass; otherwise it

[8]`http://www.osrg.net/ryu/`

[9]http://wiki.openstack.org/Quantum-NEC-OpenFlow-Plugin

[10]http://floodlight.openflowhub.org/

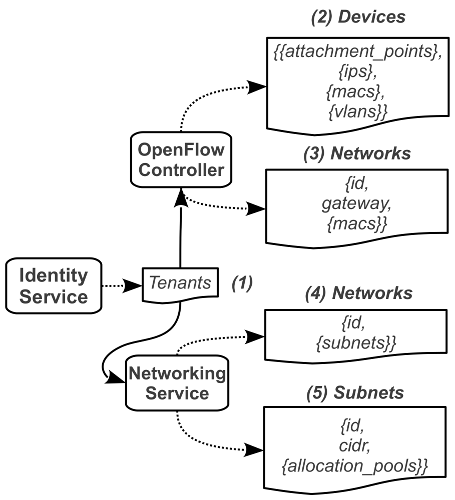


Figure 2.8: Mapping switch ports to tenants

will be dropped. This is a simple approach with some limitations, such as: virtual networks can only be defined in a single physical layer 2 domain, and limited number of gateways in a tenant network[17].

The list of tenants are retrieved from the identity service (Figure 2.8 (1)), and their networks and subnets are loaded from the networking service (Figure 2.8 (4, 5)). Then, we need to find devices in the network and their attachment points (Figure 2.8 (2, 3)). Finally, the information required for monitoring tenants' network activity is ready. The information is in form of:

$tenant_a : \{device_x : \{(mac_\alpha, ip_\alpha, port_\alpha, switch_\alpha)\}\}$

Multiple types of report can be generated, including: *a)* a simple view of a tenant's traffic in the network by aggregating statistics of ports belonging to the same tenant, *b)* a sophisticated tenant's flow information by analysing NetFlow data using the information provided by the controller. The proof of concept code is developed and results are satisfactory.

6 Discussion

There are several shortcomings in the design of existing monitoring tools, such as: *a)* lack of tenant logic with complex stakeholder relationships, *b)* lack of flexibility for the new characteristics of the cloud model (e.g. elasticity, on-demand provisioning/release), and *c)* lack of robustness for a heterogeneous infrastructure. Four approaches in two category are discussed to cover requirements imposed by the model's criteria: Non-OpenFlow approaches, and the OpenFlow enabled approach. Non-OpenFlow enabled solutions use tenants' internet layer attributes (e.g. IPs), data link frame attributes (e.g. VLAN IDs), virtual devices in the Linux network stack namespace.

The first category uses core cloud platform APIs and standard monitoring data. Required parameters for building a tenant-based view are gathered, and filters are created. Parameters define networking attributes of a tenant, such as IP subnets, VIDs, and MAC addresses. Then, monitoring data are analyzed using these parameters, and a tenant-based view of the network activities is created. The second category benefits from the OpenFlow controller's capabilities. The controller is responsible for the control plane of switches in the network, and it is aware of the tenant logic in the platform. Tenant awareness of the controller helps in developing a robust solution for recognizing and distinguishing tenants' networking behaviors.

Advantages, disadvantages, and requirements of of proposed solutions are briefly explained in Table 2.2.

7 Conclusion

In a multi-tenant model like the cloud computing, several stakeholders are involved. Distinguishing tenants' activities and provisioned resources, in the time domain, is the key factor for accountability, anomaly detection, as well as load-balancing. We have started with analysing the IaaS service model, and several stakeholders in the provider layers are identified. At each layer, monitoring can be performed with different granularities. Depending on the coarseness, the result exposes various characteristics of the system.

The architecture is powered by virtual switches and OpenFlow controllers. Two types of solutions are introduced, addressing a heterogeneous environment. The first type introduces methods for monitoring tenants' activities, when an OpenFlow controller is not available. In these methods, tenants' specifications are retrieved from the networking and identity service. Then, raw monitoring data are processed for building per-tenant traffic statistics. The second type benefits from an OpenFlow controller to build the per-tenant view. In this approach, the controller provides the unified view of the network, and is aware of the tenant logic.

Table 2.2: Solution comparison (NS: Networking Service, IS: Identity Service, VS: Virtual Switch, NG: Network Gateway)

Solution	**Requirement**	**Advantages**	**Disadvantages**
IP header	NetFlow data NS access IS access	+ Ease of implementation + Per VS statistics + Complete view of tenants IP activity	- Prone to IP spoofing - Lack of accuracy
Data Link header	sFlow data VS monitoring agent NS access IS access	+ Protocol independence + Accuracy + Per VS statistics + Reliable against IP spoofing + Complete view of tenants network activity	- NetFlow v5 incompatibility - VS agent overhead
Network gateway	NetFlow data NG monitoring agent NS access IS access	+ Simplicity + Protocol independence + Reliable against IP spoofing + Complete view of ingress/egress traffic	- No per VS statistics - No visibility into VN internal traffic - NG agent overhead
OpenFlow Controller	OpenFlow Controller NS OpenFlow agent NetFlow data NS access IS access	+ Transparent integration + Protocol independence + Accuracy + Flexibility + Reliable against IP spoofing + Complete view of tenants network activity	- Implementation complexity - Controller dependency

An obstacle for deploying these mechanisms in a large scale production environment, is the storage and processing of large data sets. We will continue to enhance our monitoring solution by collecting flow information in a distributed data-intensive processing framework. Thus, we will be able to divide a query task and execute it over a cluster of commodity servers. This leads to efficient statistical analysis of monitoring data. The result will be used for providing real-time feedback to the OpenFlow controller for updating networking decisions.

Acknowledgment

The authors would like to thank Martin Gilje Jaatun from SINTEF ICT, who provided valuable comments and assistance to the undertaking of this research.

References

[1] T. Anderson et al. “Overcoming the Internet impasse through virtualization.” In: *Computer* 38.4 (Apr. 2005), pp. 34–41.

[2] Michael Armbrust et al. “A view of cloud computing.” In: *Communications of the ACM* 53.4 (Apr. 2010), p. 50.

[3] Jeffrey R. Ballard, Ian Rae, and Aditya Akella. “Extensible and scalable network monitoring using OpenSAFE.” In: *Proceedings of the 2010 internet network management conference on Research on enterprise networking.* INM/WREN’10. 2010, pp. 8–8.

[4] Robin Berthier et al. “Nfsight: netflow-based network awareness tool.” In: *Proceedings of the 24th international conference on Large installation system administration.* LISA’10. 2010, pp. 1–8.

[5] Rodrigo Braga, Edjard Mota, and Alexandre Passito. “Lightweight DDoS flooding attack detection using NOX/OpenFlow.” In: *Local Computer Networks (LCN), 2010 IEEE 35th Conference on.* LCN’10. Oct. 2010, pp. 408–415.

[6] Andrew T. Campbell et al. “A survey of programmable networks.” In: *ACM SIGCOMM Computer Communication Review* 29.2 (Apr. 1999), p. 7.

[7] N.M. Mosharaf Kabir Chowdhury and Raouf Boutaba. “A survey of network virtualization.” In: *Computer Networks* 54.5 (Apr. 2010), pp. 862–876.

[8] N.M.M.K. Chowdhury and R. Boutaba. “Network virtualization: state of the art and research challenges.” In: *IEEE Communications Magazine* 47.7 (July 2009), pp. 20–26.

[9] Cisco Systems. *NetFlow FlowCollector Installation and User’s Guide.* Tech. rep. OL-2587-01. 1999.

[10] B. Claise. *Cisco Systems NetFlow Services Export Version 9.* RFC 3954 (Informational). Internet Engineering Task Force, Oct. 2004. URL: http://www.ietf.org/rfc/rfc3954.txt.

[11] B. Claise. *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information.* RFC 5101 (Proposed Standard). Internet Engineering Task Force, Jan. 2008. URL: http://www.ietf.org/rfc/rfc5101.txt.

[12] Dmitry Drutskoy, Eric Keller, and Jennifer Rexford. “Scalable Network Virtualization in Software-Defined Networks.” In: *IEEE Internet Computing* 17.2 (2013), pp. 20–27.

[13] Nicolas Falliere, Liam O Murchu, and Eric Chien. *W32.Stuxnet Dossier*. Technical. Version 1.4. Symantec, Feb. 2011. URL: http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf.

[14] Paul Ferguson and Geoff Huston. “What Is a VPN?” In: *The Internet Protocol Journal* 1.1 (1998). URL: http://www.cisco.com/web/about/ac123/ac147/archived_issues/ipj_1-1/what_is_a_vpn.html.

[15] “IEEE Standard for Local and Metropolitan Area Networks Virtual Bridged Local Area Networks.” In: *IEEE Std 802.1Q-2005 (Incorporates IEEE Std 802.1Q1998, IEEE Std 802.1u-2001, IEEE Std 802.1v-2001, and IEEE Std 802.1s-2002)* (2006), pp. 1–285.

[16] Lavanya Jose, Minlan Yu, and Jennifer Rexford. “Online measurement of large traffic aggregates on commodity switches.” In: *Proceedings of the 11th USENIX conference on Hot topics in management of internet, cloud, and enterprise networks and services*. Hot-ICE’11. 2011, pp. 13–13.

[17] Bethany Kanui. *FloodLight VirtualNetworkFilter*. Oct. 2012. URL: http://www.openflowhub.org/display/floodlightcontroller/.

[18] Bob Lantz, Brandon Heller, and Nick McKeown. “A Network in a Laptop: Rapid Prototyping for Software-defined Networks.” In: *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. Hotnets-IX. 2010, pp. 1–6.

[19] Nick McKeown. “Software Defined Networking.” IEEE InfoCom. 2009. URL: http://klamath.stanford.edu/~nickm/talks/infocom_brazil_2009_v1-1.pdf.

[20] Nick McKeown et al. “OpenFlow: enabling innovation in campus networks.” In: *ACM SIGCOMM Computer Communication Review* 38.2 (Mar. 2008), p. 69.

[21] Peter Phaal and Marc Lavine. *sFlow Version 5*. Tech. rep. July 2004. URL: http://www.sflow.org/sflow_version_5.txt.

[22] Amin Tootoonchian, Monia Ghobadi, and Yashar Ganjali. “OpenTM: traffic matrix estimator for OpenFlow networks.” In: *Proceedings of the 11th international conference on Passive and active measurement*. PAM’10. 2010, pp. 201–210.

[23] B. Trammell and E. Boschi. *Bidirectional Flow Export Using IP Flow Information Export (IPFIX)*. RFC 5103 (Proposed Standard). Internet Engineering Task Force, Jan. 2008. URL: `http://www.ietf.org/rfc/rfc5103.txt`.

[24] Curtis Yu et al. “FlowSense: Monitoring Network Utilization with Zero Measurement Cost.” In: *Proceedings of the 14th International Conference on Passive and Active Measurement*. PAM’13. 2013, pp. 31–41.

Paper 2:
Real-Time Handling of Network Monitoring Data Using a Data-Intensive Framework

Real-Time Handling of Network Monitoring Data Using a Data-Intensive Framework

A. TaheriMonfared[1], T.W. Wlodarczyk[1], C. Rong[1]

[1] Department of Electrical Engineering and Computer Science, University of Stavanger
4036 Stavanger, Norway

Abstract:
The proper operation and maintenance of a network requires a reliable and efficient monitoring mechanism. The mechanism should handle large amount of monitoring data which are generated by different protocols. In addition, the requirements (e.g. response time, accuracy) imposed by long-term planned queries and short-term ad-hoc queries should be satisfied for multi-tenant computing models.
This paper proposes a novel mechanism for scalable storage and real-time processing of monitoring data. This mechanism takes advantage of a data-intensive framework for collecting network flow information records, as well as data points' indexes. The design is not limited to a particular monitoring protocol, since it employs a generic structure for data handling. Thus, it's applicable to a wide variety of monitoring solutions.

1 Introduction

Monitoring and measurement of the network is a crucial part of infrastructure operation and maintenance. A good understanding of the traffic passing through the network is required for both planned and ad-hoc tasks. Capacity planning and traffic matrix processing are planned, whereas traffic engineering, load-balancing, and intrusion detection are ad-hoc tasks which often require real-time behavior.

Storage Requirements

Quite often, ad-hoc tools are used for analysing network properties [12]. Traffic dumps or flow information are common data type for an ad-hoc analysis. The data volume for these types can be extremely large.

Analytic Requirements

The storage should be distributed, reliable, and efficient to handle high data input rate and volume. Processing this large data set for an ad-hoc query should be near real-time. It should be possible to divide and distribute the query over the cluster storing the data.

Privacy Policies

Storing the packet payload which corresponds to the user data is restricted according to European data laws and regulations [14]. The same policy applies to the flow information as well.

1.1 Related Work

Li et al. [24] surveyed the state of the art in flow information applications. They identified several challenges in the fields such as: machine learning's feature selection for an effective analysis, real-time processing, and efficient storage of data sets. Lee et al. [23] proposed a mechanism for importing network dumps (i.e. libpcap files) and flow information to HDFS. They've implemented a set of statistical tools in MapReduce for processing libpcap files in HDFS. The tool set calculates statistical properties of IP, TCP, and HTTP protocols. Their solution copies recently collected NetFlow data to Hive tables in fixed intervals which doubles the storage capacity requirement. Andersen et al. [1] described the management of network monitoring datasets as a challenging task. They emphasized on the demand for a data management framework with the eventual consistency property and the real-time processing capability. The framework should facilitate search and discovery by means of an effective query definition and execution process. Balakrishnan et al. [3] and Cranor et al. [12] proposed solutions for the real-time analysis of network data streams. However, they may not be efficient for the analysis of high-speed streams in a long period [1].

1.2 Contributions

A flexible and efficient mechanism is designed and implemented for real-time storage and analysis of network flow information. In contrast to other solutions, which have analyzed binary files on distributed storage systems, a NoSQL type of data store provides real-time access to a flexible data model. The data model flexibility makes it compatible with different monitoring protocols. Moreover, the structure leads to fast scanning of a small part of a large dataset. This property provides low latency responses which facilitate exploratory and ad-hoc queries for researchers and administrators. The solution provides a processing mechanism which is about 4000 times faster than the traditional one.

The study concentrates on flow information records, due to regulatory and practical limitations such as privacy directives and payload encryption. However, one can leverage the same solution for handling wider and richer datasets which contain application layer fields. This study is a part of our tenant-aware network monitoring solution for the cloud model.

The rest of the paper is organized as follows: Section 2 explains the background information about data-intensive processing frameworks and network monitoring approaches. Section 3 describes the Norwegian NREN backbone network as a case study. Dataset characteristics and monitoring requirements of a production network are explained in this section. Section 4 introduces our approach toward solving data processing challenges for network monitoring. Section 5 discusses technical details of the implementation as well as performance tuning for improving the efficiency. Section 6 evaluates the solution by performing common queries and Section 7 concludes the paper and introduces future works.

2 Background

2.1 Framework for Data-Intensive Distributed Applications

Using commodity hardware for storing and processing large sets of data is becoming very common [21]. There are multiple proprietary, open-source frameworks and commercial services providing similar functionality such as: Apache's Hadoop[11] [28] and related projects, Google's File System (GFS) [17], BigTable [9], Microsoft's Scope [8], Dryad [19]. In the following, required components for the analysis and storage of our dataset is explained.

File System (Hadoop Distributed FS)

The first building block of our solution, for handling network monitoring data, is a proper file system. The chosen file system must be reliable, distributed and efficient for large data sets. Several file systems can fulfill these requirements, such as Hadoop Distributed File System (HDFS) [28], MooseFS[12], GlusterFS[13], Lustre[27], Parallel Virtual File System (PVFS)[7]. Despite the variety, most of these file systems are missing an integrated processing framework, except HDFS. This capability in HDFS makes it a good choice as the underlying storage solution.

Data Store (HBase)

Network monitoring data, and packet header information are semi-structured data. In a short period after their generation, they're accessed frequently, and a variety of information may be extracted from them. Apache HBase[14][16] is the most suitable non-relational data store for this specific use-case. HBase is an open-source implementation of a column-oriented distributed data source inspired by Google's BigTable [9], which can leverage the MapReduce processing framework of Apache. Data access in HBase is key-based. It means a specific key or a part of it can be used to retrieve a cell (i.e. a record), or a range of cells [16]. As a database system, HBase guarantees consistency and partition tolerance from the CAP theorem [5] (also known as Brewer's theorem).

[11]`http://hadoop.apache.org/`
[12]`http://www.moosefs.org/`
[13]`http://www.gluster.org/`
[14]`http://hbase.apache.org/`

Processing Framework (Hadoop MapReduce)

Processing large data sets has demanding requirements. The processing framework should be able to partition the data across a large number of machines, and exposes computational facilities for these partitions. The framework should provide the abstraction for parallel processing of data partitions and tolerate machine failures. MapReduce [13] is a programming model with these specifications. Hadoop is an open source implementation by Apache Software Foundation, which will be used in our study.

2.2 Network Monitoring

This study focuses on the monitoring of backbone networks. The observation can be instrumented using Simple Network Management Protocol (SNMP) metrics, flow information (i.e. packet header), and packet payload. SNMP does not deliver the granularity demanded by our use-case; also storing packets payloads from a high capacity network is not feasible, because of both scalability issues [12] and privacy policies [14]. Thus, we are more interested in the packet header, and IP flow information. An IP flow is a set of packets passing through a network between two endpoints, and matching a certain set of criteria, such as one or more identical header fields [11]. In our study, a flow is a canonical five-tuple: source IP, source port, destination IP, destination port, and protocol. Flow information is flushed out of the network device after 15 seconds of inactivity, 30 minutes of persistent activity, TCP session termination, or when the flow buffer in the device is full. This makes the start and end time of a flow imprecise [22]. IP flow information is an efficient data source for the real-time analysis of network traffic.

IP flow information can be exported using different protocols, in different formats. NetFlow [10], sFlow [26], and IP Flow Information Export (IPFIX) [11] are designed to handle network monitoring data. Collected data have a variety of use-cases. They can be used for security purposes, audit, accountability, billing, traffic engineering, capacity planning, etc.

2.3 Testing Environment

We have implemented, optimized, and tested our suggested solution. The testing environment consists of 19 nodes, which deliver Hadoop, HDFS, HBase, ZooKeeper, Hive services. The configuration for these nodes is as follows: 6x core AMD Opteron(tm) Processor 4180, 4x 8GB DDR3 RAM, 2x 3 TB disks, 2x Gigabit NIC.

3 Case Study: Norwegian National Research and Education Network (NREN)

This study focuses on the storage and processing of IP flow information data for the Norwegian NREN backbone network. Two core routers, TRD_GW1 (in Trondheim) and OSLO_GW (in Oslo), are configured to export flow information. Flow information are collected using NetFlow [10] and sFlow [26].

3.1 Data Volume

Flow information is exported from networking devices at different intervals or events (e.g. 15 seconds of inactivity, 30 minutes of activity, TCP termination flag, cache exhaustion). The data are collected in observation points, and then the anonymized data are stored for experiments. Crypto-PAn [15] is used for the data anonymization. The mapping between the original and anonymized IP address is ”one-to-one”, ”consistent across traces”, and ”preserves prefix”.

Flow information is generated by processing a sampled set of packets. Although sampled data is not as accurate as not-sampled one, studies showed they can be used efficiently for network operation and anomaly detection, by means of right methods [4, 6].

There need to be a basic understanding of the dataset for designing the proper data store. Data characteristics, common queries and their acceptable response times are influential factors in the schema design. The identifier for accessing the data can be on one or more fields from the flow information record (e.g. source or destination IP addresses, ports, Autonomous Systems (AS), MACs, VLANs, interfaces). Figure 2.1 depicts number of unique *source*, *destination IP addresses*, unique *source IP:source port*, *destination IP:destination port* tuples, unique *bidirectional flows (biflows)*, and *flow information records* per day for the TRD_GW1 in a 5 month period. The summary of numeric values for TRD_GW1 and OSLO_GW is presented in Table 2.1.

The average number of flow information records for both routers is 22 millions per day, which corresponds to 60 GBs of data in binary form. However, this number can become much bigger if flow information are collected from more sources and the sampling rate is increased.

Table 2.1: Traffic Characteristics

Traffic Type	Statistics/day		
	Avg	Max	Min
Distinct Src IPs	987104	4740760	122266
Distinct Src IPs and Src ports	6083640	13188647	844898
Distinct Dst IPs	1613040	2488893	420686
Distinct Dst IPs and Dst ports	7010330	16379274	1113095
Distinct Bidirectional flows	10683200	21454096	1829854
NetFlow records	21962800	44036078	4373665

3.2 Data Access Methods

Monitoring data can be accessed for different purposes such as: billing information, traffic engineering, security monitoring, forensics. These purposes corresponds to a big set of possible queries. The schema can be design such that it performs very well for one group of queries. That may lead to a longer execution time for the other query groups. Our main goal is reaching the shortest execution time for security monitoring and forensics queries. Three types of queries are studied, *IP based*: requires fast IP address lookups (e.g. specific IPs, or subnets), *Port based*: requires fast Port address lookups (e.g. specific services), and *Time based*: requires fast lookup on a time period.

Network monitoring data, and packet header information are semi-structured data. They have arbitrary lengths and a various number of fields. Storing this type of data as binary files in a distributed file system is challenging. The next section discusses several storage schemas and their applicability to desired access methods.

4 Solution

Two major stages in the life cycle of the monitoring data can be considered: short-term processing, and long-term archiving.

- Short-term processing: when collected monitoring data are imported into the data store, several jobs should be executed in real-time. These jobs generate real-time network statistics, check for anomaly patterns and routing issues, aggregate data based on desired criteria,

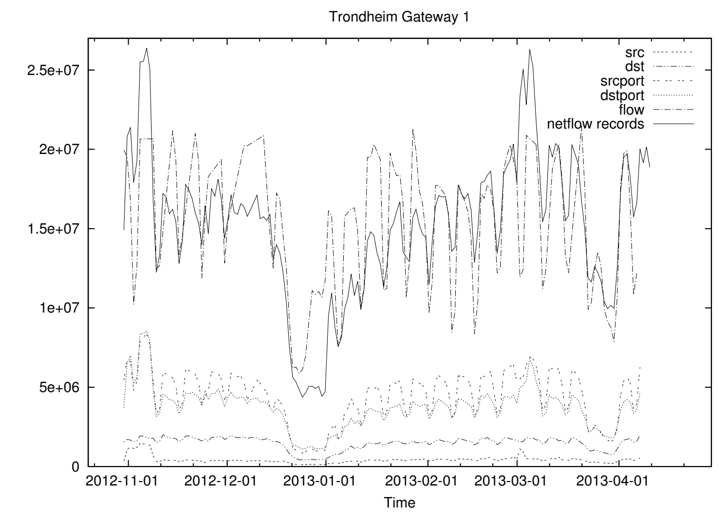


Figure 2.1: Number of distinct source IPs, source IPs: source ports, destination IPs, destination IPs:destination ports, bidirectional flows, and raw netflow records collected from Trondheim gateway 1.

and etc.

- Long-term archiving: Archived data can be accesses for security forensics, or on-demand statistical analysis.

4.1 Choice of Technologies

Apache HBase satisfies our requirements (Section 1) such as consistency and partition tolerance. Moreover, the data staging is affordable by proper configuration of cache feature, in-memory storage size, in-filesystem storage size, regions configuration and pre-splitting for each stage, and etc. For instance, short-term data can be stored in regions with large memory storage and enabled block cache. The block cache should be configured such that the Working Set Size (WSS) fits in memory [2]. While long-term archives are more suitable for storage in the filesystem.

Hive[15] is an alternative for HBase, which is not suitable for our application. It doesn't support binary key-values, and all parameters are stored as strings. This approach demands for more storage, and makes the implementation inefficient. While composite key structure is an important factor for fast data access in the design, it is not supported by Hive.

[15]`http://hive.apache.org/`

Although Hive provides an enhanced query mechanisms for retrieving data, aforementioned issues make it inapplicable to our purpose.

4.2 Design Criteria

A table schema in HBase has three major components: rowkey, column-families, and columns structures.

Row Key

A rowkey is used for accessing a specific part of data or a sequence of them. It is a byte array which can have a complex structure such as a combination of several objects. Rowkey structure is one of the most important part of our study because it has a great impact on the data access time, and storage volume demand. The followings are our criteria for designing the rowkey:

- **Rowkey Size**: Rowkey is one of the fields stored in each cell, and is a part of a cell coordinate. Thus, it should be as small as possible, while efficient enough for data access.
- **Rowkey Length (Variable versus Fixed)**: Fixed length rowkeys, and fields help us to leverage the lexicographically sorted rows in a deterministic way.
- **Rowkey Fields' Order (with respect to region load)**: Records are distributed over regions based on regions' key boundaries. Regions with high loads can be avoided by a uniform distribution of rowkeys. Thus, the position of each field in the rowkey structure is important. Statistical properties of a field's value domain are determining factors for the field position.
- **Rowkey Fields' Order (with respect to query time)**: Lexicographic order of rowkeys makes queries on the leading field of a rowkey much faster than the rest. This is the motivation for designing multiple tables with different fields order. Therefore, each table provides fast scanning functionality for a specific parameter.
- **Rowkey Fields' Type**: Fields of a rowkey are converted to byte arrays then concatenated to create the rowkey. Fields' types have significant effect on the byte array size. As an example number 32000

can be represented as a *short* data type or as a *string*. However, the *string* data type require two times more number of bytes.

- **Rowkey Timestamps vs. Cell Version**: It's not recommended to set the maximum number of permitted versions too high [2]. Thus, there should be a timestamp for the monitoring record as a part of the rowkey.
- **Timestamps vs. Reverse Timestamps**: In the first stage of data life cycle, recent records are frequently accessed. Therefore, revere timestamps are used in the rowkey.

Column Families

Column families are the fixed part of a table which must be defined while creating the schema. It's recommended to keep the number of families less than three, and those in the same table should have similar access patterns and size characteristics (e.g. number of rows) [16]. Column family's name must be of string type, with a short length. The family's name is also stored in the cell, as a part of the cell coordination. A table must have at least one column family, but it can have a dummy column with an empty byte array. We have used constant value D for our single column family across all tables.

Columns

Columns are the dynamic part of a table structure. Each row can have its own set of columns which may not be identical to other rows' columns. Monitoring data can be generated by different protocols, and they may not have similar formats/fields. Columns make the solution flexible and applicable to a variety of monitoring protocols.

There are several tables with different fields' orders in rowkeys, but not all of them have columns. Complete monitoring record is just inserted into the reference table, and others are used for fast queries on different rowkey fields.

4.3 Schemas

Section 3.2 explained desired query types and Section 4.2 described required properties of a rowkey for fast scanning. Here, three table types are introduced, each addressing one query category: IP-based, Port-based, and Time-based tables.

IP Based Tables

T1 (reference table), T2 The rowkey of this table consists of: *source IP address, source port, destination IP address, destination port,* and *reverse timestamp* (Table 2.2). Columns in this family are flexible and any given set can be stored there. Column qualifiers identifier are derived from fields' names, and their values are corresponding values from flow information records. Other tables are designed as secondary indexes. They improve access time considerably for the corresponding query group. Table T1 is used for retrieving flow information parameters that are not in the rowkey (e.g. number of sent or received packets, bytes, flows)

Table T2 has destination address and port in the lead position. This is used in combination with T1 for the analysis of bidirectional flows.

T3, T4 are suitable when source and destination addresses are provided by the query (Table 2.2). For instance, when two ends of a communication are known, and we want to analyze other parameters such as: communication ports, traffic volume, duration, etc.

Port Based Tables

T5, T6 are appropriate tables for service discovery (Table 2.2). As an example, when we want to discover all nodes delivering SSH service (on default port: 22), we can specify the lead fields on T5 and T6 (source and destination ports), and let the data store returns all service providers and their clients. If the client *c1* is communicating on the port *p1* with the server *s1* on the port 22 at time *ts*, then there is a record with the rowkey: *[22][s1][c1][p1][1-ts]* in the data store.

T7, T8 can fulfill the requirement for identifying clients who use a particular service (Table 2.2). The same record from T5, T6 will have the rowkey: *[22][c1][s1][p1][1-ts]*.

Table 2.2: IP Based and Port Based Tables

Table	Row Key					Query Type
T1	[sa]	[sp]	[da]	[dp]	[1 - ts]	Extended queries
T2	[da]	[dp]	[sa]	[sp]	[1 - ts]	
T3	[sa]	[da]	[sp]	[dp]	[1 - ts]	Src-Dst address queries
T4	[da]	[sa]	[dp]	[sp]	[1 - ts]	Src-Dst address queries
T5	[sp]	[sa]	[da]	[dp]	[1 - ts]	Service server discovery queries
T6	[dp]	[da]	[sa]	[sp]	[1 - ts]	Service server discovery queries
T7	[sp]	[da]	[sa]	[dp]	[1 - ts]	Service client discovery queries
T8	[dp]	[sa]	[da]	[sp]	[1 - ts]	Service client discovery queries

Time Based Tables

OpenTSDB[16] is used for storing time series data. This can be an efficient approach for accessing and processing flows of a specific time period. A rowkey in OpenTSDB consists of: a metric, a base timestamp, and a limited number of tags in the key-value format. Source and destination IP addresses and ports are represented as tags, and a set of metrics are defined. Five fields from the flow information record are chosen as metrics: number of input and output bytes, input and output packets, and flows.

4.4 Storage Requirement

The storage volume required for storing a single replication of a not-compressed record can be estimated using Equation (2.1) as depicted in Table 2.3. However, this estimation may vary considerably if protocols other than NetFlow v5 and sFlow are used for collecting monitoring data (e.g. IPFIX raw record can be 250 bytes, containing 127-300 fields.)

Equation (2.2) is used for calculating the required capacity for tables T2-T8 (See Table 2.3). These tables don't have columns and values, which makes them much smaller than Table 1.

[16]`www.opentsdb.net`

Table 2.3: Storage requirements IPv4

	Est. # records	Storage for T1	Storage for T2-T8	Storage for OpenTSDB	Total
Single Record	1	$(37 * 23) + (133)$ $\sim 1KB$	$7tables * 23B$ $= 161B$	$5metrics * 2B$ $= 10B$	$\sim 1KB$
Daily Import	$\sim 20million$	$1KB * 20 * 10^6$ $= 20GB$	$161B * 20 * 10^6$ $\sim 3GB$	$10B * 20 * 10^6$ $= 200MB$	$\sim 23GB$
Initial Import	$20m * 150days$ $\sim 3 * 10^9$	$1KB * 3 * 10^9$ $= 3TB$	$161B * 3 * 10^9$ $\sim 500GB$	$10B * 3 * 10^9$ $= 30GB$	$\sim 3.5TB$

$$|record_{T1}| = |cq| * (|rk| + |cfn| + |cn|) + \sum_{i \in cq} |cv_i|) \tag{2.1}$$

$$|record_{T2-T8}| = (|rk| + |cfn|) \tag{2.2}$$

where:

$\lvert x \rvert$	= x's size in byte(s)
rk	= row key (size = 23B)
cfn	= column family name
cq	= column qualifiers set
cn	= column qualifier name
cv	= column value

5 Implementation

5.1 Data Collection

A set of MapReduce jobs and scripts are developed for collecting, storing, and processing data in HBase and OpenTSDB[17]. In the MapReduce job, the map task read flow information files and prepare rowkeys as well as columns for all tables. In the next step they are written into the corresponding tables. After that another task checks data integrity by a simple row counting job. This verification is not fully reliable, but it is a basic step for the integrity check without scarifying performance.

Performance evaluation was performed by processing records of a single day. The day is chosen randomly from working days of 2013. The statistical characteristics of the chosen day represents properties of any other working

[17]Available at: `https://github.com/aryantaheri/netflow-hbase`

days. The performance of the implementation is not satisfactory in this stage. For HBase, the maximum number of operations per second is 50 with the maximum operation latency of 2.3 seconds. HDFS shows the same performance issue, the maximum number of written bytes per second is 81 MB/s. The task is finished after 45.46 minutes. Therefore, a performance tuning is required.

5.2 Performance Tuning

The initial implementation of the collection module was not optimized for storing large datasets. By investigating performance issues, seven steps are recognized as remedies [20, 2]. These improvements will also enhance query execution process, and are applied there as well.

Using LZO compression Although compression demands more CPU time, the HDFS IO and network utilization are reduced considerably. Compression is applied to store files (HFiles) and the algorithm must be specified in the table schema for each column family. The compression ratio is dependent on the algorithm and the data type, and for our dataset with the LZO algorithm the ratio is about 4.

Disabling Swap Swappiness is set to zero on data nodes, since there is enough free memory for the job to complete without moving memory pages to the swap [20].

Disabling Write Ahead Log (WAL) All updates in a region server are logged in WAL, for guaranteeing durable writes. However, the write operation performance is improved significantly by disabling it. This has the risk of data loss in case of a region server failure [2].

Enabling Deferred Log Flush (DLF) DLF is a table property, for deferring WAL's flushes. If WAL is not disabled (due to the data loss risk), this property can specify the flushing interval to moderate the WAL's overhead [2].

Increasing heap size 20TB of the disk storage is planned to be used for storing monitoring data. The formula for calculating the estimated ratio of disk space to heap size is: $RegionSize/MemstoreSize * ReplicationFactor *$

HeapFractionForMemstores [18]. This leads to a heap size of 10GB per region server.

Specifying Concurrent-Mark-Sweep Garbage Collection (CMS-GC) Full garbage collection has tremendous overhead and it can be avoided by starting the CMS process earlier. Initial occupancy fraction is explicitly specified to be 70 percent. Thus, CMS starts when the old generation allocates more than 70 percent of the heap size [20].

Enabling MemStore-Local Allocation Buffers (MSLAB) MSLAB relaxes the issue with the old generation heap fragmentation for HBase, and makes garbage collection pauses shorter. Furthermore, it can improve cache locality by allocating memory for a region from a dedicated memory area [25].

Pre-Splitting Regions The pre-splitting of regions has major impact on the performance of bulk load operations. It can rectify the hotspot region issue and distribute the work load among all region servers. Each region has a start and an end rowkeys, and only serves a consecutive subset of the dataset. The start and end rowkeys should be defined such that all regions will have a uniform load. The pre-splitting requires a good knowledge of the rowkey structure and its value domain.

Tables T1-T4 start with an IP address, and T4-T8 have a port number in the lead position. Thus, they demand different splitting criteria. The initial splitting uses a uniform distribution function, and later it's improved by an empirical study. IPv4 space has 2^{32} addresses, and the address space is split uniformly over 15 regions, as shown in Table 2.4. Furthermore, port number is a 16 bit field with 65535 values and the same splitting strategy is applied for it, Table 2.5.

The performance gain for storing a single day of flow information is considerable. On average, 754 HBase operations are performed in a second (x30 more operations/s), the average operation latency is decreased to 27 ms (x14 faster), and the job is finished in 15 minutes (x3 sooner). Despite high efficiency improvement, there are some hotspot regions which should be investigated more.

Tables 2.4 and 2.5 show regions' start keys,the number of store files, and their sizes. It can be observed that splitting regions using the uniform key distribution function doesn't lead to a uniform load in regions.

In tables T1-T4, regions R4, R5, R10, R12 have big store files compared

Table 2.4: Initial region splits for tables T1-T4 (Store file size in MBytes-Number of store files)

Region	Starting IP address	T1	T2	T3	T4
1		30-1	0-0	0-0	5-1
2	17.17.17.17	23-1	0-0	0-0	0-0
3	34.34.34.34	32-1	6-1	5-1	0-0
4	51.51.51.51	172-1	22-1	21-1	22-1
5	68.68.68.68	325-1	57-1	57-1	57-1
6	85.85.85.85	77-1	11-1	10-1	11-1
7	102.102.102.102	85-1	9-1	13-1	0-0
8	119.119.119.119	57-1	11-1	0-0	11-1
9	136.136.136.136	102-1	11-1	10-1	11-1
10	153.153.153.153	543-1	92-1	82-1	97-1
11	170.170.170.170	21-1	0-0	0-0	0-0
12	187.187.187.187	887-1	138-1	141-1	139-1
13	204.204.204.204	73-1	11-1	10-1	11-1
14	221.221.221.221	5-1	0-0	0-0	1-1
15	238.238.238.238	0-1	0-0	0-0	0-0

to the rest of regions. Highly loaded regions serve entries within the following IP address spaces (anonymized) : *R4 → [51.51.51.51, 68.68.68.68), R5 → [68.68.68.68, 85.85.85.85), R10 → [153.153.153.153, 170.170.170.170), R12 → [187.187.187.187, 204.204.204.204)*

By investigating these IP address blocks, we identified that some of them contains Norwegian address blocks[18] and some others are popular services providers. In addition, empty regions contain special ranges such as: private networks and link-local addresses.

In tables T5-T8, regions R1, R12, R13, R14, R15 have high loads, and they serve the following port numbers: *R1 → [0, 4369), R12 → [48059, 52428), R13 → [52428, 56797), R14 → [56797, 61166), R15 → [61166, 65536),*

For tables T5-T8, R1 covers well known ports (both system ports and user ports) suggested by Internet Assigned Numbers Authority (IANA)[19],

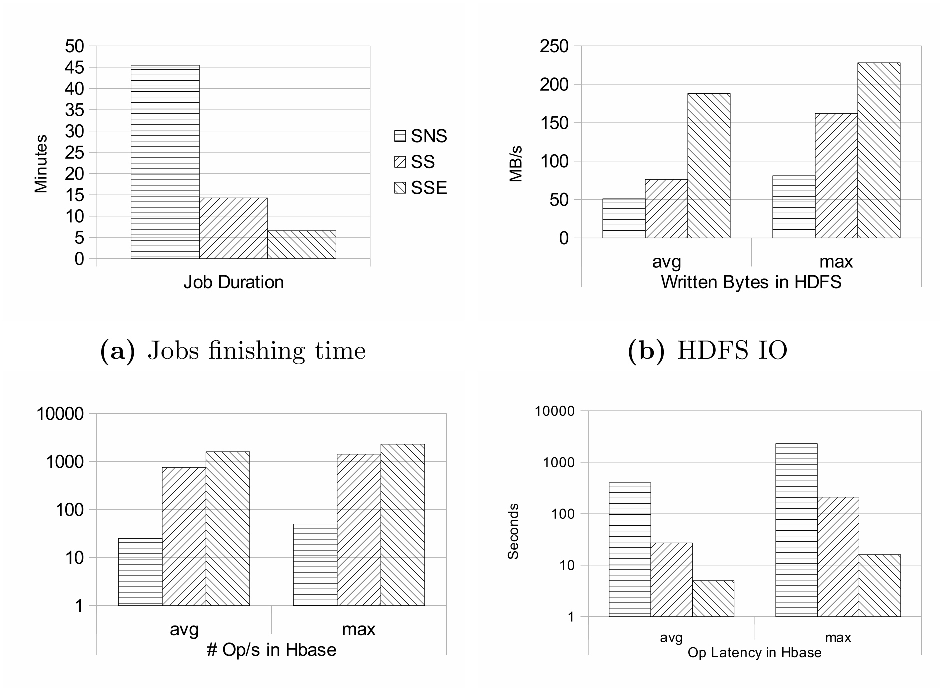
[18] `http://drift.uninett.no/nett/ip-nett/ipv4-nett.html`
[19] `http://www.iana.org/`

Table 2.5: Initial region splits for tables T5-T8 (Store file size in MBytes-Number of store files)

Region	Starting Port number	T5	T6	T7	T8
1		197-1	137-1	198-1	137-1
2	4369	7-1	0-0	0-0	0-0
3	8738	0-0	0-0	0-0	0-0
4	13107	0-0	0-0	0-0	0-0
5	17476	0-0	0-0	0-0	0-0
6	21845	0-0	9-1	8-1	0-0
7	26214	0-0	0-0	0-0	0-0
8	30583	0-0	0-0	0-0	10-1
9	34952	0-0	12-1	0-0	12-1
10	39321	0-0	13-1	10-1	12-1
11	43690	9-1	12-1	0-0	12-1
12	48059	37-1	49-1	38-1	60-1
13	52428	25-1	49-1	26-1	50-1
14	56797	25-1	37-1	25-1	38-1
15	61166	26-1	24-1	25-1	25-1

and R12-R15 contains short-lived ephemeral ports (i.e. dynamic/private ports). In the empirical splitting, the difference between system ports, user ports, and private/dynamic (ephemeral) ports will be taken into account.

A large fraction of records have port numbers of popular services (e.g. HTTP(S), SSH) or IP addresses of popular sources/destinations (e.g. Norwegian blocks, popular services). Therefore, regions should not be split using a uniform distribution over the port number range or the IP address space. The splitting is improved by taking these constrains into consideration and the result is significant. The average number of operations per second is 1600 (x64 more), the latency is 5ms (x80 less), and the job duration is reduced to 6.57 minutes (x7.5 faster). The results are depicted in Figure 2.2.



(a) Jobs finishing time **(b)** HDFS IO

(c) Operations per seconds in HBase **(d)** Operation latency in HBase

Figure 2.2: Storage performance under different implementations (SNS: Single day processing without pre-splitting, SS: Single day processing with a uniform splitting function, SSE: Single day processing with an empirical pre-splitting function)

6 Evaluation

This section analyzes several query types and their response times.

6.1 Top-N Host Pairs

Finding Top-N elements is a common query type for many datasets. In our dataset, elements can be IP addresses, host pairs, port numbers, etc. In the first evaluation, a query for finding Top-N host pairs is studied for a 150 days period. These pairs are hosts which have exchanged the most traffic on the network. The query requires processing of all records in the table T1, and aggregation of input and output bytes for all available host pairs, independent of the connection initiator and port numbers. Table T1 has 5 billion records.

Traditional tools (e.g. NFdump) are not capable of answering this query, because the long period corresponds to an extremely large dataset. For this purpose, two chaining MapReduce jobs are written for HBase tables. The first one identifies host pairs and aggregates their exchanged traffic. The second one sorts pairs based on the exchanged traffic.

On average, the first job finishes after 26 minutes, and the second one after 19 seconds. These are reasonable duration for processing a large dataset, since, Top-N host pairs queries are not executed very frequently, and there is no real-time demand for them.

6.2 Service Server Discovery for a Given Period

This query type contains time filters which means a subset of the dataset in the given time range is of interest. The query can be executed using two methods. The first method uses HBase tables to retrieve records which satisfy non-time criteria (i.e. intermediate result). Then, compliant records with the time filter are returned as the target dataset. Finally, the target dataset is processed according to the query specifications. Since, time criteria can not be evaluated in each data node[20], the time range filtering is performed in a single node. Therefore, this method is inefficient when the intermediate result is large.

The second method benefits from OpenTSDB. OpenTSDB key structure simplifies accessing data within a time range, at the cost of storage volume and response time. This method retrieves records within the time frame, first. Then, other filters are evaluated and the final processing is performed. This approach is efficient for small periods when the estimated number of compliant records with all filters is high.

Figure 2.3 depicts response times of several queries using multiple methods. The first two methods (i.e. *HBase*, and *OpenTSDB*) are explained earlier. The other methods use a traditional tool (i.e. NFdump) for retrieving and processing records. *NFD1* is executed over the complete dataset, and *NFD2* processes only a subset of the dataset which satisfies the time constrain. HBase outperforms OpenTSDB by an average factor of 87 and NFD1 by an average factor of 4472. Its performance is not comparable with NFD2, since NFD2 has a limited dataset.

[20]Because the timestamp has a trailing position in the rowkey structure.

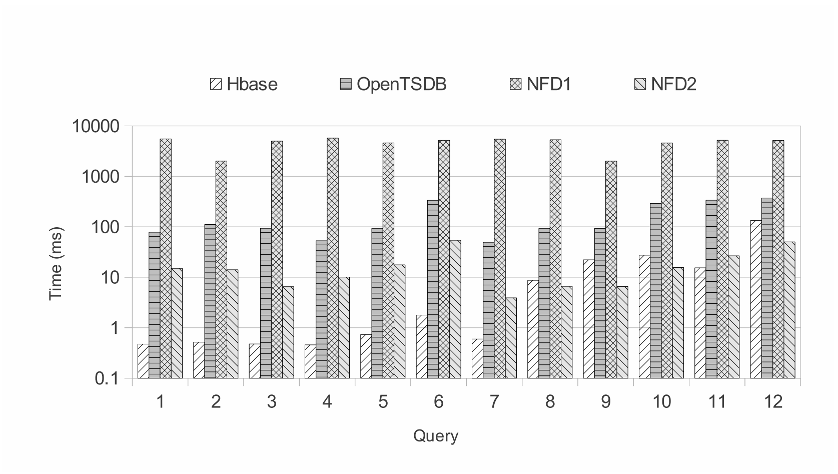


Figure 2.3: Queries performance evaluation

7 Conclusion

The paper proves the effectiveness of a data-intensive processing framework for delivering scalable and efficient network monitoring services. The proposed mechanism is not dependent on a specific network monitoring protocol, and it's applicable to any protocol as long as rowkey design criteria are satisfied. Data point structure is designed by careful analysis and conversion of monitoring record's fields. Therefore, the collection's process and storage volume overheads are reduced, and real-time data retrieval is accomplished.

Long-term queries are performed by MapReduce jobs and short-term queries are executed through available scanning APIs. These two accessing methods fulfill response time requirements of planned (e.g. statistical analysis, evidence gathering) and ad-hoc (e.g. forensics) activities.

Further Work

There are several areas which required further study and improvement such as: advanced query interface for network operators and researchers, embedded analytical engine for statistical studies, robust underlying infrastructure for enhanced availability, and integration with a cloud platform's network management service for a better real-time monitoring and security enforcement mechanisms using Software Defined Networking (SDN) technologies.

Acknowledgment

The authors would like to thank Olav Kvittem and Arne Oslebo from UNINETT and Martin Gilje Jaatun from SINTEF ICT, who provided valuable comments and assistance to the undertaking of this research.

References

[1] David G Andersen and Nick Feamster. “Challenges and opportunities in Internet data mining.” In: *Parallel Data Laboratory, Carnegie Mellon University, Research Report CMU-PDL-06-102* (2006).

[2] Apache Software Foundation. *The Apache HBase^{TM} Reference Guide*. 2012. URL: `http://hbase.apache.org/book.html`.

[3] Hari Balakrishnan et al. “Retrospective on Aurora.” In: *The VLDB Journal* 13.4 (Dec. 2004), pp. 370–383.

[4] Daniela Brauckhoff et al. “Impact of packet sampling on anomaly detection metrics.” In: *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*. IMC’06. 2006, p. 159.

[5] Eric A Brewer. “Towards robust distributed systems.” In: *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing*. Vol. 19. PODC’00. 2000, pp. 7–10.

[6] Valentín Carela-Español et al. “Analysis of the impact of sampling on NetFlow traffic classification.” In: *Computer Networks* 55.5 (Apr. 2011), pp. 1083–1099.

[7] Philip H. Carns et al. “PVFS: A Parallel File System for Linux Clusters.” In: *In Proceedings of the 4th Annual Linux Showcase and Conference*. 2000, pp. 317–327.

[8] Ronnie Chaiken et al. “SCOPE: easy and efficient parallel processing of massive data sets.” In: *Proceedings of the VLDB Endowment* 1.2 (2008), pp. 1265–1276.

[9] Fay Chang et al. “Bigtable: A Distributed Storage System for Structured Data.” In: *ACM Transactions on Computer Systems* 26.2 (June 2008), pp. 1–26.

[10] B. Claise. *Cisco Systems NetFlow Services Export Version 9*. Request for Comments 3954. IETF, Oct. 2004. URL: `http://www.ietf.org/rfc/rfc3954.txt`.

[11] B. Claise. *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information*. Request for Comments 5101. IETF, Jan. 2008. URL: `http://www.ietf.org/rfc/rfc5101.txt`.

[12] Chuck Cranor et al. "Gigascope: A Stream Database for Network Applications." In: *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*. SIGMOD'03. 2003, pp. 647–651.

[13] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters." In: *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*. OSDI'04. 2004, pp. 10–10.

[14] European Parliament Council. *Directive 2002/58/EC*. July 2002. URL: `http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:32002L0058:EN:NOT`.

[15] Jinliang Fan et al. "Prefix-preserving IP address anonymization: measurement-based security evaluation and a new cryptography-based scheme." In: *Computer Networks* 46.2 (Oct. 2004), pp. 253–272.

[16] Lars George. *HBase: the definitive guide*. O'Reilly Media, Incorporated, 2011.

[17] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. "The Google File System." In: *ACM SIGOPS Operating Systems Review* 37.5 (Dec. 2003), p. 29.

[18] Lars Hofhansl. *HBase region server memory sizing*. Jan. 2013. URL: `http://hadoop-hbase.blogspot.no/2013/01/hbase-region-server-memory-sizing.html`.

[19] Michael Isard et al. "Dryad: distributed data-parallel programs from sequential building blocks." In: *Proceedings of the 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*. EuroSys'07. 2007, p. 59.

[20] Y Jiang. *HBase Administration Cookbook*. Packt Publishing, Limited, 2012.

[21] Srikanth Kandula et al. "The Nature of Data Center Traffic: Measurements & Analysis." In: *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*. IMC'09. 2009, pp. 202–208.

[22] Ramana Rao Kompella and Cristian Estan. “The power of slicing in internet flow measurement.” In: *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*. IMC’05. 2005, pp. 9–9.

[23] Yeonhee Lee and Youngseok Lee. “Toward scalable internet traffic measurement and analysis with Hadoop.” In: *ACM SIGCOMM Computer Communication Review* 43.1 (Jan. 2012), p. 5.

[24] Bingdong Li et al. “A survey of network flow applications.” In: *Journal of Network and Computer Applications* 36.2 (Mar. 2013), pp. 567–581.

[25] Todd Lipcon. *Avoiding Full GCs in Apache HBase with MemStore-Local Allocation Buffers*. Mar. 2011. URL: `http://blog.cloudera.com/blog/2011/03/avoiding-full-gcs-in-hbase-with-memstore-local-allocation-buffers-part-3/`.

[26] Peter Phaal and Marc Lavine. *sFlow Version 5*. Tech. rep. July 2004. URL: `http://www.sflow.org/sflow_version_5.txt`.

[27] Philip Schwan. “Lustre: Building a file system for 1000-node clusters.” In: *Proceedings of the 2003 Linux Symposium*. Vol. 2003. 2003.

[28] Konstantin Shvachko et al. “The Hadoop Distributed File System.” In: *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*. MSST’10. May 2010, pp. 1–10.

Paper 3: Flexible Building Blocks for Software Defined Network Function Virtualization

Flexible Building Blocks for Software Defined Network Function Virtualization

A. TaheriMonfared[1], **C. Rong**[1]

[1] Department of Electrical Engineering and Computer Science, University of Stavanger
4036 Stavanger, Norway

Abstract:

Current virtual networks offered by IaaS cloud providers are not under complete control of their tenants. The virtual network configuration is carried out by the service provider, and the functionality is limited by the provider's offerings.
This paper presents a new approach for building and maintaining tenant-programmable virtual networks. This type of virtual networks are the basic building blocks for network function virtualization, and significantly facilitates the implementation of network functions in software. Our approach gives tenants complete control over provisioned virtual network components, and simplifies the integration with on-premises resources. The implementation confirmed the practicality and scalability of the solution, at the cost of a small overhead.

1 Introduction

The cloud computing model provides on-demand access to a shared pool of resources [16]. Depending on the service model, resource types may vary. In the Infrastructure as a Service (IaaS) model, a customer (i.e. tenant) is supposed to have the most flexibility and control over provisioned resources such as virtual machines and virtual networks.

Network virtualization is a powerful abstraction of physical network substrates, and flexibility is an important aspect of it [1]. However, this has not been achieved in today's cloud services. Virtual networks are not under customers' control, and have a very limited flexibility. For instance, a customer can define a basic IP addressing scheme, or access control lists (ACLs) at most. Service providers are responsible for creating and maintaining virtual networks, in collaboration with infrastructure providers. In a public deployment, network functionalities are restricted to those exposed by the provider. While, in a private one, utilizing available functionalities requires tremendous cooperation of involved entities (e.g. operation center, internal tenants).

There has been some efforts among service providers for delivering reliable and efficient network services (including network virtualization), using Software Defined Networking (SDN) mechanisms. However, mechanisms are not exposed to customers, and they may just experience better guarantees and availability. There are several obstacles in exposing the functionality, including clashing customers' configurations (e.g. overlapping RFC 1918 [19] addresses) and fine-grained access control on APIs.

This paper proposes a new approach for the network virtualization, which is similar to the full machine virtualization technique, in terms of accessibility and isolation. In this approach, the cloud provider takes advantage of SDN mechanisms for creating virtual networks, and tenants utilize the same mechanisms to take over the control of them.

Each virtual network has a dedicated and isolated set of networking elements, that are directly accessible and fully controllable by the tenant. They have negligible performance overhead on other tenants and the infrastructure, while provide sufficient management access to the provisioned networking resources for tenants. For instance, tenants can create tunnels, tag interfaces, and utilize different forwarding protocols in their virtual networks, or seamlessly integrate them with their on-premises resources. Programmable virtual networks are the basic building blocks for delivering network-aware services or building specific-purpose overlays.

Section 2 discusses the background for cloud computing, and the new paradigms in networking. Section 3 explains our approach, its requirements, advantages, and disadvantages. Section 4 studies scalability and performance of the solution. The related work to programmable virtual networks are briefly mentioned in Section 5. The final remarks are pointed in Section 6.

2 Background

2.1 Virtual Networks in Infrastructure as a Service (IaaS) Model

A tenant is a customer of IaaS services and it can have multiple virtual networks. A virtual network may have multiple subnets, which are distinguished from each other using their Classless Inter-Domain Routing (CIDR) blocks. Overlapping CIDR blocks for different networks' subnets may or may not be supported, since it depends on the employed network virtualization technology.

In OpenStack, VMs are attached to an integration bridge, and communicate with other entities through a tunnel bridge. The integration bridge works as a virtual Top of Rack switch, and tags VMs traffic according to their corresponding tenants.

2.2 Software Defined Networking and OpenFlow

SDN facilitates new methods for managing and configuring networks. The key process in SDN is the abstractions between different layers of networking mechanisms. Shenker et al. [20, 12] introduces three level of abstractions in the network, *distributed state abstraction*, *specification abstraction*, and *forwarding abstraction*. The distribution abstraction provides a global view of the network, and hides the details of distributed states from the higher level mechanisms. The specification abstraction builds a simple model of the network, and makes the abstract configuration decoupled from the physical infrastructure. The forwarding abstraction make the forwarding plane more flexible, by shielding the hardware details from the control plane. OpenFlow [15] is one approach for forwarding abstraction, which separates the control plane from the forwarding plane physically.

An OpenFlow switch has a set of flow tables and a group table. An OpenFlow controller can add, update, and delete flow entries in a flow table of the switch. Each flow entry has a matching pattern, a set of ordered actions, a priority, and counters. When a packet enters a port, and the highest-priority matching flow entry in the first table is found, corresponding actions are applied in order; Then the match data and action set are sent to the next table. An action can be forwarding the packet to one or more port(s), dropping, or modifying it [17].

2.3 Network Function Virtualization

European Telecommunications Standards Institute[21] defines Network Function Virtualization (NFV) [7] as a network architecture which utilizes virtualization for delivering network functions. Functions are realized in software that can be deployed, migrated, and replicated on standard hardware. The software is decoupled from proprietary hardware, and can evolve beyond hardware appliances' lifecycles.

3 Tenant Controlled Networks

In the IaaS model of cloud services, tenants don't have access to network devices, which connect their provisioned resources. They may only have a limited knowledge about the resource distribution, such as availability zones and regions. For instance, the underlying network topology, architecture, and other characteristics are not exposed to tenants.

Moreover, networking mechanisms (e.g. routing protocols) can not be modified and tenants are bound to decisions made by providers. A new network function, which is not supported by the provider, can only be realized in virtual machines. This has several drawbacks such as significant performance overhead, possible single point of failure, and placement challenges (e.g. for middlebox functions).

Providers can not give access to tenants for controlling their virtual networks, since there is no reliable method for segregating network configurations and enforcing policies. In addition, a tenant configuration may have significant impact on other tenants' networks, as well as the provider infrastructure.

In our proposed mechanism, a tenant manages a dedicated set of virtual network components using Open vSwitch DataBase (OVSDB) protocol [18], and programs the data planes through OpenFlow protocol [17], from a logically centralized SDN controller. The tenant's controller is directly

[21]http://www.etsi.org

connected to virtual switches, which means there is no additional cost for passing events and instructions through another software layer (e.g. infrastructure's SDN controller). Tenants' controllers are decoupled from the infrastructure provider's controller, so they can fail independently. Thus, a failure in the infrastructure's controller won't affect tenants' controllers – assuming there are fail-over forwarding mechanisms; And a failed tenant's controller won't impose overhead to the infrastructure's controller (e.g. polling costs, liveness checking).

The tenant's controller maintains a unified view of the virtual network, and it has insights into the topology, architecture, and resource distribution over the physical infrastructure. The information helps the tenant to implement new network functions efficiently, and deploy services properly.

It should be noted that this approach does not necessarily need tenants intervention. Those tenants who are not willing or do not have the competence, to control their virtual networks, can delegate the task to the provider or any other third party.

The solution is implemented on top of OpenDaylight [14] controller and its OVSDB plugin, with 2762 lines of code, which is available online[22].

3.1 Components

In addition to the common integration and tunnel bridges, each compute node has a pair of dedicated integration and tunnel bridges for each tenant's network. These tenant network bridges are created only on compute nodes which are hosting VMs from that tenant. Integration and tunnel bridges for tenant *xy* are denoted by *brint-xy* and *brtun-xy*. Virtual bridges in a compute node are represented by records in the Open vSwitch (OVS) Bridge table and are part of the same OVS instance.

Moreover, there should be a dedicated transport network for a tenant. This network type can be maintained by OpenStack Neutron or the infrastructure provider. For the sake of simplicity in this study, the latter is the case here.

A tenant transport network connects tenant's tunnel bridges together through endpoint interfaces (Figure 2.2). The interface (denoted by *ex-xy* for tenant *xy*) type is *Internal*, and it works as a connection between the bridge and the host kernel's TCP/IP stack. The tunnel endpoint interfaces and an interface (e.g. *eth1*) on the physical transport network are attached to the common tunnel bridge (Figure 2.1).

[22]https://github.com/aryantaheri/ovsdb

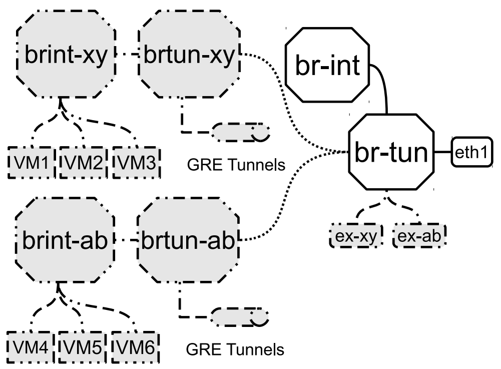


Figure 2.1: Dedicated bridges for two tenants in a compute node.

Equations (2.1), and (2.2) show the number of bridges in a compute node and the total number of them in the infrastructure. As it's shown, the number of bridges in a compute node increases linearly with the number of hosted tenants. Although, the number of bridges increases considerably, the overhead is not significant (Section 4).

$$|bridges_c| = 2 \times (|tns_c| + 1) \tag{2.1}$$

$$|bridges| = (\sum_{c \in cns} 2 \times (|tns_c| + 1)) + |ons|(2 \times |tns| + 3) \tag{2.2}$$

where:

tns = tenant networks set with a running VM

tns_c = tns on a compute node c

cns = compute node set

ons = OpenStack network server set

When a new VM is scheduled on a compute node, the presence of tenant bridges is verified. If they didn't exist, the tenant bridge pair and the tunnel endpoint interface are created. Then, the VM's TAP device (i.e. virtual link layer device) is attached to the tenant integration bridge and appropriate forwarding rules are pushed.

3.2 Connectivity

VMs of a tenant in a node are connected to the tenant's integration bridge. This bridge works as a virtual ToR switch and provides intra-connectivity.

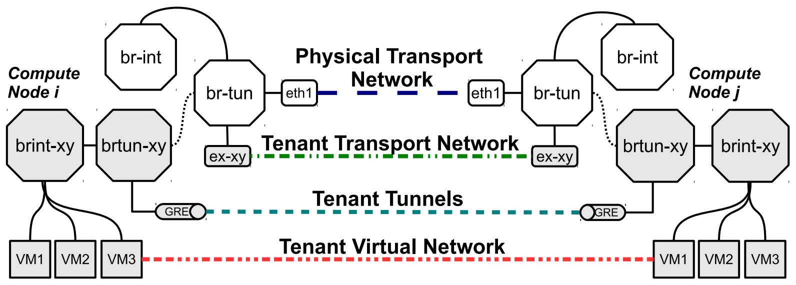


Figure 2.2: Logical and physical networks.

However, VMs across compute nodes communicate through tenant's tunnel bridges, and this bridge type handles inter-connectivity. Integration and tunnel bridges of a tenant are connected using a pair of patch ports. Figure 2.1 depicts the networking components in a single compute node when two tenants are hosted.

3.3 Tunnels

When a tunnel endpoint interface is added, a *udev*[13] rule is triggered, and the interface IP address is pushed to the OVS Interface table. Upon the table modification, the SDN controller is notified, and establishes Generic Routing Encapsulation (GRE) [8] tunnels to discovered endpoints. A tenant's network has its individual tunnel set ($tnts_n$). Tunnels are only established between compute nodes which are hosting VMs attached to that network. Equations (2.3) and (2.4) show the upper bound of number of tunnels for a network, and total number of created tunnels in the platform.

$$|tnts_n| \leq \binom{|cns| + |ons|}{2} \tag{2.3}$$

$$|tnts| \leq |tns| \binom{|cns| + |ons|}{2} \tag{2.4}$$

where:

$tnts \;\; = $ tenant network tunnel set
$tnts_n = tnts$ for network n

3.4 Flow Programming

When there is no matching flow entry for a packet, the switch sends a Packet-In event to the controller to retrieve further instructions. The event may contain the full packet or some parts of the header. The processing time at the controller and the round trip time, for the event and its response, is costly. Thus, proactive flow programming of switches is the preferred method for an efficient forwarding. Therefore, flow rules are calculated in a deterministic way and pushed to the switches, when a new entity is added to a virtual network.

Once a VM's port is attached and tunnels are created, the controller pushes four types of flow rules (Table 2.1) to relevant bridges. Whenever a packet traverses a tunnel, it's marked with the corresponding tenant network's segmentation ID (i.e. GRE tunnel key). The tenant tunnel bridge in the host of the new VM is called *local*, and tunnel bridges in other tenant hosts are called *remote*.

Rule types are local ingress, local egress, local flood, and remote egress.

(i) The local ingress rule matches the traffic from tunnels on the tenant network's segmentation ID (i.e. GRE tunnel key), then tag it with the tenant's internal VID and forward it to the local integration bridge.

(ii) The local egress rule identifies other VMs' MAC addresses on the same virtual network at the remote sides, and choose designated tunnels for reaching them. The matching traffic's VID is removed and the traffic is forwarded through designated tunnels.

(iii) The local flood rule matches broadcast or unknown unicast traffic, and sends them out through all tenant's tunnels.

(iv) The remote egress rule is applied to the remote bridges. It matches traffic from remote tenant integration bridge on the destination MAC address, and the tenant's internal VID. Then the VID is popped and the traffic is forwarded through the designated tunnel.

The controller creates *O(N)* flow entries in each compute node, where N is the total number of instances which may belong to several tenant networks. Each tenant network tunnel bridge has $O(N_t)$ entries, where N_t is the number of instances on that network.

Rule	Flow	Match	Actions
Local Ingress	r-tun→l-tun→l-int	tunnel port, GRE key	push VID, fw to patch-port
Local Egress	l-int→l-tun→r-tun	patch port, VID, Dst MAC	pop VID, mark GRE, fw to tunnel-port
Local Flood	l-int→l-tun→r-tun	patch port, VID	pop VID, mark GRE, fw to tunnel-port
Remote Egress	r-int→r-tun→l-tun	patch port, VID, Dst MAC	pop VID, mark GRE, fw to tunnel-port

Table 2.1: Flow Rules

3.5 Packet Flow

Figure 2.3 depicts the packet flow between two VMs which are attached to the same tenant network, and hosted on two compute nodes. Traffic from VM3 on node i is tagged in *brint-xy* with the tenant network's VLAN ID on node i. If the destination VM is not hosted on the same node, *brint-xy* sends out the traffic to *brtun-xy*, otherwise it will be forwarded locally. When the packet matches relevant flow entries on *brtun-xy*, it is sent out through the GRE tunnel. Since, tunnel endpoints are on the same subnet as *ex-xy* interface, node i TCP/IP stack routes the packet using *ex-xy* interface. At this stage, the common tunnel bridge (*br-tun*) forwards the packet to the physical interface *eth1* on the transport network. Finally, the packet is routed through the physical transport network and reaches *eth1* interface on node j, and the reverse process happens there.

3.6 Trade-offs

The approach has five major benefits for an enterprise tenant. First, the tenant directly accesses the control and management planes using OpenFlow and OVSDB protocols. Second, each virtual network has a dedicated set of virtual components (e.g. virtual switches, interfaces, etc.). These two benefits facilitate the implementation of virtual network functions for a tenant. For instance, one can enforce the required middlebox

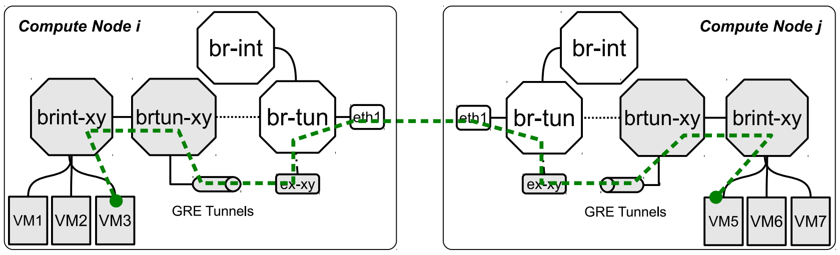


Figure 2.3: Flows of packets between two VMs of a tenant across two compute nodes

functions to the provisioned virtual network, without involvement of the service provider (i.e. lock-in), or procurement of new proprietary appliances (i.e. additional costs).

Third, tenants' tunnel bridges are connected by GRE tunnels, which facilitate layer 2 isolation of inter-compute VMs' communications. Forth, SDN mechanisms can be used for a unified management of on-premises and off-premises resources. Fifth, virtual network topology and architecture are decoupled from the physical infrastructure, which makes possible transparent modification of the infrastructure.

This approach is not a panacea, and there is a trade-off between flexibility and performance. By introducing dedicated components, the start-up time increases and the implementation becomes more complex.

3.7 Performance Tuning

Initial results were not satisfactory. Thus, the following tuning were applied to improve the performance:

- MTU size on VMs: The GRE overhead causes the packets to be fragmented, and reduces the performance significantly. By decreasing the MTU size on VMs, GRE packets won't be fragmented.

- Vhost_net: It reduces the number of system calls and avoids context switching, by moving packets between the guest and the host system using the kernel instead of QEMU.

- Hardware Offloading: Where possible, the offloading functionality of the Linux kernel is used to improve the performance and reduce the CPU load. Tuned parameters are RX and TX check-summing, Generic Segmentation Offload (GSO), and Generic Receive Offload (GRO).

- Kernel's IP neighbor table: The table size is increased to avoid overflow.

4 Evaluation

The proposed approach must scale in a large infrastructure, with many virtual machines and virtual networks. Thus, several metrics have been measured including reachability time and available bandwidth. The experiment is carried out for a different number of instances and networks, which yields to a variety of instance distribution over compute nodes, and virtual networks. Each experiment is applied to two scenarios: first, tenants don't have full control (i.e. instances are attached to the common bridge), and second, tenants have full control (i.e. tenant instances are attached to dedicated bridges). These scenarios are denoted by CNB (Common Network Bridges) and DNB (Dedicated Network Bridges), respectively.

The virtual machine image used for creating instances is a customized version of CirrOS[5] based on Buildroot[4]. The VM type is m1.tiny with one VCPU, and 512 MB memory. The monitoring aggressiveness of experiments is decreased to moderate its negative impact on the overall performance.

The testbed has five compute nodes, one cloud controller node, one SDN controller, and one monitoring node. Each node has an AMD Opteron Processor 4180, 6 cores, 32 GB memory, and two Gigabit network interfaces. Management and data (VM inter-connection) networks are separated and use dedicated interfaces, which are connected to an HP 5406zl switch. All cloud nodes are running Ubuntu LTS 12.04, OpenStack Havana, and Open vSwitch 2.0.0.

4.1 Reachability Time

The first experiment measures the reachability time of an instance. The period between an instance spawn up request time (t_{rq}) and the first ICMP echo reply received time (t_{ier}) is called instance reachability time (t_r). The boot requests are sent from the platform controller node to the management API in one batch. And the ICMP echo requests are sent from the network controller node (i.e. an interface in the tenant network's namespace) toward the allocated IP addresses for each instance. This is an effective metric for the analysis of a variety of networking mechanisms in a virtualization platform. Since, the total overhead of not-networking

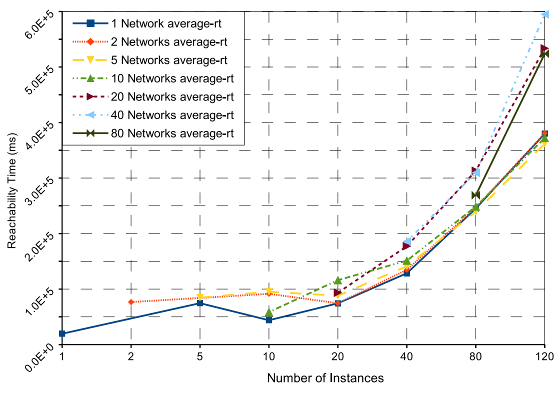


Figure 2.4: Average reachability time for DNB

components of the platform is uniformly reflected in this metric, and it can be masked among different mechanisms.

Reachability experiment results have eight fields, number of instances, number of networks, total number of records, success records, failed records, missing records, average reachability time and the reachability time standard deviation.

Figure 2.4 depicts the average reachability time, when each tenant's network has a dedicated set of bridges and tunnels (i.e. DNB). The Y axis is the reachability time in milliseconds, and X axis is the number of instances. Each data series represents an experiment which has a constant number of networks. An experiment is divided into sub-experiments with a fixed number of requested instances. Sub-experiment's results are data points of the data series, and show the average reachability time for the given number of instances when they're distributed over the experiment's networks.

As depicted in Figure 2.5, DNB does not perform as well as CNB in this test. It takes longer for a VM to become reachable, when it is attached to a dedicated bridge, and utilizes dedicated tunnels. Although there is an extra start up cost in DNB, this overhead is much less significant when a large number of instances is requested (e.g. 80 instances). Particularly, when the instance distribution is uniform, the overhead is negligible for the last $n - |cns|$ instances, where n is the total number of instances. In DNB, approximately the first $|cns|$ instances faces the overhead of bridge and tunnel creation, and the remaining instances will use existing tenant network bridges.

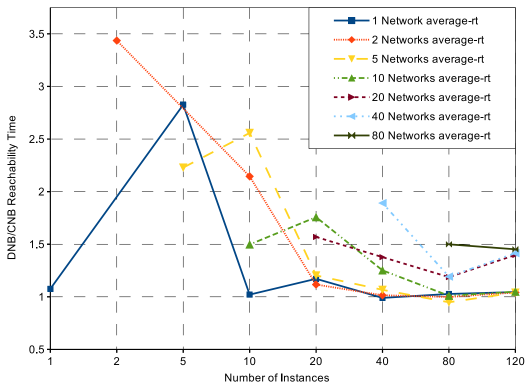


Figure 2.5: Average reachability time comparison (DNB/CNB)

4.2 Throughput

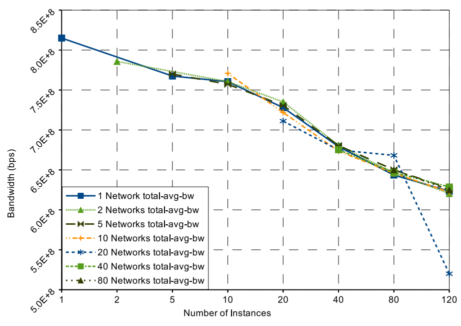
The available bandwidth for VMs in a cloud environment is a critical metric for the performance analysis. The experiment evaluates the available TCP and UDP bandwidth between the network controller and VMs, in both directions individually. Iperf is used for measuring TCP and UDP bandwidth, and for each direction the transmission period is 20 seconds.

The TCP experiment reports average and standard deviation of the available bandwidth for both directions and the combined one. The UDP experiment reports average and standard deviation of bandwidth, jitter, transferred datagrams, and out-of-order datagrams.

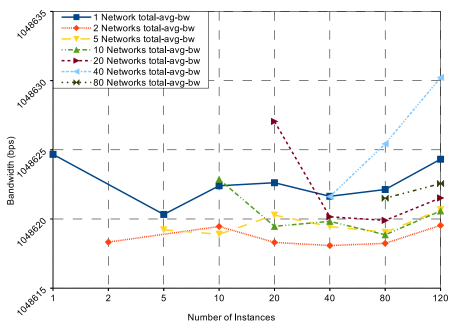
Figure 2.6 shows the average available bandwidth for TCP (2.6a) and UDP (2.6b), when dedicated bridges are utilized. The UDP throughput is consistent, and fluctuates around 1 Mbps for a different number of networks and instances. However, the TCP bandwidth decreases when the number of instances increases, and doesn't change significantly when the number of networks increases for the same number of instances.

Moreover, the TCP bandwidth in each direction for DNB is presented in Figure 2.7. It shows that the average throughput for the VM to controller direction (Figure 2.7b) is higher than the opposite one (Figure 2.7a). This is due to the VM's type (i.e. m1.tiny) and its processing power. Generally, RX operations are more demanding and CPU intensive; Therefore, the processing power can become a bottleneck for the ingress traffic toward a VM.

The same set of experiments are performed for CNB, and results are compared with DNB's experiments. UDP performance for DNB improved



(a) Average bidirectional TCP bandwidth



(b) Average bidirectional UDP bandwidth

Figure 2.6: Average bidirectional TCP and UDP bandwidth for DNB

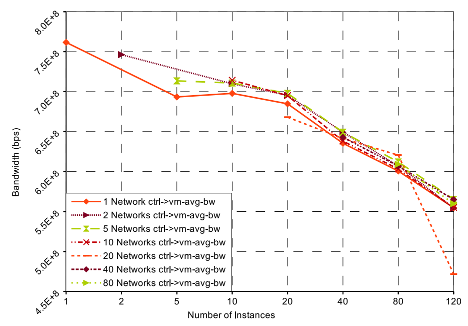
slightly, while its TCP performance degraded about 8% on average (Figure 2.8). Although, the performance overhead of DNB for TCP is not negligible, the delivered functionality is considerable.

5 Related Work

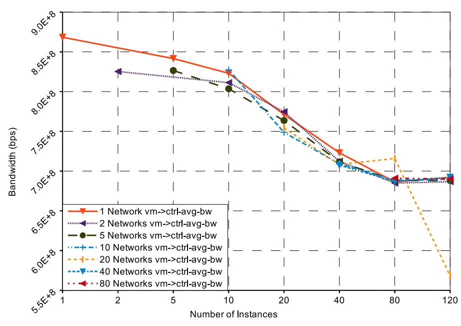
Oktopus [2] is a system where a tenant expresses network requirements and gets a predictable environment in shared setting, while provider's revenue is not significantly affected. FlowN [6] uses VLAN tagging for creating virtual networks, and isolating address spaces. A virtual network topology is completely decoupled from the physical topology and remains unchanged even after changes in the provisioned resource distribution. Tenants of FlowN share a common controller, similar to container-based virtualization, and their VLAN IDs along switch mapping information are stored in a relational database (i.e. MySQL).

Keller et al. [10] propose an approach for the abstraction of a single router for each customer, and argue that it leads to a better control of the virtual network and reliable service delivery. CloudNet [22] and SEC2 [9] are architectures which utilize VPNs to seamlessly and securely connects provisioned cloud resources to on-premise resources. They isolate customers' virtual networks on the shared infrastructure using VLAN tagging.

CloudNaaS [3] delivers isolated virtual networks, with QoS, middlebox functions, and flexible address space. OpenFlow and VLAN tagging are used for programming flow rules on switches, and isolating traffics,



(a) Average TCP bandwidth from Controller to VMs



(b) Average TCP bandwidth from VMs to Controller

Figure 2.7: TCP bandwidth breakdown for DNB

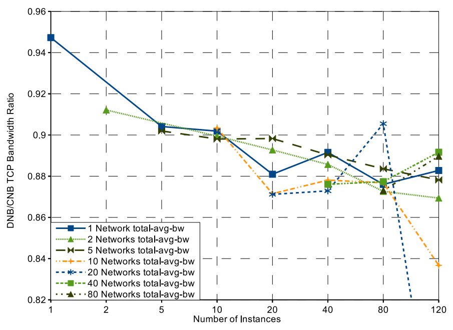


Figure 2.8: Bidirectional TCP bandwidth comparison (DNB/CNB)

respectively. These features are delivered by the service provider, and a customer doesn't have direct control of their resources.

Slicing the network (e.g. FlowVisor [21]) is another way of providing access to a part of the network, and giving complete access to tenant over that portion. However, network slices are coupled with the physical network topology, and a tenant can see a subset of physical topology. Moreover, FlowVisor [21] doesn't support cloud platforms, and is not compatible with the recent versions of the OpenFlow protocol.

NVP [11] is a network virtualization platform for multi-tenant datacenters. It's based on a familiar abstraction, so tenants can apply their enterprise network policies to provisioned virtual networks, without modifications. Each flow entry's match and action fields are updated with the packet's metadata, to isolate logical datapaths. NVP uses a declarative language for expressing the logic, which makes the forwarding state computation decoupled from state transitions and event ordering. The language is only available for NVP developers, and can not be used by tenants. Software switching (realized by OVS) is also an important aspect of NVP, that facilitates fast innovation and design flexibility.

Our proposed approach differs from NVP in two aspects; First, NVP uses extra metadata in flow entries to isolate tenants while DNB creates a dedicated set of entities for each tenant. Although OVS uses a single underlying datapath across tenants' bridges in a compute node, they are isolated from a tenant's perspective. Second, in our approach, tenants control their virtual networks directly, and are not dependent on a proprietary solution. Thus, they can avoid vendor lock-in.

6 Conclusion

This paper presented a new type of virtual network, with dedicated components for each tenant. Enterprise tenants can directly control their virtual networks, enforce policies, apply configurations, and engineer traffic, much like what they can do with on-premises resources. Switches in the networks support OpenFlow and OVSDB protocols, and any controller with compatible southbound APIs can configure them. The solution is implemented as a bundle for OpenDaylight controller, and experiments are performed in an OpenStack deployment. Our experience shows that the approach is not only practical, efficient, and scalable, but also has significant benefits for tenants. The performance overhead is not significant, and the deployment cost is moderated for a service provider.

Acknowledgement

The authors would like to thank Madhu Venugopal and Brent Salisbury for their insightful comments. This work is done in collaboration with Norwegian NREN UNINETT.

References

[1] T. Anderson et al. "Overcoming the Internet impasse through virtualization." In: *Computer* 38.4 (Apr. 2005), pp. 34–41.

[2] Hitesh Ballani et al. “Towards predictable datacenter networks.” In: *ACM SIGCOMM Computer Communication Review* 41.4 (Oct. 2011), p. 242.

[3] Theophilus Benson et al. “CloudNaaS: a cloud networking platform for enterprise applications.” In: *Proceedings of the 2Nd ACM Symposium on Cloud Computing*. SOCC’11. 2011, pp. 1–13.

[4] *Buildroot*. URL: `http://buildroot.uclibc.org/`.

[5] *CirrOS*. URL: `https://launchpad.net/cirros`.

[6] Dmitry Drutskoy, Eric Keller, and Jennifer Rexford. “Scalable Network Virtualization in Software-Defined Networks.” In: *IEEE Internet Computing* 17.2 (2013), pp. 20–27.

[7] ETSI. “Network Functions Virtualisation.” In: Oct. 2012. URL: `http://portal.etsi.org/NFV/NFV_White_Paper.pdf`.

[8] D. Farinacci et al. *Generic Routing Encapsulation (GRE)*. Request for Comments 2784. IETF, Mar. 2000. URL: `http://www.ietf.org/rfc/rfc2784.txt`.

[9] Fang Hao et al. “Secure Cloud Computing with a Virtualized Network Infrastructure.” In: *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*. HotCloud’10. 2010, pp. 16–16.

[10] Eric Keller and Jennifer Rexford. “The ”Platform As a Service” Model for Networking.” In: *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking*. INM/WREN’10. 2010, pp. 4–4.

[11] Teemu Koponen et al. “Network Virtualization in Multi-tenant Datacenters.” In: *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*. NSDI’14. 2014, pp. 203–216.

[12] Teemu Koponen et al. “Onix: A Distributed Control Platform for Large-scale Production Networks.” In: *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*. OSDI’10. 2010, pp. 1–6.

[13] Greg Kroah-Hartman. “udev – A Userspace Implementation of devfs.” In: *Proc. Linux Symposium*. 2003, pp. 263–271.

[14] Linux Foundation. *OpenDaylight*. URL: `http://www.opendaylight.org/`.

[15] Nick McKeown et al. “OpenFlow: enabling innovation in campus networks.” In: *ACM SIGCOMM Computer Communication Review* 38.2 (Mar. 2008), p. 69.

[16] Peter Mell and Timothy Grance. *The NIST Definition of Cloud Computing.* Tech. rep. SP 800-145. National Institute of Standards and Technology, Information Technology Laboratory, Sept. 2011.

[17] Open Networking Foundation. *OpenFlow Switch Specification.* Apr. 2013.

[18] Ben Pfaff and Bruce Davie. *The Open vSwitch Database Management Protocol.* Request for Comments 7047. IETF, Dec. 2013. URL: `http://www.ietf.org/rfc/rfc7047.txt`.

[19] Y. Rekhter et al. *Address Allocation for Private Internets.* Request for Comments 1918. IETF, Feb. 1996. URL: `http://www.ietf.org/rfc/rfc1918.txt`.

[20] Scott Shenker et al. “The future of networking, and the past of protocols.” In: *Open Networking Summit* (2011). URL: `http://www.opennetsummit.org/archives/apr12/site/talks/shenker-tue.pdf`.

[21] R. Sherwood et al. “Flowvisor: A network virtualization layer.” In: *OpenFlow Switch Consortium, Tech. Rep* (2009).

[22] Timothy Wood et al. “The Case for Enterprise-ready Virtual Private Clouds.” In: *Proceedings of the 2009 Conference on Hot Topics in Cloud Computing.* HotCloud’09. 2009.

Paper 4: Virtual Network Flavors: Differentiated Traffic Forwarding for Cloud Tenants

Virtual Network Flavors: Differentiated Traffic Forwarding for Cloud Tenants

A. TaheriMonfared[1], **C. Rong**[1]

[1] Department of Electrical Engineering and Computer Science, University of Stavanger
4036 Stavanger, Norway

Abstract:
Today, a cloud system user can select a specific type of virtual machine for deployment based on needs, for instance memory or storage size. In terms of networking, however, no similar mechanism exists which allows users to select a virtual network based on characteristics such as link speed and QoS. The lack of such a mechanism makes it difficult for users to manage VM instances along their associated networks. This limits the efficacy and scalability of cloud computing suppliers.
This paper presents a novel approach for defining *virtual network flavors* and differentiated forwarding of traffic across the underlay networks. The flavors enable tenants to select network properties including maximum line rate, maximum number of hops between two virtual machines, and priority. Measures such as metering, prioritizing, and shaping facilitate steering traffic through a set of paths to satisfy tenants' requirements. These measures are designed such that the legacy parts of the underlay network can also benefit from them. Software Defined Networking (SDN) mechanisms are an essential part of the solution, where the underlay and overlay networks are managed by a network operating system. The implementation and evaluation data are available for further development[5].

1 Introduction

Today's cloud infrastructure as a service provider supports a variety of virtual machine types. These types are frequently referred to as *flavors*, which defines a virtual machine's specifications, such as the number of vCPU, memory, and storage size. Flavors aid users by simplifying the selection and specification of VMs. However, there exists no similar mechanism for the virtual networks connecting these VMs. Virtual networks are mostly created with similar specifications, which are also limited. They lack quality of service (QoS) support, path calculation options, etc. Per-tenant virtual network building blocks, as presented in [14], propose an extreme approach where the new architecture delegates all network functions to tenants. While this solution can be beneficial for an enterprise customer, it is a burden for the average customer. Moreover, a tenant is not in control of the underlay network and cannot influence the forwarding decisions. Thus, a provider should deliver more functionality for virtual networks.

Tunneling is a common technique for creating virtual networks and isolating tenants' traffic in the underlay. However, this technique gives rise to several issues in adapting legacy network functions to handle tenants' traffic. In particular, identifying, classifying, and engineering tenants' traffic in the core network, without decapsulating, are challenging tasks.

Moreover, typical data center network topologies consist of a multi-rooted tree [6]. Therefore, there are multiple paths between any two nodes. Virtual network traffic between two endpoints of a tenant should be forwarded through the path which fulfills the tenant's QoS requirements. A central controller with a unified view of the network can perform significantly better than hash-based Equal-Cost Multi-Path (ECMP) load-balancing [6].

This paper proposes an approach for differentiated forwarding of overlay traffic across the underlay network (Figure 2.1), such that it satisfies the virtual network flavor constraints. Initially, the virtual network flavor should be defined as a structure that presents a tenant traffic class, maximum rate, priority, etc (§3). Then, the following steps are necessary to enforce the flavor requirements. The first step is to discover the underlay network topology. The underlay topology is later used to find a set of flavor-compliant paths between VMs which are connected to the same virtual network (§4). Therefore, the tenant's points of presence in the data center should be identified and the topology of its virtual network should be discovered (§5). Then, tenants' traffic are classified according

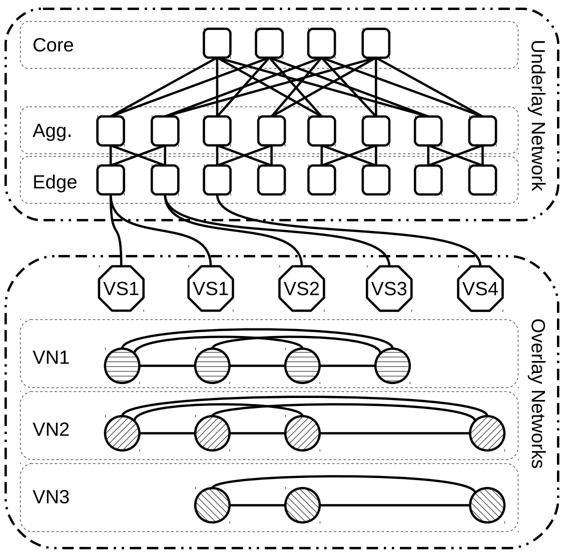


Figure 2.1: High Level Overlay and Underlay Architecture

to their network flavors (§6). The classification is performed only once, at the endpoint virtual switch (§7). The overlay traffic is marked such that a legacy network node can also benefit from the classification. Finally, the aforementioned end-to-end paths, for each class, are programmed in the overlay and underlay networks (§8). In addition, an extensive evaluation framework is developed to study the effectiveness and accuracy of the flavor implementation (§9).

2 Related Work

Plug-n-Serve [7] is a load-balancing system for web services. It uses the network state and servers load to customise flow routing and update service capacity (e.g. number of hosts).

Wang et al. [15] proposes a proactive load-balancing approach for online services using OpenFlow. It creates a minimal set of wildcard rules to aggregate clients' requests and update the set to reflect the system dynamics.

Hedera [6] focuses on scheduling large flows in a data center with a topology inspired by the fat tree topology. It shows that dynamic flow scheduling can substantially improve network bandwidth in multi-rooted trees, while the deployment cost is moderated. Simulated Annealing and

Global First Fit algorithms are compared to ECMP, and the results show that the former outperforms the rest in most cases.

Khan et al. [8] introduces a mechanism for virtual network QoS support in the Multi-Protocol Label Switching (MPLS) transport domain. OpenFlow domains are the ingress and admission points to the MPLS transport network, and have their own disjoint controllers. However, the transport network is configured by legacy means. Each virtual network session admission, at an OpenFlow domain, requires communication with the domain's controller. This requirement imposes an overhead on the session establishment process, which is caused by the communication round-trip time and the decision-making logic in the controller.

3 Virtual Network Flavors

Customers' traffic in a cloud infrastructure can be isolated using different techniques, such as IP block splitting, VLAN tagging, and tunneling. Maintaining IP blocks is a tedious task – the VLAN field has a limited size (up to 4094 under IEEE 802.1Q), and tunnels introduce additional overheads, in terms of forwarding capacity and traffic volume. However, tunneling is the most flexible approach and can be widely used if the overheads are reduced by employing hardware acceleration (e.g. Virtual eXtensible Local Area Network (VXLAN) [9]) or hardware-friendly protocols (e.g. Stateless Transport Tunneling (STT)[4]).

This paper focuses on virtual networks which use tunneling techniques for traffic isolation and end-to-end forwarding. However, the mechanisms explained here can be expanded, with minor extensions, to cover other network virtualization techniques.

Virtual network flavor specifies the quality of end-to-end paths between any two VMs which are connected to a virtual network. Although our proposed mechanisms support varying flavors for the paths in a virtual network (i.e. virtual network path flavor), for the sake of simplicity the case is presented where paths in a virtual network use the same flavor (i.e. virtual network flavor). Path flavor plays an essential role in providing fine-grained QoS for provisioned resources by a tenant. For instance, resources involved in latency-sensitive tasks utilize the shortest path, while high traffic volume operations are forwarded through the least utilized path.

Each flavor has a class that is used for coarse-grained traffic classification and facilitates the legacy network support. Since legacy networks have a limited capacity for traffic classes (e.g. 6-bit Differentiated Services Code Point (DSCP)), multiple flavors may have the same class. In addition to the class, a flavor determines the end-to-end priority and maximum rate of virtual network traffic in the underlay network. In the following text, the underlay refers to the infrastructure's network that is controlled by the infrastructure provider; while overlay refers to the tenant's virtual network that is established on top of the infrastructure network, Figure 2.1.

As the virtual switch inside a compute node has high throughput, the inter-VM communication inside a single hypervisor is not shaped. Therefore, traffic engineering takes place for the inter-hypervisor VM's communication.

4 Underlay Topology Discovery and Path Calculation

The data center network has a tree-like topology with redundant paths between any two nodes. Therefore, it can be represented as a weighted directed sparse graph, where an edge has a REFERENCE_SPEED/LINK_SPEED weight. The underlay topology is constructed by processing node and link updates, which are sent from the controller platform.

For a different set of tenants with varying network demands, shortest path routing is not flexible enough [1]. The aim is to calculate the first k-shortest loopless paths (KSP) between any given node pair, where k is a positive integer. A modified version of Yen's algorithm [16] is implemented for path calculation. Moreover, a set of constraints can be used to further limit the result. For instance, the constraints can ensure that packets visit a set of nodes before reaching the destination to apply middle-box functions, or avoid another set to bypass congestion. Modularity of the approach makes it flexible to update the algorithm with another group of policies.

5 Overlay Topology Discovery

Tunnels are established on demand between interested endpoints. Tunnel endpoints are responsible for marking tenants' traffic with their tunnel keys, prior to encapsulation. Two types of tunnel discovery are performed – proactive and reactive. The proactive approach is suitable for Open-

Flow 1.0 and 1.3 protocols. It listens to the management plane events which are dispatched from the OVSDB southbound plug-in. When a new tunnel interface is identified, the key, IP address, interface ID, and the corresponding tenant ID are added to a data store. The data store is further processed to build a matrix of tunnel endpoints for each tenant, that represent the tenant's virtual network. The data store reflects the current status of tenants' overlays, and is updated by relevant events (e.g. add, remove, modify). This approach is applicable where tunnels are "port-based" and created with explicit flow keys. In the port-based model each tenant has a separate tunnel and the port fully specifies the tunnel headers [11].

The reactive approach takes advantage of the cloud platform APIs for retrieving the tenant network properties (e.g. tunnel key, provisioning hosts, virtual ports, etc.). By correlating the properties with the information from the SDN controller, tunnel endpoints, which are involved in the tenant overlay, are discovered. This approach is necessary for "flow-based" tunnels, introduced in OpenFlow 1.3. Flow-based tunnels do not strictly specify the flow key in the tunnel interface and one OpenFlow port can represent multiple tunnels. Therefore, the matching flow rule should set the key and other required fields, such as source/destination IP addresses.

6 Tenant Traffic Classification in Underlay Network

Topology discovery for the overlay and underlay networks was explained in the previous sections, as well as path calculation between two endpoints. However, this is not enough for classifying tenants' traffic in the underlay, when it is encapsulated. Classes are essential for differentiated forwarding of overlay traffic.

Irrespective of the isolation technique chosen for the network virtualization, classifying tenant traffic in the underlay network is a challenging task. If IP block splitting or VLAN tagging are used for isolation, source/destination IP addresses or VLAN IDs suffice for tenant traffic identification, respectively. However, distinguishing tunneled traffic is not trivial. The tenant IP packet is encapsulated in the tunnel packet, and the tunnel ID (64 bits) is inside the tunnel header. OpenFlow 1.3 [10] proposes 40 match fields, and can not match on the tunnel ID in a transit node.

Therefore, either the first packet of a flow should be sent to the controller, for decapsulation, or another field should be used for the classification.

The former approach is not practical, since the overhead is significant. For the latter approach, the IPv6 header has a ”flow label” field (20 bits), which can be used for this purpose. However, there is no equivalent field in the IPv4 header. In IPv4, the DSCP field is used. This approach has two major benefits: proactive flow programming (i.e. efficient forwarding) and seamless integration with the legacy network DiffServ domains.

A third approach is to use an IP addressing scheme in the underlay which reflects the virtual network tenant and forwarding classes in addition to the location. Further implementation and evaluation of this concept is a part of the future work.

7 Endpoint Virtual Switch Architecture

Tenants’ tunneled traffic should be marked, according to its class, before leaving a host. However, when a packet is sent to a tunnel interface, the Linux routing table determines the egress interface. This decision cannot be overridden by the virtual switch. Therefore, the OpenFlow LOCAL port on the switch is configured such that the routing table will choose it as the egress for all tunneled traffic (Figure 2.2). The configuration consists of assigning an IP address from the provider underlay subnet to the LOCAL port.

Since, the LOCAL port is chosen as the egress interface, external interfaces should be added to the virtual switch. The external interfaces connect the host to the underlay network. In addition, the virtual switch is supplied with a set of rules to forward the traffic between LOCAL and external interfaces, and avoid loops in case of persistent broadcast packets.

Moreover, tunnels are created such that they copy the inner IP header ToS to the outer one. This provides visibility to the DSCP value of a packet in transit.

8 Differentiated Forwarding

This section explains the end-to-end network programming steps for implementing the overlay QoS. It consists of controlling endpoint switches of a traffic exchange and underlay transit nodes along the path.

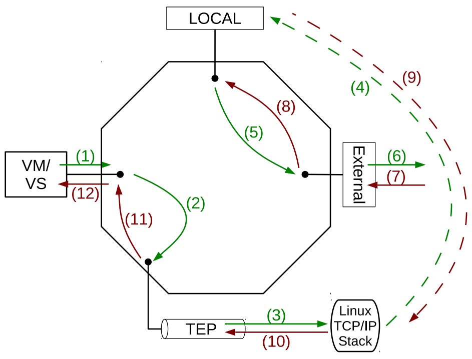


Figure 2.2: Virtual Switch Configuration

8.1 Programming Endpoints

The tenant's egress IP traffic from VMs is marked with the DSCP value derived from the endpoint pair's flavor (§6). The encapsulated packets are further processed in the switch and an external interface is chosen according to the provided policy for the tenant and the tunnel destination. The detailed steps are as follows (Figure 2.2): *1*) Marking the tenant's traffic with the chosen DSCP value. *2*) Sending the marked traffic to the tunnel. *3*) Resolving the egress port using the Linux routing table. *4*) Capturing the tunneled traffic on the switch LOCAL port. *5*) Checking the tunnel packets and marking those without DSCP by the biggest value for the given endpoint pair. *6*) Forwarding the traffic through an external interface.

The encapsulated traffic destined for a tenant VM(s) is forwarded through the underlay and received on an external interface of the destination host. It is decapsulated in the tunnel interface and forwarded to the VM(s) or another switch: *7*) Receiving traffic on the external interface. *8*) Forwarding ingress traffic from an external interface to the LOCAL port. *9*) Resolving the tunnel port. *10*) Capturing the traffic on the tunnel port. *11*) Decapsulating the packet. *12*) Forwarding the packet to the destination.

Additional measures are taken to proxy the external interface ARP request/response through the management network.

8.2 Programming Underlay

Underlay (also known as transit) nodes are networking devices in the edge, aggregation, and core networks, that are responsible for forwarding tenants' traffic between compute nodes. Tenant traffic identification is explained in §6, and §8.1 has discussed overlay traffic marking in each endpoint. Transit datapaths should be programmed such that they honor the classification made at the endpoints. The *(in_port, ip_src, ip_dst, mark)* tuple determines the class of a tenant flow in transit. However, efficiently programming the tuple across the network is a challenging task. This section proposes a few approaches for the differentiated forwarding of tenants' traffic in transit nodes.

Programming transit nodes on a given path between two endpoints

Endpoints are compute nodes which are hosting resources provisioned by a tenant. This approach finds a path between two given endpoints for a tenant, such that it complies with the tenant's virtual network flavor. Then, it programs intermediate nodes to forward the overlay traffic.

It is proactive and utilizes the forwarding table space efficiently. The proactive mechanism reduces the runtime overhead, and responds well to small flows by avoiding further communication with the controller (i.e. slow path).

Programming complete reachability matrix between all endpoints for all classes

This approach calculates routes for all network classes between all endpoints. If provisioned resources by all tenants are uniformly distributed over endpoints, it has less computational cost compared to the previous one, while the storage cost is of the same order. Like the previous one, this is proactive and has a minimal impact on the forwarding speed.

However, if the tenants distribution is not uniform, the forwarding table space is not used efficiently and may cause a state explosion.

Programming on PacketIn messages

A third approach is to wait for PacketIn messages from intermediate nodes, when they do not have a matching flow rule for a packet. The switch sends a PacketIn message to the controller, which contains the packet and its arrival port. Then, the controller finds the corresponding tenant for the flow and calculates the path for it, according to the network flavor. Finally, it programs all the nodes on the path for that flow. Therefore, other packets will have a matching flow entry, and would be forwarded in the datapath. This is a reactive mechanism for programming the network. Although it uses the table space efficiently, the overhead is significant, specifically for short-lived flows.

The evaluation (§9) focuses on the first approach, which provides a balance between the number of installed flow entries and the runtime computational complexity. In addition, the path between two endpoints can be updated dynamically, with least disruption. The new path flow entries have lower priorities compared to the old ones. Therefore, the target traffic uses the old path, while the new path is programmed across the underlay. When all flow tables are updated, the old path is removed and the traffic will match the new one. This approach ensures that packets are not forwarded to switches without matching flow entries.

8.3 Traffic Engineering Methods

Implementing traffic engineering using SDN mechanisms can be more flexible and effective compared to IP and MPLS based TEs [1]. Unified views of the networking substrates and applications, through southbound and northbound APIs, provide visibility into the network status and the applications' requirements. The SDN specification abstraction [13] makes the proactive programming and dynamic reprogramming of the network efficient, as devices are not handled individually. Therefore, the controller can efficiently react to the network status and application demand.

In addition to the chosen path between two endpoints, meters and queues of the OpenFlow switches are exploited for traffic engineering and QoS differentiation in the underlay.

Meter

A meter is a part of a flow entry instruction set. It measures the rate of all the packets which are matched to the flows to which it is attached. The meter counters are updated when packets are processed. Each meter has a set of bands which specify the minimum rate at which their action can be applied. A band can support ”drop” or ”dscp remark” operations [10]. Meter creation, configuration, and flow assignment are performed by the OpenFlow wire protocol version 1.3.0.

In our design, each traffic class has a set of meters, which are attached to the flows mapped to it. The minimum rate of a meter band is specified in the network flavor, otherwise it is REFERENCE_BW/CLASS_NUM, where the REFERENCE_BW represents the physical port speed (i.e. 1 Gbps for our test-bed) and CLASS_NUM is the traffic class specified in the flavor. When the packet rate is over the band rate, the band applies and packets are dropped.

Queue

A port in an OpenFlow switch can have one or more queues, where flow entries are mapped to them. A queue determines how flows are treated, using a set of properties: a minimum rate, a maximum rate, and an experimenter property [10]. The maximum rate of a queue is set to REFERENCE_BW/(MQ-QP), where MQ and QP are the maximum number of queues and the queue priority, respectively. In contrast to the meter, queue configuration is handled by the OVSDB protocol, while flow mapping is done by the OpenFlow wire protocol.

Flows of a tenant are forwarded to a queue according to the virtual network flavor priority. The queue priority determines which one is served first. Therefore, in addition to the rate limiting function, overlay traffic with a higher priority class is forwarded before lower priority ones.

Meters are more useful for fine-grained rate limiting, since a switch can support more meters than queues, and meters are per-flow while queues are per-port. Therefore, a meters' configuration can be more granular. Although the number of queues in hardware is limited, they provide better guarantees for traffic prioritization and are more effective for coarse-grained policies. A combination of meters and queues can deliver complex QoS frameworks [10].

For the evaluation, minimum rates of meter bands and maximum rates of queues are chosen with significant differences for each flavor to clearly visualize the impact on the network QoS.

9 Evaluation

Flavors should create a clear distinction between virtual networks' QoS. An extensive evaluation framework is developed to measure a variety of QoS parameters and study the system behavior under different scenarios.

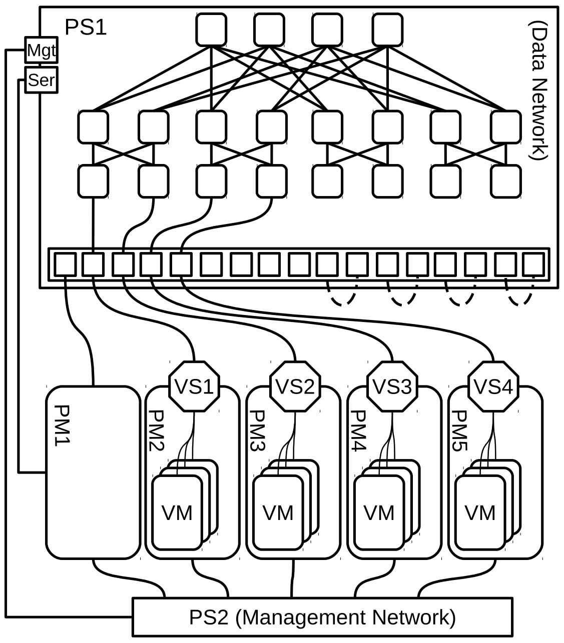


Figure 2.3: Evaluation Test-Bed

9.1 Evaluation Framework

The framework has a modular architecture and allows new experiments to be added as plug-ins. The experiment covers all scenarios that should be assessed. It requires a set of parameters, which are used to create a group of sub-experiments. A sub-experiment is executed according to the specified execution type and the results are comparable. Some of the parameters used for the virtual network throughput analysis are as follows:

- *Instances range*: The minimum and maximum number of virtual machine instances which are involved in the measurements.
- *Networks range*: The minimum and maximum number of virtual networks, created in this experiment.
- *Classes*: The traffic classes (specified in the flavors), assigned to virtual networks, where lower class numbers have higher priorities and better QoS.
- *Instance distribution*: The distribution of instances over networks.
- *Network distribution*: The distribution of networks over classes.

- *Path length*: The maximum number of hops between two instances of a class.
- *Instances execution type*: Determines whether instances should perform the task simultaneously or sequentially (i.e. `i:{true,false}`).
- *Classes execution type*: Determines whether networks of a class should execute the task in parallel or in series (i.e. `c:{true,false}`).
- *TE Strategy*: Specifies the traffic engineering method(s) in the underlay network (i.e. `none, meter, queue, meter_queue`).
- *Failure threshold*: The upper threshold for the permitted number of failures before a sub-experiment is terminated.

Before executing a sub-experiment, all the configurations are set to their default values and the cloud platform is reinitialized. This will avoid propagation of side-effects from previous sub-experiments and increase precision. After the reinitialization, virtual networks and instances are created according to the aforementioned parameters. Then, the SDN controller programs the network and the framework waits for the instances to become reachable. Once enough instances are reachable, the framework instructs instances to perform the measurement and report back. If the number of reachable instances is less than the failure threshold, the process is terminated with an error report and the next sub-experiment is started.

When all reachable instances finish their tasks, the reports are stored and processed. Finally, all instances and networks are deleted.

In addition to the input parameters, each report contains a set of task-specific results. For instance, the throughput measurement reports on TCP/UDP rate, Rx/Tx CPU utilization, Round Trip Time (RTT), retransmission occurrence, number of report errors, number of missing values, start/end time and hosting hypervisors. The processing phase consists of report classification based on common parameters, data aggregation, statistical analysis and plotting. The outcomes are presented in two forms, TE strategy comparison (§9.4) and execution type comparison (§9.5).

9.2 Execution Types

Evaluating virtual network performance is not a trivial task [14]. Tenants of a cloud service provision and release resources dynamically; they also have workloads with varying traffic characteristics throughout the day.

Table 2.1: Traffic Engineering Method Properties

	Meter	**Queue**	
Class	**Min Rate**	**Priority**	**Max Rate**
1	1 Gbps	6	1 Gbps
2	500 Mbps	5	500 Mbps
3	333 Mbps	4	333 Mbps
4	250 Mbps	3	250 Mbps

Therefore, virtual network QoS should be studied under different scenarios where a realistic workload is simulated in the network.

An execution type defines the scheduling method of a sub-experiment. It determines how competing networks with different classes and their instances should perform the throughput measurement. To limit the scope of the evaluation, two parameters are considered: network classes and instances execution types. The network class execution type specifies whether throughput of competing networks should be measured concurrently. Whereas, the instance execution type controls the parallel or series throughput measurement between two instances in a network.

The number of concurrent individual measurements depends on the execution type, and estimated values are presented in Table 2.2. However, it might not be the exact number in an arbitrary given point of time, because a strict mechanism is not used for the experiment scheduling. As an example, some instances of a network class may become reachable sooner than others, when all networks and their instances have been requested simultaneously. Therefore, it may show some stochastic behaviors and noises in the results.

Although concurrent execution of the measurements decreases each class throughput, higher priority classes perform better than lower priority ones.

9.3 Evaluation Setup

The platform is deployed on 5 Intel NUCs, with Core i5 processors, 16 GByte memory, 140 GByte SSD storage, and two Gigabit Ethernets. Each node is connected to the management and data networks. The management network uses an 8-port Gigabit switch, and the data network uses an OpenFlow capable Pica-8 P-3290 48-port switch. The whitebox

Table 2.2: Scheduling Method Properties

Type	# Concurrent Individual Measurements
`c:false,i:false`	*1*
`c:false,i:true`	*(#instances choose 2)*
`c:true,i:false`	*#classes*
`c:true,i:true`	*#classes* × *(#instances choose 2)*

switch has 8 physical queues (0-7), and the highest priority queue (i.e. queue 7) is dedicated to the switch-controller communication.

As shown in Figure 2.3, the SDN (PM1) and cloud platform (PM2) controllers have dedicated nodes and the rest are compute nodes (PM3,4,5). A Fat-Tree topology with K-port commodity switch is emulated in the whitebox switch (PS1), where K is 4. As the whitebox switch does not support Open vSwitch patch-port, the remaining physical ports are connected in pairs to patch the emulated switches.

The SDN controller modules are developed as part of the OpenDaylight project, and the cloud platform is OpenStack. The implementation has 10,613 lines of code and is available for the community, along with the raw evaluation data and additional analysis [5].

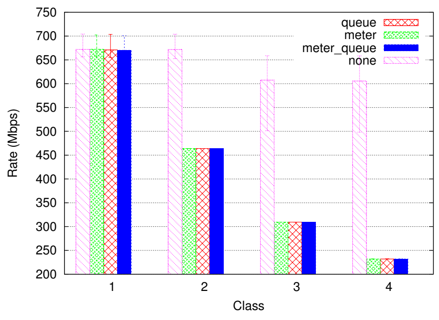
The virtual machine image used for creating instances is a heavily customized version of CirrOS[3] based on Buildroot[2]. The VM flavor is m1.nano with one vCPU and 256 MB memory.

9.4 Traffic Engineering Strategy Analysis

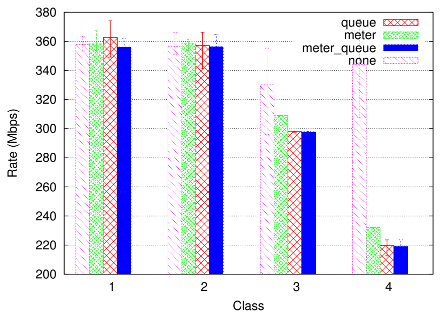
Figure 2.4 represents the mean TCP throughput between any two instances on a network, with a distinct class, that are hosted on different hypervisors. Each sub-figure depicts a specific execution type, and data series are different strategies.

Figure 2.4a plots the TCP throughput when at any point of time measurement is performed between two instances of a single class. As explained in §9.2, since this scheduling method offers the least concurrency, each class achieves the maximum throughput under different TE strategies. When the only class enforcement method is the chosen path (i.e. strategy is `none`), the differences between classes' throughput are not significant. However, other strategies make considerable differences between classes.

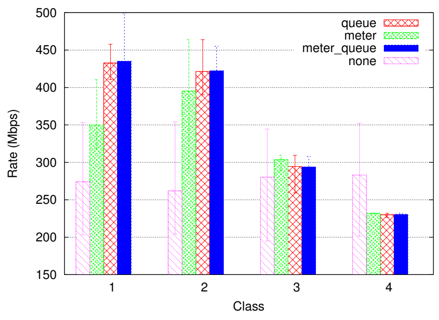
As shown in Figures 2.4b, 2.4c and 2.4d, when instances or networks



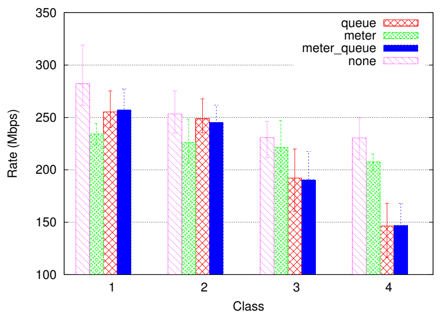
(a) Execution Type `c:false,i:false`



(b) Execution Type `c:false,i:true`



(c) Execution Type `c:true,i:false`



(d) Execution Type `c:true,i:true`

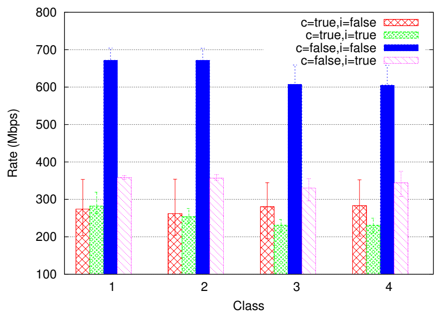
Figure 2.4: Comparing the impact of QoS strategies on TCP throughput for different execution types

execute the measurements concurrently, the TCP throughput is decreased.

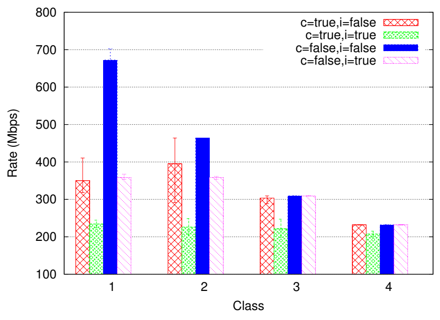
9.5 Execution Type Analysis

Figure 2.5 also represents the mean TCP throughput between any two instances, where each sub-figure depicts a specific strategy, and data series represent execution types. It can be observed that the throughput is mandated by the class priority, and it is independent of the scheduling method.

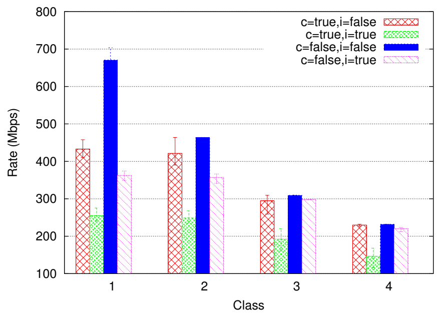
In Figure 2.5c, each queue has a maximum rate of REFERENCE_BW/CLASS_NUM; the TCP throughput of all classes are limited by this constraint. However, if the maximum rate is set to REFERENCE_BW, when classes are not executed concurrently (i.e. `c:false,i:*`), lower priority classes will perform as good as the strategy `none` case in Figure 2.5a; whereas the throughput in other execution types remain the same. It is due to the queue scheduling



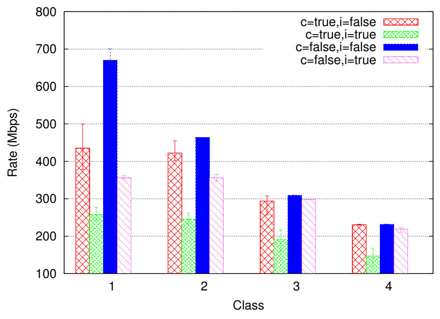
(a) Strategy `none`



(b) Strategy `meter`



(c) Strategy `queue`



(d) Strategy `meter_queue`

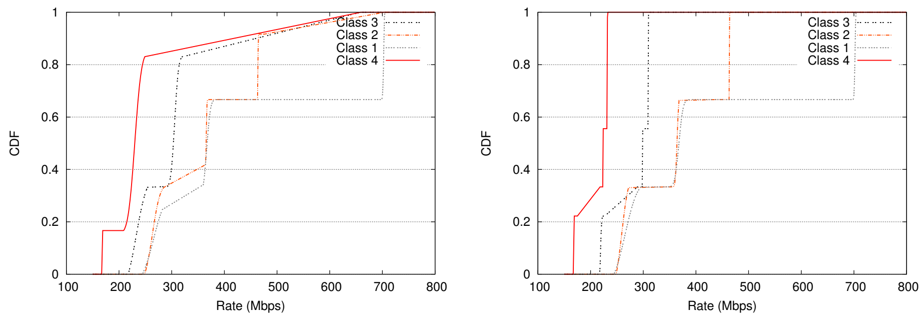
Figure 2.5: Comparing the impact of execution types on TCP throughput for different QoS strategies

mechanisms of the switch, where the higher priority queues are served first.

Figure 2.6 depicts the CDF of the 90th percentile TCP throughput for each class, independent of the scheduling method. Figure 2.6a presents the CDF with and without traffic engineering mechanisms explained in §8.3, while Figure 2.6b only includes the 90th percentile throughput when at least one TE method is employed. It can be deduced that higher priority classes (lower class numbers) perform better than the lower priority ones.

10 Conclusion

Virtual network flavor is a crucial feature, but missing in cloud platforms design. The proposed approach utilizes SDN mechanisms for delivering flavors, and providing QoS for overlay virtual networks. SDN abstractions make overlay traffic classification and underlay traffic engineering much



(a) TE methods: `none, meter, queue, meter_queue`

(b) TE methods: `meter, queue, meter_queue`

Figure 2.6: CDF of the 90th percentile TCP throughput for each class independent of the experiment scheduling approach.

more efficient. The logically centralized SDN controller not only provides a unified view of the network through southbound APIs, but also has visibility into applications' demands through northbound APIs. Standard (i.e. OpenFlow 1.3 [10]) and open (i.e. OVSDB [12]) protocols are used for the control and management planes, and all operations are performed using them. Therefore, there is no manual configuration involved for network management and programming.

Acknowledgment

This work is done in collaboration with Norwegian NREN (UNINETT).

References

[1] Ian F. Akyildiz et al. "A roadmap for traffic engineering in SDN-OpenFlow networks." In: *Computer Networks* 71 (Oct. 2014), pp. 1–30.

[2] *Buildroot*. URL: `http://buildroot.uclibc.org/`.

[3] *CirrOS*. URL: `https://launchpad.net/cirros`.

[4] B. Davie and J. Gross. *A Stateless Transport Tunneling Protocol for Network Virtualization*. Internet Draft (Informational). Network Working Group, Oct. 2014. URL: `https://tools.ietf.org/html/draft-davie-stt-06`.

[5] *Differentiated Traffic Forwarding for Cloud Tenants.* URL: http://www.ux.uis.no/~aryan/docs/dc/vnet-flavor/.

[6] Mohammad Al-Fares et al. "Hedera: Dynamic Flow Scheduling for Data Center Networks." In: *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation.* NSDI'10. 2010, pp. 19–19.

[7] Nikhil Handigol et al. "Plug-n-Serve: Load-balancing web traffic using OpenFlow." In: *ACM SIGCOMM Demo* (2009).

[8] Ashiq Khan et al. "Quality-of-service (QoS) for virtual networks in OpenFlow MPLS transport networks." In: *Cloud Networking, 2013 IEEE 2nd International Conference on.* CloudNet'13. Nov. 2013, pp. 10–17.

[9] M. Mahalingam et al. *Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks.* RFC 7348 (Informational). Internet Engineering Task Force, Aug. 2014. URL: http://www.ietf.org/rfc/rfc7348.txt.

[10] Open Networking Foundation. *OpenFlow Switch Specification version 1.3.0 (Wire Protocol 0x04).* Apr. 2012.

[11] Ben Pfaff. *Open vSwitch Manual.* URL: http://benpfaff.org/~blp/ovs-fields.pdf.

[12] Ben Pfaff and Bruce Davie. *The Open vSwitch Database Management Protocol.* Request for Comments 7047. IETF, Dec. 2013. URL: http://www.ietf.org/rfc/rfc7047.txt.

[13] Scott Shenker et al. "The future of networking, and the past of protocols." In: *Open Networking Summit* (2011). URL: http://www.opennetsummit.org/archives/apr12/site/talks/shenker-tue.pdf.

[14] Aryan TaheriMonfared and Chunming Rong. "Flexible Building Blocks for Software Defined Network Function Virtualization." In: *2014 IEEE 10th International Conference on Quality, Reliability, Security and Robustness in Heterogeneous Networks (QShine).* IEEE. 2014.

[15] Richard Wang, Dana Butnariu, and Jennifer Rexford. "OpenFlow-based Server Load Balancing Gone Wild." In: *Proceedings of the 11th USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services.* Hot-ICE'11. 2011, pp. 12–12.

[16] Jin Y Yen. “Finding the k shortest loopless paths in a network.” In: *Management Science* 17.11 (1971), pp. 712–716.

Paper 5:
On the Feasibility of Deep Packet Inspection for Multi-Tenant Data Center Networks

On the Feasibility of Deep Packet Inspection for Multi-Tenant Data Center Networks

A. TaheriMonfared[1], F. Ramos[2], C. Rong[1]

[1] Department of Electrical Engineering and Computer Science, University of Stavanger
4036 Stavanger, Norway

[2] Department of Informatics, University of Lisbon
1749-016 Lisbon, Portugal

Abstract:

Cloud data centers comprise a multi-tenant and challenging networking environment. In this environment, a passive deep packet inspection service for the internal traffic flows is considered to be impractical and not scalable. Several problems hinder the service deployment, including the traffic volume, choke-point risks, limited flow mirroring capabilities, and cost.
These challenges can be relaxed by distributing the monitoring service over commodity switches and hosts. The distribution and its effectiveness is studied by formalizing the service as an optimization problem. The objective is to find a monitoring switch and host for each designated traffic flow such that the infrastructure constraints are not violated and the service overhead is minimized. Software-Defined Networking mechanisms are used for the service node orchestration and network programming. These mechanisms implements flow mirroring and program the network according to the optimizer solutions. The results indicate that the approach is cost efficient and scalable. For instance, 10% of traffic can be monitored by using 0.5% of hosts for processing and 20% switches for traffic mirroring.

1 Introduction

Deep Packet Inspection (DPI) is a common service in most middleboxes [7]. Enterprises use DPI for understanding the network traffic characteristics, managing bandwidth utilization, and improving the premises security. Typically, this service is deployed at the network borders to achieve a complete view of the ingress and egress traffic.

Cloud computing model provides a shared pool of resources for a dynamic set of tenants with varying requirements and policies [24]. The model reduces the barriers for introducing new businesses and decreases the infrastructure investments. Therefore, it is a compelling environment for various customers, including large-scale enterprises.

Tenants' applications can have malicious intents or become compromised. An attack origin and target can be inside or outside the cloud. Those attacks with an endpoint outside the cloud infrastructure are easier to detect using legacy methods (e.g. DPI at the borders). However, attacks within the cloud infrastructure are significantly harder to detect [25]. The reasons are various, including the lack of multi-tenancy support in legacy solutions, and scalability challenges of the internal traffic monitoring.

Although cloud providers offer a variety of middlebox services, the DPI service has captured much less attention. Therefore, an enterprise tenant should provision virtual machines and deploy the DPI service. Then, the tenant needs to steer traffic through the virtual DPI machines. This approach has several challenges, including the lack of control on forwarding mechanisms of virtual networks, bottleneck risks, and high costs. Recent solutions have proposed distributed DPI [7] and middlebox outsourcing [34]. However, these solutions do not consider the overhead of distribution or outsourcing on the network infrastructure. The model proposed in this paper shows that traffic mirroring for payload analysis (e.g. DPI) has a considerable overhead for cloud data center networks and previous middlebox distribution proposals are not applicable to the DPI service.

Payload analysis has three fundamental requirements: capturing, transporting, and processing the designated traffic. The packet can be made available by port mirroring (or traffic mirroring, in general) in a switch that is on the path of the packet. The mirrored traffic are processed in service nodes (i.e. monitoring hosts). In large-scale data centers, the service nodes can not be on-path for various reasons [17]. For instance, on-path service nodes have scalability issues, disrupt the bisection capacity of the topology, and become network choke-points. Moreover, this type of services, with high volume and velocity messages, can not be outsourced to off-premises resources. Therefore, the service nodes should be off-path and connected to the edge layer of the data center network.

This paper proposes a distribution mechanism for the DPI service. Contrary to a traditional enterprise setting, in a multi-tenant cloud data center deploying a DPI at the border is not enough. Moreover, distributed solutions that are oblivious to the network do not scale. The proposed solution takes into account the service overhead on the networking and computing resources. In addition, it leverages on SDN mechanisms to offer a flow-based DPI solution for multi-tenant networks. This service is deployed by the cloud provider for monitoring the data center network. It also can be exposed to tenants as a middlebox service.

2 Related Work

Agent-based Solutions

Seawall [35] deploys rate controllers at the edges (i.e. hypervisors) for performance isolation, without trusting the hypervisors. Seawall benefits from a software-based network monitoring solution, where receiving hypervisor provides low cost feedback to the sending hypervisor. The extension is implemented in the virtual Network Interface Card (NIC), and monitors congestion signals (e.g. ECN mark). This monitoring approach only detects congestion, and does not support packet analysis.

Hasselmeyer et al. [14] propose an agent-based monitoring approach for cloud data center. Each resource has a set of agents which generate monitoring events and send them to monitoring applications through a messaging bus. The monitoring applications can be hosted by the cloud tenant or infrastructure provider. Although the architecture is scalable for a small volume of discrete events, it is not applicable to the network traffic analysis.

Traffic Statistics

InteMon [16] is a real-time mining system for large clusters. It uses Simple Network Management Protocol (SNMP) for retrieving metrics from nodes, and can not analyze packets payloads.

OpenNetMon [1] is an SDN controller module that provides per-flow statistics for fine-grained Traffic Engineering (TE). It derives flow throughput, delay, and loss using PacketIn, PacketOut, and FlowRemoved OpenFlow [23] messages in combination with an adapting polling of switch flow statistics. Similarly, OpenTM [40] proposes a flow statistics polling mechanism for estimating the network traffic matrix. It shows that a higher accuracy can be achieved with a moderated switch overhead, when the querying strategy uses a non-uniform distribution with a higher probability for polling switches closer to the flow destination. However, flow statistics and network traffic matrix are not sufficient for analyzing sophisticated attacks.

Monitoring Traffic Routing

OpenSAFE [3] is a system for routing the monitoring traffic from a source (i.e. observation point) to a set of sinks while selecting a subset of traffic. It uses a high-level language, called ALARMS, for route management. The language instructions are programmed in the network using the OpenFlow protocol. However, OpenSAFE does not control the switch management plane, and leaves it for the network administrators. In addition, the costs of switch configuration, network utilization, and host initialization are not reflected in the route management solution. Traffic mirroring can overload the network forwarding capacity and cause significant processing costs for switches and hosts. Therefore, it is essential to validate a monitoring request before implementing it.

Traffic Sampling

FleXam [36] is an extension to the OpenFlow capable switch that facilitates per-flow packet sampling. It supports two sampling types: deterministic and stochastic. The sampling technique can be used for capturing the designated traffic flow, and it complements the approach introduced here.

Processing Frameworks

Gigascope [9] is a Data Stream Management System (DSMS) for network applications. It has a SQL-like query language with language primitives and operators which are tuned for network analysts. OpenNF [11] is a control plane application. It manages the network forwarding state and internal network function (NF) state. OpenNF scales NF instances up or down without disrupting traffic flows. The objective is achieved by extending NFs and synchronizing states among them. The processing frameworks (e.g. [9, 11, 39, 7]) complement this work. These solutions can be deployed on the monitoring hosts for a scalable, flexible, and real-time analysis of collected traffic.

Traffic Mirroring

Planck [32] is a network monitoring system that provides measurements an order of magnitude faster than the state-of-the-art solutions. It uses port mirroring on all switches in a data center and directly connects collector hosts to switches. An SDN controller installs mirroring flow rules in the switches and provides collector hosts with network topology and routing information. The information is used to identify traffic flows and in/out ports of a packet in a switch. The SDN controller can subscribe to the events dispatched by the collectors to react to network congestion. However, it has several limitations. First, directly connecting servers to switches in different level of a fat-tree is not scalable and feasible. Second, mirroring traffic in all switches is not efficient, since the same flow visits several switches in the network and mirroring it in all switches increases the monitoring cost significantly.

Service Chaining and Orchestration

Service Function Chaining (SFC) tackles several challenges in the delivery of network services. For instance, one challenge is the tight coupling of service models, network topology, and the infrastructure resources. Another one is the services dependencies, such that a modification in one service impacts other services [29]. The SFC architecture introduces an approach for service function deployment, that facilitates service reordering, topology independence, and metadata exchange between resources [13].

To some extent, this work follows the SFC architecture and improvises where the architecture puts too much burden or has practical limitations. For instance, the monitoring host implements Service Function (SF), the monitoring switch is Service Function Forwarder (SFF), the SDN controller enforces the classification, and generic tunneling techniques are used instead of the SFC encapsulation with Network Service Header (NSH) [30].

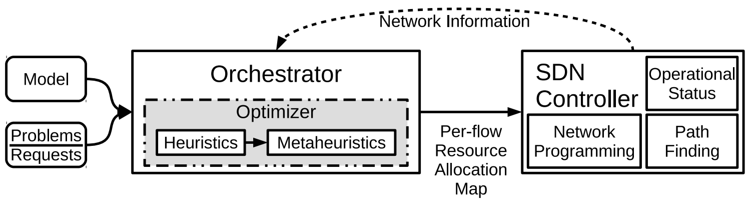


Figure 2.1: Components of the system.

3 Design

The objective is to distribute the payload analysis service, for a set of designated traffic flows, across commodity off-path service nodes using commodity on-path switches. The approach, to achieve this objective, has two phases. First, solving the combinatorial optimization problem of finding an optimal set of hosts and switches for a given set of traffic flows. Second, programming the network to reflect the solution from the previous step.

Solving the optimization problem and studying previous service distribution proposals demand a network model. The model should capture the properties of a data center network and the characteristics of a payload analysis service. The model is discussed in Section 4. The optimization problem has a large search space and requires efficient algorithms. Two approaches for finding shortest paths for the fat-tree topology are introduced in Section 5. The optimization problem is solved for a variety of problem instances (i.e. monitoring scenarios) with different search algorithms. The solutions and search algorithms are evaluated in Section 6. Flow mirroring is an important aspect of the solution that facilitates fine-grained per-tenant payload analysis. The mechanisms for implementing flow mirroring in an OpenFlow-capable network are described in Section 7.

Consequently, the system has two essential components: an orchestrator for coordinating monitoring tasks between monitoring entities, and a transport mechanism for delivering monitoring messages between involved entities. Figure 2.1 depicts the components of the system. The implementation and evaluation framework are open-source and the raw data is available online[23].

[23]`http://www.ux.uis.no/~aryan/docs/dc/mon/`

4 Network Model

A network infrastructure model is required for handling the increasing complexity and growing requirements of new network architectures and application services [18].

The model captures various aspects of a data center network, that are required for solving the optimization problem. The model builds a network topology according to the properties specified by the input parameters. The focus of this paper is data center network with the Fat-Tree topology [10]. However, the model accepts an arbitrary topology and is not restricted to a specific architecture.

Figure 2.2 depicts a high-level network architecture and infrastructure resources. The following describes the model components that are required for solving the problem.

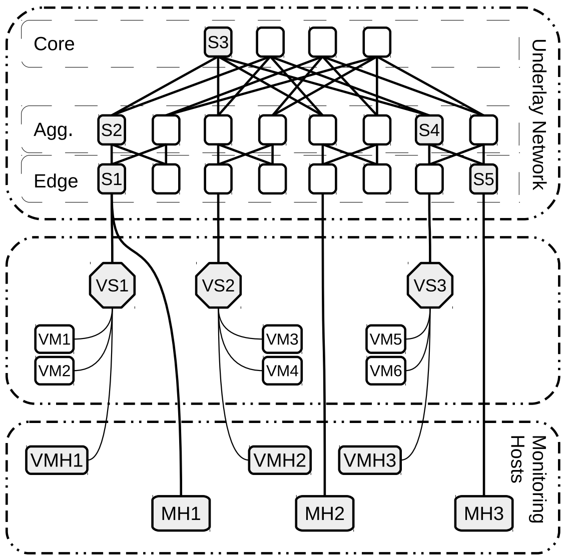


Figure 2.2: High-Level Architecture (S: switch, VS: virtual switch, VM: virtual machine, MH: monitoring host, VMH: virtual monitoring host)

4.1 Solution

A candidate solution for this monitoring problem specifies a monitoring switch and a monitoring host for each target traffic flow, such that none of hard constraints are violated. There can be more than one solution for this problem, and finding the best solution in a limited time is not feasible.

Therefore, this is a NP-complete combinatorial optimization problem [15].

4.2 Monitoring Switch

A monitoring switch, whether physical ($\mathtt{S_i}$) or virtual ($\mathtt{VS_i}$), is responsible for mirroring a subset of forwarded traffic to a designated monitoring host. The traffic subset definition depends on the granularity of the monitoring policy and this paper uses IP flows for this purpose. A flow is "a sequence of packets sent from a particular source to a particular unicast, anycast, or multicast destination" as defined by RFC3697 [31]. Such a definition can also cover the sequential packets between two endpoints, where one or more fields from the 5-tuple IP header are not available, because of fragmentation or encryption.

The monitoring switch for a traffic flow is one of the switches on the flow's path. A monitoring switch can communicate with one or more monitoring hosts. However, a particular traffic flow is mirrored to a single monitoring host at a given point of time.

Any switch in the network can become a monitoring switch if it supports mirroring and can establish an isolated path to one or more monitoring hosts. The isolated path can be achieved using tunneling techniques (e.g. Virtual eXtensible Local Area Network (VXLAN) [21]), VLAN tagging, dedicated links, etc.

Switch Cost

In the model, when the monitoring function of a switch is enabled an initial cost (`sw.init_cost`) is incurred. For each additional flow, which is monitored by the switch, a significantly smaller cost is added. The reuse cost is a function of the switch initial cost and is determined by a ratio variable (`sw.perflow_reuse_cost_ratio`).

The service not only introduces costs for monitoring switches, but also has overheads for switches and links between the monitoring switches and the monitoring hosts. Transit switches forward the extra monitoring traffic and transit links face higher utilization. Therefore, the switch forwarding and fabric capacities and the link speed are parts of hard constraints. While, the switch initialization and reuse costs and the link utilization are in soft constraints.

The switch forwarding capacity is represented in packet-per-second (pps) and the fabric is in bit-per-second (bps).

Switch Location

The monitoring switches locations and their relative distance to monitoring hosts have significant impact on the accuracy and overhead of the service. The mirroring can take place at different switches in the underlay network (Figure 2.2):

Source Edge (S1) The traffic observed at a switch close to the flow source may not be identical to the one received at the destination. Middle-box functions (e.g. Network Address Translation (NAT)) as well as packet loss are some reasons for such an inaccuracy. The distance of the switch from the monitoring host varies according to the host location. It can be one, three, or five for a host connected to the same edge, same pod, or a different pod, respectively. Although the utilization of the links closer to the edge are lower than the ones in the core, their loss rates are higher [5]. Traffic burstiness (i.e. ON/OFF pattern for packet arrivals) in the edge is one of the reasons for the higher loss. If the monitoring switch does not exploit a buffering technique, the loss rate may increase even further.

Aggregation (S2, S4) The aggregation layer has a lower utilization compared to other layers [5]. The distance of a switch in the aggregation layer varies between two and four and it depends on the monitoring host location (i.e. local or remote pod). A monitoring switch in the aggregation layer provides higher accuracy for a given flow, when it is closer to the flow destination. However, the amount of traffic flow passing through an aggregation switch is limited to a subset of inter-pod and intra-pod but not intra-edge flows. Thus, this layer has limited visibility and a single aggregation switch can not observe the whole traffic of the pod.

Core (S3) Core switches observe traffic from inter-pod flows. Since they are in the middle of the path between two endpoints, they provide a moderate accuracy. For any core switch, the monitoring host is reachable in three hops.

Destination Edge (S5) It has some similarities with Source Edge, but it is more accurate. However, the overhead of monitoring the broadcast and

Monitoring Switch Location	Coverage for Flows from/to			
	Local vEdge	Local Edge	Local Pod	Remote Pod
vEdge	Y	P	P	P
Edge	N	Y	P	P
Aggregation	N	N	P	P
Core	N	N	N	P

Table 2.1: Monitoring switch coverage feasibility for flows from different sources (Y: Yes, N: No, P: Partial)

multicast traffic is high, because the same flow is captured several times in the destination domain or address group. In contrast to aggregation and core switches, an edge switch provides complete coverage for flows with the same source or destination, when they are forwarded through different transit nodes (e.g. Equal Cost Multi-Path routing (ECMP)). The coverage is crucial where the processing logic is not distributed and a single monitoring host requires access to all flows between two end-points.

4.3 Virtual Monitoring Switch

As shown in Figure 2.2, a virtual switch connects a set of virtual machines together. For the sake of simplicity, virtual machines of all tenants, in a physical machine, are connected to the same virtual switch.

The cost of enabling and reusing the monitoring function for a virtual switch is similar to the one defined for the physical switch (Section 4.2). However, the location of virtual switch is limited to the edge layer (vEdge).

Moreover, switches in the underlay (e.g. physical switches) do not have visibility into intra-virtual switch traffic between virtual machines. In addition, it is easier to detect network virtualization techniques and program tenants' virtual networks in the endpoint virtual switches [38].

Table 2.1 depicts a switch coverage based on the switch location and the traffic pattern.

4.4 Monitoring Host

Monitoring host is a physical or virtual machine with packet processing capabilities. It can be a dedicated node with hardware acceleration module (e.g. Open-Source Network Tester [2]) or a commodity server with monitoring software. The initial cost of adding a new monitoring host (`monitoring_host.cost`) is significantly high. In addition, assigning new flows to a host increases the host processing overhead. Therefore, the model uses the processing capacity as a hard constraint, and the overhead as a soft constraint.

A physical monitoring host can be attached to any switch in the edge layer, and a virtual one is connected to a virtual switch. The host's network connectivity is a major bottleneck for the monitoring. Increasing the networking capacity of the host can be achieved by using extra network interfaces on the host. The extra interfaces can be connected to the same edge or another edge on the same pod. The model uses bonded interfaces for increasing the capacity, and the number of interfaces is a configuration parameter. The default value is 4.

The monitoring host virtualization is effective in two scenarios. First, the provider can use the scalability and increases or decreases the monitoring capacity. The initialization cost of a virtual monitoring host is much lower than the physical one. However, the processing capacity of a virtual machine and the constraints on the hardware acceleration are the limiting factors. Second, the virtual monitoring hosts can be used for delivering a dedicate monitoring service for each tenant.

Moreover, the failure of a monitoring host doesn't affect the end-to-end traffic forwarding. However, it may cause monitoring data loss.

4.5 Link

A link in the model connects two nodes of the network. It has a limited capacity (i.e. speed), and traffic mirroring increases the link load significantly. For a single mirrored flow, all links between the monitoring switch and host face an additional traffic volume which is more than the flow rate (due to the tunneling overhead). Therefore, the link capacity is considered a hard constraint, and the overall network utilization is a soft constraint.

The model can assign different costs to links according to their endpoints' layer. For instance, links which are terminated at the edge layer can have lower costs compared to the links connecting aggregation to core. As a consequence, the model utilizes edge links more frequently. This is useful when the core utilization is high and an extra monitoring traffic should be avoided.

In addition, the link cost can be scaled up or down using a parameter called pod-sensibility (PS). Decreasing the PS value increases the link cost, and results in choosing monitoring hosts and switches which are closer

Monitoring Switch Location	Distance from Monitoring Host (MH) in				Distance from vMonitoring Host (vMH) in			
	Local vEdge	Local Edge	Local Pod	Remote Pod	Local vEdge	Local Edge	Local Pod	Remote Pod
vEdge	x	2	4	6	1	3	5	7
Edge	x	1	3	5	-	2	4	6
Aggr	x	-	2	4	-	-	3	5
Core	x	-	-	3	-	-	-	4

Table 2.2: Distances between monitoring switches and hosts

to each other. In other words, PS determines the impact of monitoring switch-host distance on the total cost. The link cost is calculated using Equation 2.1.

$$link_{cost} = \frac{switch_{cost} * flow_{rate}}{link_{speed} * link_{podsensibility}} \tag{2.1}$$

Table 2.2 has a summary of the number of links (i.e. distance) between two nodes.

4.6 Traffic Flow

A set of flows models the traffic in the network. A flow is defined by a 6-tuple, source and destination addresses and ports, protocol number, and rate. Flows are generated according to three parameters that determine the traffic distribution patterns in a data center network.

Parameters are inter-pod (*p1*), intra-pod (*p2*), and intra-edge (*p3*) probabilities. The pod traffic distribution parameters (*p1, p2*) specify presence of a flow between two hosts in different pods or in the same pod (but not on the same edge). The edge distribution parameter (*p3*) determines the existence of a flow between two hosts connected to the same edge.

Equations 2.2, 2.3, and 2.4 present the inter-pod, intra-pod, and intra-edge traffic volume. F_i is the number of flows that belong to the i^{th} communication pattern. C_i is the coefficient for determining the number of flows within the i^{th} pattern. k is the number of switch ports in the Fat-Tree topology. In a cloud data center, F_3 is approximately 75% of the total traffic (i.e. $F_1 + F_2 + F_3$) [4].

$$F_1 = p1 * C_1 * \frac{(k^5) * (k-1)}{16} \tag{2.2}$$

$$F_2 = p2 * C_2 * \frac{(k^4) * (k-2)}{16} \tag{2.3}$$

$$F_3 = p3 * C_3 * \frac{(k^3) * (k-2)}{8} \tag{2.4}$$

4.7 Score

A solution is evaluated based on its score. The score determines the solution's impact (i.e. cost) on the network. It has two parts: hard and soft scores. The hard score represents the broken hard constraints. A feasible solution should have a hard score of zero. For instance, a link utilization can not be more than its capacity, otherwise the link becomes congested. The soft score represents the total overhead that should be kept as low as possible (i.e. broken soft constraints) .

5 Path Finding

The search domain for finding candidate solutions is large. Finding a candidate solution requires calculating shortest paths between many endpoints, which introduces a high cost and reduces the search speed significantly. Therefore, a fast algorithm for finding shortest paths is needed [28, 22, 33].

The Fat-Tree topology is a structured graph. The structure is exploited to introduce deterministic algorithms for calculating end-to-end paths. Since there can be more than one path with equal costs between any two given hosts, the flow's hash value is used to choose a path from a set of candidate paths. The benefit of a hash-based path calculation is twofold. First, it makes the cost calculation and the modeling deterministic. Second, it distributes the traffic load between equal cost paths uniformly.

5.1 State Machine Path Finder

The first algorithm benefits from a finite-state machine as shown in Figure 2.3. The state machine is used to validate and limit moves, while traversing the graph. It starts from the state S and all other states are accepting states. Since the topology is wide and has a limited depth, the algorithm

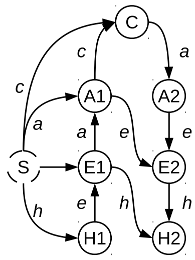


Figure 2.3: A finite-state machine for improving the performance of graph traversal in fat-tree topology. States are S: starting state, H{1,2}: host, E{1,2}: edge, A{1,2}: aggregation, C: core. Inputs h, e, a, and c correspond to visiting a link that leads to a host, an edge switch, an aggregation switch, or a core switch.

performs a Depth-First Search (DFS) and validate each move in the state machine. The moves are determined by the link type, which is the link destination.

Although this approach for graph traversal is much faster than the one without the state machine, it is not efficient for finding all paths between a significant number of node pairs in a large topology. The primarily use-case of this approach is to find all paths to all possible destinations of a node.

5.2 Numeric Path Finder

The numeric approach labels all switches and hosts with unique identifiers, as shown in Figure 2.2. Then, it takes advantage of the mathematical relations between numeric labels and calculates paths between two numbers using a set of simple functions. The costly part of the algorithm is the graph lookup, which finds the corresponding destination for a given link. The time complexity is $\mathcal{O}(k)$, where k is the number of ports in the switch, and space complexity is bounded.

Figure 2.4 compares the performance of the numeric method and Yen's K Shortest loopless Paths (YKSP) [41]. YKSP uses Dijkstra for finding the shortest path between two nodes. Each method has two derivatives, that find the paths between two nodes with or without calculating and storing the sub-paths between intermediate nodes. The y-axis is the mean number of node pairs that have been processed per second. The x-axis is the number of ports per switch. The results demonstrate that the numeric method performs significantly better than YKSP, in this topology. The deficiency in the k=*4* case is caused by the limited measurement

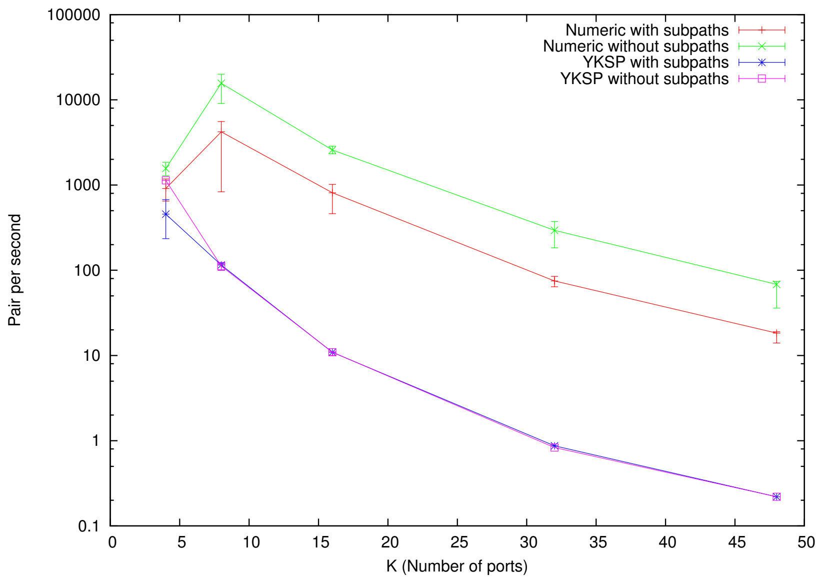


Figure 2.4: Performance comparison of the numeric approach and YKSP algorithm.

granularity and the topology size. In other words, the bootstrapping time for small size networks outweighs the algorithm performance gain.

When an end-to-end path between two nodes is calculated, a set of sub-paths between intermediate nodes are derived. The calculation and storage of sub-paths reduces the number of processed pairs per second. However, it improves the efficiency when the objective is to find a large number of paths. Figure 2.5 depicts the number of generated paths after finding paths between the maximum of 2000 host pairs.

Total sub-paths of a path is: $\sum_{i=0}^{L}(L-i)$, where L is the path length.

Although the exact performance is dependent on the specific implementation, the improvement is fundamental and regardless of the implementation.

6 Evaluation

This section begins with explaining the methods used for solving the optimization problem and problem instances (i.e. inputs). Then, the solutions are evaluated and the results are interpreted.

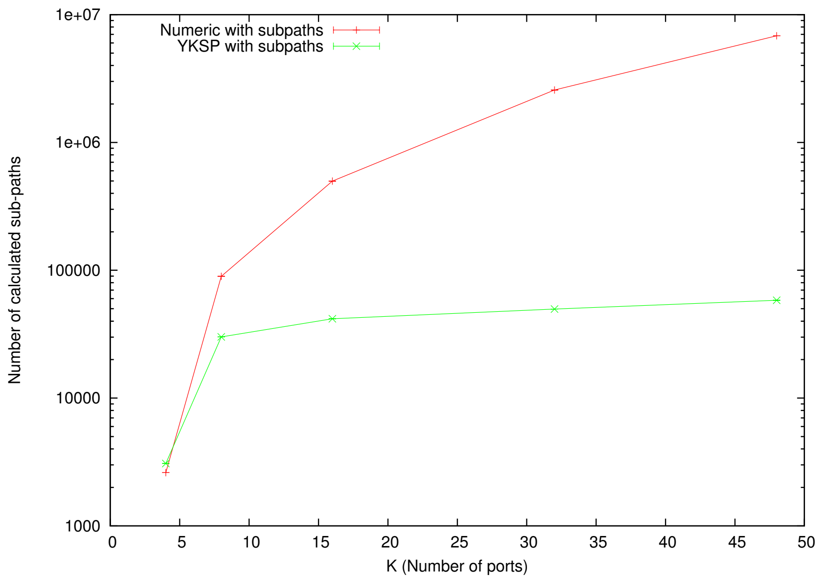


Figure 2.5: The total number of paths generated after finding paths between maximum 2000 host pairs.

6.1 Heuristics and Metaheuristics

A set of heuristic techniques provide each traffic flow with an initial value for the monitoring host and monitoring switch. The intermediary results are further optimized by a few metaheuristics [6, 20]. Metaheuristics provide the opportunity for escaping from local optima. The OptaPlanner [27] engine is used for solving the optimization problem.

Several heuristic techniques are evaluated for initializing the problem, including random selection from the Cartesian product of the monitoring host and monitoring switch sets *(h0)*; choosing monitoring switch and host close to the flow source, while over-provisioning *(h1)* or under-provisioning *(h2)* monitoring hosts; choosing monitoring switch and host close to the flow source or destination, while over-provisioning *(h3)* or under-provisioning *(h4)* monitoring hosts; choosing switches at the core, while over-provisioning *(h5)* or under-provisioning *(h6)* monitoring hosts.

Heuristics are evaluated by comparing solutions' scores (not shown). And, the best one *(h3)* is used for the rest of evaluation.

After the initial search for the feasible solutions, three metaheuristic techniques are employed, including Simulated Annealing [19], Late Acceptance Hill-Climbing [8], and Tabu Search [12].

Each of these metaheuristics has various configuration parameters that determine its effectiveness for a particular problem. The parameters' values are chosen by benchmarking each method with different values (not shown). The following values achieve the best scores.

The Simulated Annealing starting temperature is 200 hard and 1500 soft scores, which is approximately equal to the score difference a single move can make. Late Acceptance has a bin size of 400, which specifies the number previous solutions that are considered when accepting a new candidate. The Tabu size is 7, which determines the number of previously-visited solutions that are avoided in each step.

6.2 Inputs

The monitoring problem is solved for various inputs. Each input represents a problem and determines the infrastructure specifications and traffic characteristics. Input parameters are divided into two groups. The first group of parameters can vary for each problem (Table 2.3). The second group should be common to all problems (Table 2.4), which are supposed to be compared. The infrastructure cost, PS value, and traffic flow rate can vary between inputs. The network topology (i.e. represented by the number of ports in the switch), the traffic flow characteristics (i.e. inter-pod, intra-pod, intra-edge), and the ratio of monitored traffic should be identical in all problems. The restriction is introduced to make the size and solutions of the problems comparable.

	Parameters				
Input	**Host Cost**	**Switch Init Cost**	**Switch Reuse Cost**	**Pod Sensibility**	**Flow Rate (Mbps)**
0	1000	10	0.05	0.1	100
1	1000	10	0.05	1	100
2	1000	10	0.05	1	150
3	1000	10	0.05	1	300
4	1000	10	0.05	10	100
5	10000	10	0.05	1	100
6	10000	10	0.05	1	150
7	10000	10	0.05	1	300

Table 2.3: Parameters that can vary for each input.

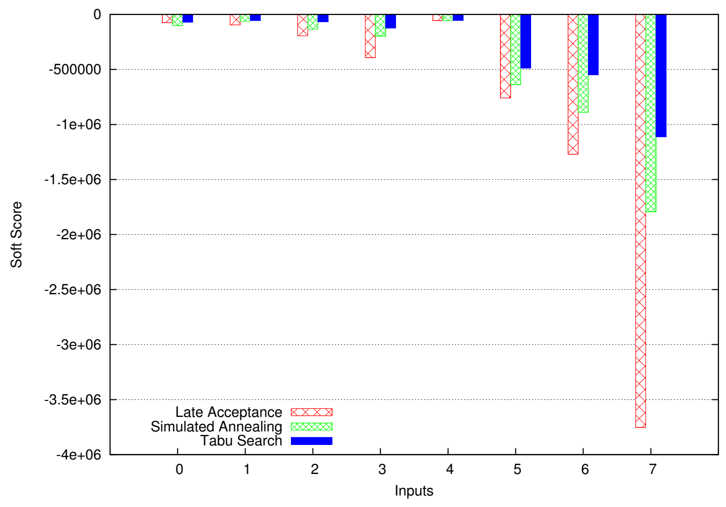


Figure 2.6: Soft score of the best candidate solution for different algorithms.

Parameter	**Value**
# Ports (k)	48
Inter-Pod Probability (p1)	0.000001
Intra-Pod Probability (p2)	0.0001
Intra-Edge Probability (p3)	0.011
Monitored Traffic	10%

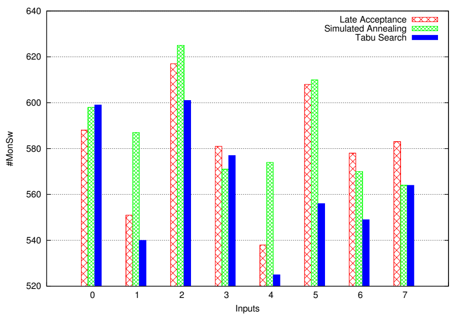
Table 2.4: Parameters that are common in all inputs.

To limit the number of problem instances, the evaluation considers all monitoring hosts to be physical machines with a high initialization cost.

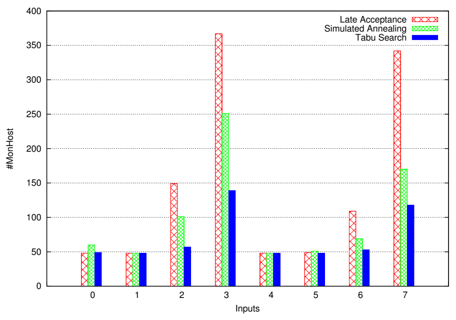
The traffic characteristics are derived from previous empirical studies on data center networks (e.g. [4]). In addition, the evaluation is performed on the model of a large-scale data center with more than 27000 servers and 2800 switches (i.e. k=*48*).

6.3 Scores

Figure 2.6 depicts the soft score of the best candidates for each algorithm. The soft score represents the total overhead of the monitoring solution, which is the sum of extra link utilization, switch cost, and host cost.



(a) The number of configured monitoring switches.



(b) The number of initiated monitoring hosts.

Figure 2.7: Monitoring switches and hosts usage.

6.4 Number of Required Nodes

A candidate solution determines the monitoring switch and host for each traffic flow. Figures 2.7a and 2.7b show the number of switches and hosts for each algorithm. The flow rate has a significant impact on the number of monitoring hosts. The impact is a result of the limited network bandwidth capacity between monitoring switches and a particular monitoring host. In other words, the bottleneck is not the processing capacity of the host, but the available bandwidth to that host.

6.5 Monitoring Switch Distribution

Figure 2.8 depicts the number of switches and the fraction of switches in each layer of the hierarchy. For all inputs and algorithms, the edge switches are utilized more than the switches in core or aggregation. Typically, the aggregation and core switches are in the second and third place. Again, the link capacity between the switches is considered to be the main restricting factor in choosing switch layer. The farther the switch is from the edge, the monitoring traffic overhead on the links is more significant.

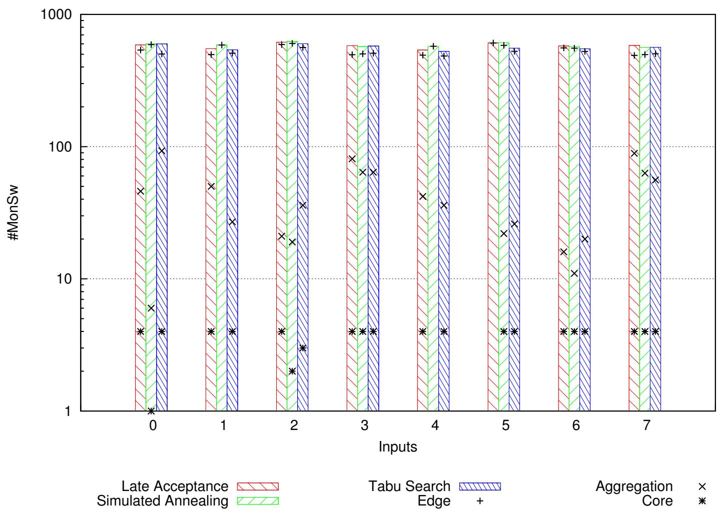


Figure 2.8: The number of monitoring switches and their distribution over the topology hierarchy.

The aggregated usage of a switching layer is the total number of times switches in that layer are used in the monitoring service. Figure 2.9 depicts the switches quantity and the aggregated switch usage grouped by the switching layer (i.e. switching layer usage). It shows that the mean usage of a switch at the edge is higher than other layers.

6.6 Monitoring Node Reuse

Moreover, it is crucial to understand how many times an individual switch or host is reused. Figure 2.10 provides that the switch or host reuse is inversely proportional to the total number of switches or hosts, respectively, and the constant is the number of traffic flows.

6.7 Distance

As explained earlier, the distance of a monitoring switch from a monitoring host determines the impact of monitoring overhead on the network utilization. The distance is depicted in Figure 2.11. The average distance is 3, that means most flows are mirrored and processed in the source or destination pods.

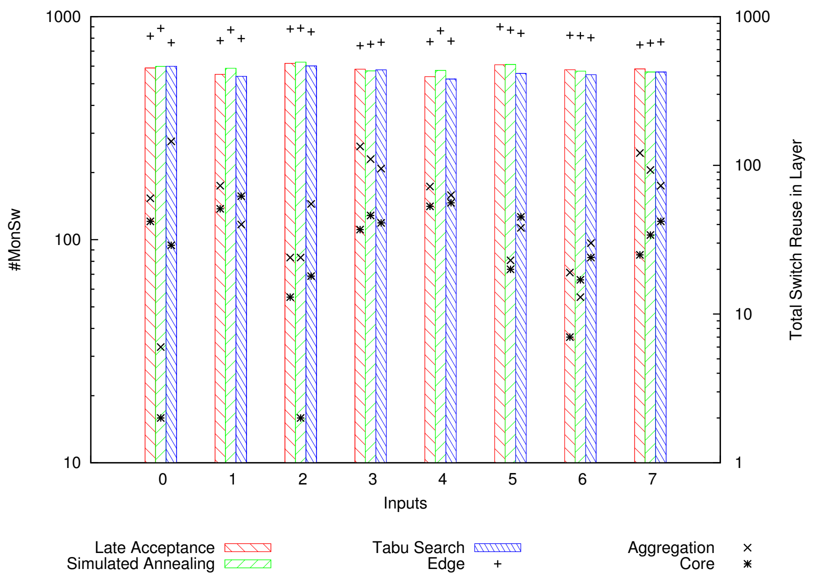


Figure 2.9: The number of monitoring switches and the aggregated usage grouped by the topology layer.

7 Network Programming

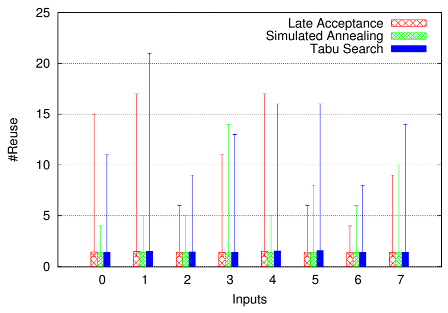
Switches, with the traffic mirroring capability, can send a copy of the traffic to/from a set of ports to the mirroring port. However, a fine-grained (e.g. flow-based) mirroring is challenging using legacy devices.

Given that the switching devices support control and management planes protocols (e.g. OpenFlow and OVSDB), three approaches can be envisioned for flow-based traffic mirroring. Figure 2.12 depicts the paths that each of these approaches proposes.

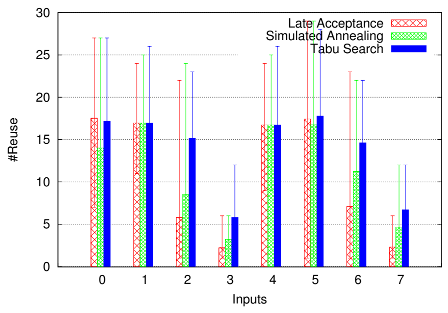
P1- Virtual Remote Flow-Based Filtering (VRFBF)

If the mirroring switch is a virtual one or supports embedded virtual switches (e.g. Pica8 hardware switches), it would be possible to create a virtual switch that is responsible for filtering port-based mirrored traffic. The VRFBF switch is directly connected to the mirroring switch (e.g. using a patch port); and it filters the incoming traffic based on the flow monitoring criteria.

This is a beneficial approach when more control over the monitoring infrastructure is required. For instance, a separated monitoring controller



(a) Monitoring Switch Reuse Stats.



(b) Monitoring Host Reuse Stats.

Figure 2.10: Monitoring Switch and Host reuse statistics.

can control the filtering switches and provides detailed statistics about the framework performance (similar to the [37]). It addition, as opposed to LFBF, it does not need vendor-specific OpenFlow extensions (e.g. resubmit).

P2- Remote Flow-Based Filtering (RFBF)

In a physical switch with limited virtualization capabilities, the mirrored traffic should be forwarded to another switch for further filtering. Although the traffic can be directly sent to the monitoring host, this can introduce a significant load on the network, depending on the distance between the switch and the host. Since the traffic is encapsulated, the intermediate switch (RFBF) decapsulates the traffic, performs the filtering, encapsulates again, and forwards the traffic to the host.

P3- Local Flow-Based Filtering (LFBF)

Local filtering does not use the port-based mirroring function of the switch. Instead, multiple output actions, in the OpenFlow instruction set, forward the traffic to the destination and send a copy toward the monitoring host. Therefore, the flow-based mirroring function can be implemented inside a single switch, whether virtual or physical. The filtering and mirroring logic can be implemented in one or more flow tables, while preserving other functions. Other functions remains intact by partitioning the flow tables to support orthogonal packet processing. These functions can be access control, L2 forwarding, L3 routing, etc.

If the controller does not have a clear partitioning logic, a similar action

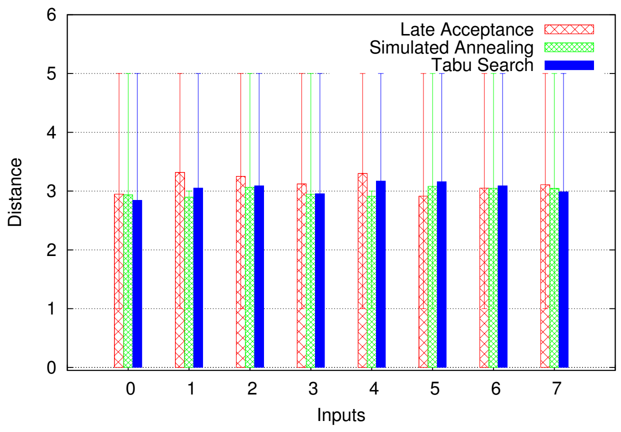


Figure 2.11: The mean distance between monitoring switches and hosts.

to the Open vSwitch [26] *resubmit(port, table)* is required. This action makes a second lookup on a given table, and is not part of the standard OpenFlow protocol.

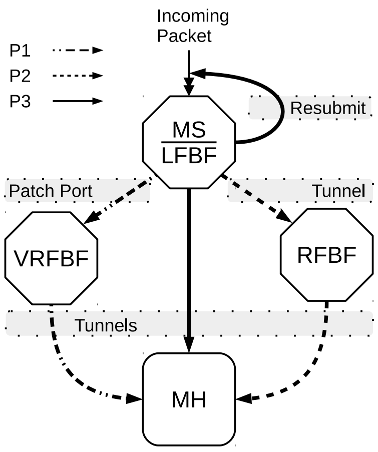


Figure 2.12: Flow-based filtering and delivery approaches for traffic mirroring (MS: Mirroring Switch, MH: Monitoring Host, LFBF: Local Flow-Based Filtering Switch, RFBF: Remote Flow-Based Filtering Switch, VRFBF: Virtual Remote Flow-Based Filtering Switch)

8 Conclusion

Current solutions for data center network monitoring, with the purpose of payload analysis, are inflexible, costly, and not scalable to the trending computing models. This paper proposes a solution that distributes the service across several commodity switches and servers.

The switches make a copy of the designated traffic and the servers perform the processing. Finding the most suitable switch and host, for mirroring and processing a traffic flow, is challenging, specially when the flow size is large. Therefore, the service distribution and coordination logic is formalized as a combinatorial optimization problem.

The problem size demands for efficient algorithms. Consequently, a shortest path algorithm for Fat-Tree networks is introduced that is significantly faster than traditional graph algorithms.

Typical switches have no or limited supports for flow mirroring, as opposed to port mirroring. This practical limitation is rectified by introducing a reliable and vendor-agnostic approach for implementing the flow mirroring logic.

The results indicate that monitoring 10% of traffic in a large-scale data center, with more than 27000 hosts, can be achieved by deploying 0.5% of hosts as monitoring hosts and configuring 20% of switches as monitoring switches.

Moreover, the proposed shortest path algorithm for this topology is two to three order of magnitude faster than the previous algorithms.

References

[1] Niels L. M. van Adrichem, Christian Doerr, and Fernando A. Kuipers. "OpenNetMon: Network monitoring in OpenFlow Software-Defined Networks." In: *Network Operations and Management Symposium, 2014 IEEE*. NOMS'14. May 2014, pp. 1–8.

[2] Gianni Antichi et al. "OSNT: open source network tester." In: *IEEE Network* 28.5 (Sept. 2014), pp. 6–12.

[3] Jeffrey R. Ballard, Ian Rae, and Aditya Akella. "Extensible and scalable network monitoring using OpenSAFE." In: *Proceedings of the 2010 internet network management conference on Research on enterprise networking*. INM/WREN'10. 2010, pp. 8–8.

[4] Theophilus Benson, Aditya Akella, and David A. Maltz. “Network traffic characteristics of data centers in the wild.” In: *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*. IMC’10. 2010, p. 267.

[5] Theophilus Benson et al. “Understanding data center traffic characteristics.” In: *ACM SIGCOMM Computer Communication Review* 40.1 (Jan. 2010), p. 92.

[6] Christian Blum and Andrea Roli. “Metaheuristics in combinatorial optimization: Overview and conceptual comparison.” In: *ACM Computing Surveys* 35.3 (Sept. 2003), pp. 268–308.

[7] Anat Bremler-Barr et al. “Deep Packet Inspection As a Service.” In: *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*. CoNEXT’14. 2014, pp. 271–282.

[8] Edmund K Burke and Yuri Bykov. “A late acceptance strategy in hill-climbing for exam timetabling problems.” In: *In Proceedings of the conference on the Practice and Theory of Automated Timetabling*. PATAT’08. 2008.

[9] Chuck Cranor et al. “Gigascope: A Stream Database for Network Applications.” In: *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*. SIGMOD’03. 2003, pp. 647–651.

[10] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. “A scalable, commodity data center network architecture.” In: *ACM SIGCOMM Computer Communication Review* 38.4 (Oct. 2008), p. 63.

[11] Aaron Gember-Jacobson et al. “OpenNF: enabling innovation in network function control.” In: *Proceedings of the 2014 ACM Conference on SIGCOMM*. SIGCOMM’14. 2014, pp. 163–174.

[12] Fred Glover. “Tabu Search—Part I.” In: *ORSA Journal on Computing* 1.3 (Aug. 1989), pp. 190–206.

[13] J. Halpern and C. Pignataro. *Service Function Chaining (SFC) Architecture*. (Informational). Internet Engineering Task Force, May 2015. URL: `https://www.ietf.org/id/draft-ietf-sfc-architecture-08.txt`.

[14] Peer Hasselmeyer and Nico d'Heureuse. "Towards holistic multi-tenant monitoring for virtual data centers." In: *Network Operations and Management Symposium Workshops, 2010 IEEE/IFIP*. NOMS'10. 2010, pp. 350–356.

[15] Brandon Heller et al. "ElasticTree: Saving Energy in Data Center Networks." In: *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*. NSDI'10. 2010, pp. 17–17.

[16] Evan Hoke, Jimeng Sun, and Christos Faloutsos. "InteMon: Intelligent System Monitoring on Large Clusters." In: *Proceedings of the 32Nd International Conference on Very Large Data Bases*. VLDB'06. Seoul, Korea, 2006, pp. 1239–1242.

[17] Srikanth Kandula et al. "The Nature of Data Center Traffic: Measurements & Analysis." In: *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*. IMC'09. 2009, pp. 202–208.

[18] A. Kind et al. "Advanced network monitoring brings life to the awareness plane." In: *IEEE Communications Magazine* 46.10 (Oct. 2008), pp. 140–146.

[19] Scott Kirkpatrick, C Daniel Gelatt, Mario P Vecchi, et al. "Optimization by simulated annealing." In: *Science* 220.4598 (1983), pp. 671–680.

[20] Sean Luke. *Essentials of Metaheuristics*. second. Lulu, 2013.

[21] M. Mahalingam et al. *Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks*. RFC 7348 (Informational). Internet Engineering Task Force, Aug. 2014. URL: `http://www.ietf.org/rfc/rfc7348.txt`.

[22] Santosh Mahapatra, Xin Yuan, and Wickus Nienaber. "Limited Multi-path Routing on Extended Generalized Fat-trees." In: *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International*. IPDPSW'12. May 2012, pp. 938–945.

[23] Nick McKeown et al. "OpenFlow: enabling innovation in campus networks." In: *ACM SIGCOMM Computer Communication Review* 38.2 (Mar. 2008), p. 69.

[24] Peter Mell and Timothy Grance. *The NIST Definition of Cloud Computing*. Tech. rep. SP 800-145. National Institute of Standards and Technology, Information Technology Laboratory, Sept. 2011.

[25] Yogesh Mundada, Anirudh Ramachandran, and Nick Feamster. "Silverline: Data and network isolation for cloud services." In: HotCloud'11. 2011.

[26] *Open vSwitch*. URL: `openvswitch.org`.

[27] *OptaPlanner*. URL: `http://www.optaplanner.org/`.

[28] Bogdan Prisacari et al. "Fast pattern-specific routing for fat tree networks." In: *ACM Transactions on Architecture and Code Optimization* 10.4 (Dec. 2013), pp. 1–25.

[29] P. Quinn and T. Nadeau. *Problem Statement for Service Function Chaining*. RFC 7498 (Informational). Internet Engineering Task Force, Apr. 2015. URL: `http://www.ietf.org/rfc/rfc7498.txt`.

[30] Paul Quinn and Uri Elzur. *Network Service Header*. Internet Draft. Internet Engineering Task Force, Mar. 2015. URL: `https://datatracker.ietf.org/doc/draft-ietf-sfc-nsh/`.

[31] J. Rajahalme et al. *IPv6 Flow Label Specification*. RFC 3697 (Proposed Standard). Obsoleted by RFC 6437. Internet Engineering Task Force, Mar. 2004. URL: `http://www.ietf.org/rfc/rfc3697.txt`.

[32] Jeff Rasley et al. "Planck: millisecond-scale monitoring and control for commodity networks." In: *Proceedings of the 2014 ACM Conference on SIGCOMM*. SIGCOMM'14. 2014, pp. 407–418.

[33] German Rodriguez et al. "Oblivious routing schemes in extended generalized Fat Tree networks." In: *Cluster Computing and Workshops, 2009. IEEE International Conference on*. CLUSTER'09. 2009, pp. 1–8.

[34] Justine Sherry et al. "Making middleboxes someone else's problem: network processing as a cloud service." In: *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. SIGCOMM'12. 2012, p. 13.

[35] Alan Shieh et al. "Seawall: Performance Isolation for Cloud Datacenter Networks." In: *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*. HotCloud'10. 2010, pp. 1–1.

[36] Sajad Shirali-Shahreza and Yashar Ganjali. “Efficient Implementation of Security Applications in OpenFlow Controller with FleXam.” In: *High-Performance Interconnects, 2013 IEEE 21st Annual Symposium on.* HOTI’13. Aug. 2013, pp. 49–54.

[37] Aryan TaheriMonfared and Chunming Rong. “Flexible Building Blocks for Software Defined Network Function Virtualization.” In: *2014 IEEE 10th International Conference on Quality, Reliability, Security and Robustness in Heterogeneous Networks (QShine).* Aug. 2014, pp. 37–43.

[38] Aryan TaheriMonfared and Chunming Rong. “Virtual Network Flavors: Differentiated Traffic Forwarding for Cloud Tenants.” In: *Wired & Wireless Internet Communications: The 13th International Conference on.* WWIC’15. 2015.

[39] Aryan Taherimonfared, Tomasz Wiktor Wlodarczyk, and Chunming Rong. “Real-Time Handling of Network Monitoring Data Using a Data-Intensive Framework.” In: *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on.* Dec. 2013, pp. 258–265.

[40] Amin Tootoonchian, Monia Ghobadi, and Yashar Ganjali. “OpenTM: traffic matrix estimator for OpenFlow networks.” In: *Proceedings of the 11th international conference on Passive and active measurement.* PAM’10. 2010, pp. 201–210.

[41] Jin Y Yen. “Finding the k shortest loopless paths in a network.” In: *Management Science* 17.11 (1971), pp. 712–716.