Architectures for Virtualization and Performance Evaluation in Software Defined Networks

Von der Fakultät für Elektrotechnik und Informatik
der Gottfried Wilhelm Leibniz Universität Hannover
zur Erlangung des akademischen Grades

Doktor-Ingenieur

genehmigte

Dissertation

von

Dipl.-Ing. Zdravko Bozakov
geboren am 22. März 1980 in Sofia

2016

1. Referent	:	Prof. Dr.-Ing. Markus Fidler
2. Referent	:	Prof. Dr.-Ing. Ralf Steinmetz
Tag der Promotion	:	26. August 2015

ABSTRACT

Within half a century the Internet has transformed from a small scale research project to a globally distributed communications system which permeates all facets of modern life. A crucial factor for the success of the Internet is its simple yet extremely versatile architecture which has repeatedly adapted to changing demands. This flexibility can be attributed to the widespread use of suitable abstractions which reduce complexity and open interfaces. Well known abstractions are the end-to-end principle, which decouples the complexity of the Internet topology from applications deployed at the end hosts, as well as the Internet layer model which enables an independent development and deployment of physical, network and application layer technologies, among others. The concept of virtualization is another abstraction which has been successfully employed to facilitate the transition between legacy and emerging network technologies. Today virtualization plays a indispensable role for supporting elasticity and scalability in data center networks and is viewed as an essential tool for combating the ossification of the Internet architecture.

To ensure that networks remain well equipped to support future network services with highly variable demands, the research community has proposed the software defined networking (SDN) paradigm which decouples the network control logic from the underlying physical devices. The network forwarding plane is viewed as a collection of programmable, interconnected nodes managed by a logically centralized network controller which maintains a global view of the network state. Network applications use this centralized abstraction of the network substrate to deploy their control logic. It is expected, that the increased level of abstraction will enable the development of systematic methods for deploying network services and verifying network functionality.

In this thesis we develop concepts and mechanisms required to enable the virtualization of SDNs. The provision of isolated, programmable network slices enables new business models for infrastructure providers and virtual network operators. In addition, we derive techniques for augmenting the global network view with statistical metrics extracted from network measurements. Such metrics are essential for enabling effective traffic engineering in SDNs. The thesis is structured in two parts according to these objectives.

First, we provide an analysis of the requirements for virtual software defined network topologies and use the identified constraints to derive SDN virtualization strategies. We propose a general architecture for SDN virtualization which supports the instantiation of arbitrary, programmable virtual topologies. The architecture employs a scalable network hypervisor to conceal the virtualization mechanisms from the vir-

tual network tenants and to provide full isolation of the virtual resources. We establish that optimization and resilience mechanisms in virtual software defined networks (vSDNs) with arbitrary topologies must be handled solely by the tenant. Further, we show that by offloading standard topology objectives, such as resilience or elasticity, to the physical domain the complexity of virtual network topologies is significantly reduced. Thus, we outline a virtual network abstraction which provides a connectivity service between multiple tenant points of presence, defined by demands at the end-points. We analyze the embedding costs for such services and show that significant gains may be achieved by obtaining an accurate characterization of the tenant traffic requirements. We propose a virtual router architecture as an example of a layer 3 connectivity service. The platform combines open software and off-the-shelf hardware to offer tenants virtual router instances which are functionally and logically indistinguishable from a traditional router.

The allocation of resources in a multi-tenant environment motivates the second part of the thesis where we focus on the extraction of traffic characteristics and the derivation of quality of service (QoS) parameters. As Internet traffic is known to posess long-memory, over-provisioning is an inevitable strategy for maintaining reliable performance in computer networks. As a consequence, a tight characterization of the carried network traffic is essential for dimensioning resource reservations in the network substrate. To this end, we evaluate approaches for extracting flow-level traffic correlations from network observations, obtained from packet samples or switch counter queries. We use random sampling to reduce the monitoring load while maintaining a high resolution of the estimate, analyze the impact of the sampling strategies on the estimates and derive methods for reversing sampling distortions. An analytical evaluation quantifies the effects of finite sampling durations on the autocovariance estimators. The autocovariance estimates are used to perform Monte Carlo simulations with the goal of evaluating the queue length distribution for the observed flow for arbitrary capacity assignments. To this end, we outline approaches for synthesizing independent sample paths which exhibit the same correlation structure as the observed traffic flow and also match its increment distribution under certain conditions. We demonstrate that the use of random sampling does not negatively affect the obtained estimate of the queue length distribution.

Software-Defined Networking · Virtualization · Performance Evaluation

ZUSAMMENFASSUNG

Innerhalb eines halben Jahrhunderts hat sich das Internet von einem kleinen Forschungsprojekt in ein globales Kommunikationssystem entwickelt, welches alle Facetten des Lebens durchdringt. Ein wesentlicher Faktor, der zum Erfolg des Internets beigetragen hat, ist seine einfache und zugleich extrem flexible Architektur, die sich wiederholt an verschiedene Anforderungen angepasst hat. Diese Flexibilität liegt an der Nutzung von offenen Schnittstellen sowie an den umfassend verwendeten Abstraktionen, die wesentlich zur Reduktion der Komplexität beitragen. Bekannte Abstraktionen im Internet sind das Ende-zu-Ende-Prinzip, welches die Komplexität der Internettopologie von den Applikationen auf Endgeräte entkoppelt, sowie das Schichten-Modell, das die unabhängige Entwicklung und den Einsatz von Technologien beispielsweise in den Physikalischen-, Netzwerk- und Applikationsschichten ermöglicht. Das Konzept der Virtualisierung stellt eine weitere Netzwerkabstraktion dar, welche erfolgreich eingesetzt wurde, um einen Übergang zwischen Legacy- und neuen Netzwerktechnologien zu ermöglichen. Heute ist Virtualisierung unabdingbar, um Skalierbarkeit und Elastizität in Datenzentren zu gewährleisten. Darüber hinaus wird Virtualisierung als Schlüsseltechnologie zur Bekämpfung der sogenannten "Ossifizierung" der Internetarchitektur gesehen.

Um sicherzustellen, dass Computernetzwerke auch zukünftig eine Vielzahl von Services mit unterschiedlichen Anforderungen unterstützen können, hat die Forschungsgemeinschaft das Software-Defined Networking (SDN) Paradigma konzipiert. Dieser Ansatz entkoppelt die Kontrolllogik eines Netzwerks von der physikalischen Infrastruktur. Dadurch wird der Datenpfad als eine Sammlung von programmierbaren Netzknoten abstrahiert, welche durch einen logisch zentralisierten Controller gemanagt werden. Der Controller erstellt dabei fortdauernd eine globale Sicht des Netzwerks und stellt diese Netzwerkapplikationen zur Verfügung. Es wird erwartet, dass durch die sich daraus ergebende höhere Abstraktionsebene neue systematische Ansätze für die Einführung von Netzwerkdiensten sowie die Verifikation von Netzwerkfunktionalitäten ermöglicht werden.

Ziel dieser Arbeit ist die Entwicklung von Konzepten und Mechanismen, welche die Virtualisierung von SDNs ermöglichen. Die Bereitstellung von isolierten, programmierbaren virtuellen Netzwerken eröffnet neuartige Geschäftsmodelle sowohl für Netz-Provider als auch für Betreiber von Diensten auf Basis von virtuellen Netzen. Des Weiteren werden messbasierte Methoden hergeleitet, welche die globale Netzwerksicht um statische Metriken ergänzen. Solche Metriken sind essentiell, um Traffic-Engineering in SDNs zu ermöglichen. Abgeleitet aus diesen Zielen ist die Dissertation in zwei Abschnitte unterteilt.

Zunächst wird eine Analyse der Anforderungen für virtuelle SDN-Topologien durchgeführt. Aus den identifizierten Einschränkungen werden zwei SDN-Virtualisierungsstrategien hergeleitet: Erstens wird eine allgemeine Virtualisierungsarchitektur entwickelt, welche die Instantiierung von beliebigen, programmierbaren virtuellen Topologien erlaubt. Diese Architektur verwendet einen skalierbaren Hypervisor, um die eingesetzten Virtualisierungsmechanismen von den vSDN-Betreibern (Kunden) zu verbergen und somit eine Isolierung der virtuellen Ressourcen zu gewährleisten. Es wird gezeigt, dass in einer solchen Architektur Optimierungs- und Resilienzanforderungen von den Kunden implementiert werden müssen. Des Weiteren wird gezeigt, dass die Komplexität von virtuellen SDN-Topologien durch eine Auslagerung von Funktionen wie Elastizität und Redundanz in die physikalische Domäne deutlich reduziert wird. Daher wird zweitens eine vereinfachte Abstraktion für virtuelle Netze vorgestellt, die einen Konnektivitätsservice zwischen mehreren Kundenstandorten (PoPs) zur Verfügung stellt. Dieser Service ist durch die Anforderungen an den Netzwerk-Endpunkten definiert. Für dieses Szenario wird eine Analyse der Embedding-Kosten durchgeführt. Es wird gezeigt, dass diese durch eine genaue Charakterisierung der Anforderungen von Kundenverkehr signifikant reduziert werden können. Als Beispiel für einen Layer 3-Konnektivitätsservice wird die Implementierung einer Architektur für virtuelle Router vorgestellt. Diese Plattform kombiniert offene Software und kommerzielle Hardware, um Kunden virtuelle Routerinstanzen zur Verfügung zu stellen, die sich logisch und funktional nicht von physikalischen Routern unterscheiden.

Die Allokation von Ressourcen in Multi-Tenant-Umgebungen motiviert den zweiten Teil der Arbeit, in dem ein Schwerpunkt auf die Extraktion von Eigenschaften des Internet-Verkehrs und auf die Herleitung von Quality-of-Service (QoS) Parametern gesetzt wird. Eine bekannte Eigenschaft von Netzwerkverkehr ist, dass dieser über lange Zeitabschnitte eine starke Autokorrelation aufweist. Dieses als „Long-Memory" bezeichnete Merkmal hat negative Auswirkungen auf die Netzwerkperformanz und erfordert ein Over-Provisioning der Netzwerkressourcen, um ein zuverlässiges Serviceniveau im Netzwerk zu gewährleisten. Daher ist es im Kontext der Netzwerkvirtualisierung besonders wichtig, die im Netzwerk übertragenen Flows genau zu charakterisieren, um vorhandene Netzwerkkapazitäten effizient zuzuweisen. In dieser Arbeit werden Methoden entwickelt, welche die Autokorrelationen von einzelnen Traffic-Flows aus Beobachtungen von Datenzählern in Switches oder aus Paketsamples extrahieren. Dabei werden Zufallsstichproben eingesetzt, um die Monitoring-Last zu minimieren und gleichzeitig eine hoch-aufgelöste Abschätzung der Korrelationsstruktur zu gewährleisten. Weiterhin wird der Einfluss von Samplingstrategien auf die Schätzung evaluiert, und es werden Methoden hergeleitet, welche Sampling-Verzerrungen ausgleichen. Eine analytische Evaluation quantifiziert die Effekte von zeitlich begrenzten Messungen auf die Korrelationsstruktur

und zeigt Möglichkeiten auf, diese Effekte unter bestimmten Umständen zu eliminieren. Die extrahierte Korrelationsinformation wird verwendet, um mit Hilfe von Monte Carlo-Simulationen, die Verteilung der Puffergröße abzuschätzen, die sich für einen betrachteten Flow bei einer bestimmten Kapazitätszuweisung ergibt. Dazu werden Ansätze für die Generierung von statistisch unabhängigen Prozessen präsentiert, die die gleichen Korrelationseigenschaften wie der observierte Verkehrsflow aufweisen. Zusätzlich, wird unter bestimmten Bedingungen auch die Verteilung der Inkremente abgebildet. Es wird gezeigt, dass der Einsatz von Zufallsstichproben keine negativen Auswirkungen auf die Schätzung der Puffergröße mit sich bringen.

CONTENTS

LIST OF FIGURES

ACRONYMS

ARFIMA autoregressive fractionally integrated moving average

ASTA arrivals see time averages

BGP border gateway protocol

CBR constant bit rate

CCDF complementary cumulative distribution function

CDF cumulative distribution function

CLT central limit theorem

CPX controller proxy

DiffServ differentiated services

DoS denial of service

EF expedited forwarding

FCM flow cache manager

FMP flow management proxy

FMP flow management proxy

IP Internet protocol

ISP Internet service provider

IntServ integrated services

InP infrastructure provider

LAN local area network

LRD long range dependency

MBAC measurement based admission control

MCF minimum cost flow

MC Monte Carlo

NIMASTA non-intrusive mixing arrivals see time averages

NUMA non-uniform memory access

OF OpenFlow

OS operating system

PASTA Poisson arrivals see time averages

PD positive definite

PHB per-hop behavior

PSTN public switched telephone network

PoP point of presence

RMM resource manager module

RWA routing and wavelength assignment

SDN software defined networking

SLA service level agreement

SSP successive shortest paths

TCAM ternary content-addressable memory

TEG tenant edge gateway

TUID tunnel identifier

VE virtual environment

VLAN virtual local area network

VMM virtual machine monitor

VM virtual machine

VNO virtual network operator

VN virtual network

VPN virtual private network

VRC virtual router controller

VRS virtual router service

VRaaS virtual routers as a service

vSDN virtual software defined network

VTID virtual flow table identifier

XID context identifier

fBm fractional Brownian motion

fGn fractional Gaussian noise

i.i.d. independent and identically distributed

PoP point of presence

pps packets per second

QoS quality of service

Part I

DISSERTATION

1 INTRODUCTION

The Internet permeates all facets of modern life. Within half a century it has undergone a phenomenal transition from a small scale DARPA research project, the ARPANet, to a globally distributed system, providing the underpinnings of today's society, economy and information exchange. The success of the network is based largely on a simple and elegant design which has enabled a continuous adaption to ever changing demands. Central to its design is the frequent use of abstractions. In its most simple abstraction the Internet acts as a single system which provides best effort, end-to-end connectivity between end-hosts over arbitrarily interconnected nodes. Hiding internal complexity from end-hosts, while exposing a minimal set of interfaces greatly simplifies the deployment of diverse services at the end hosts. Similarly, the Internet layer model abstracts network functions and, for instance, enables physical layer technologies to be developed independently from transport layer mechanisms. Identifying the most suitable practical abstractions and continuously searching for new ones to solve emerging challenges has been a major strength of the Internet's design and a most significant contribution of the research community.

The concept of virtualization, i.e. the abstraction of physical resources, such as link capacity, memory space, storage and processing units, into purely logical entities, is another widely used type of abstraction employed throughout the Internet architecture. Originally developed as a means for resource sharing in mainframe computers, numerous forms of virtualization have been employed in the networking domain. While the term *virtualization* has only recently come into ubiquitous use, historically the virtualization paradigm is deeply ingrained into the Internet architecture. It has played a key role in maintaining flexibility in the network architecture and facilitating the transition between legacy and emerging technologies.

Eliminating the coupling between virtual and physical resources enables a remapping of the virtual entities to new physical resources without the need for altering logical functionality. This concept was exploited as a migration path during the early deployment stages of the Internet, where connectivity between Internet nodes was implemented as an overlay on top of the public switched telephone network (PSTN). As network technology matured and became financially feasible connections were migrated to dedicated network links. Virtualization techniques are regarded as a means to combat the ossification of the Internet architecture in the future.

Furthermore, the virtual resource abstraction enables a concurrent operation of virtual entities on a common physical substrate. As a result,

physical components spend less time idling, leading to a better utilization of the available infrastructure. A prime example of concurrency in the Internet architecture, is the use of packet switching as a fundamental mechanism. In the context of virtualization, packet switching may be regarded as a mapping of multiple circuits to a single physical connection, thereby providing significant multiplexing gains compared to traditional circuit switched architectures.

An additional benefit of virtualization, is the reduction of complexity and the mitigation of scalability issues. This is achieved by segmenting large systems into smaller, more easily manageable logical entities. This strategy is typically employed in large local area networks (LANs) which are segmented into virtual local area networks (VLANs) to enforce administrative structure and policies. As a result, the control of each VLAN may be delegated to a different administrative domain. Moreover, limiting the size of network segments mitigates the impact of broadcast storms which cause network congestion.

In the last decade server virtualization technology has matured significantly and has seen almost universal adoption in the data center domain. The virtualization of computing and storage hardware has lead to significant reductions of operational costs by improving the utilization of available physical resources. Moreover, the use of virtualization in the data center offers the notion of elasticity, i.e., the ability to quickly adapt to changing workloads by instantiating (or destroying) virtual machines (VMs) and load-balancing resources on-the-fly. As a result, operators have been able to build efficient, large scale data centers, while offering on-demand resources at a low cost. In turn, the novel virtualized data center infrastructure has spurred a wave of innovative business models and services - colloquially referred to as cloud technologies - which offload hardware hosting to a third party.

With the rapid developments in data centers it became evident that the rate of innovation in networking domain is not sufficient to fulfill the varied requirements in highly virtualized computing environments. For instance, the migration of VMs between physical nodes requires accurately timed redirection of associated traffic flows. Similarly, traditional routing and forwarding approaches may limit the scalability of distributed applications running on virtualized clusters.

Today, the control logic of general purpose network equipment is tightly integrated into the forwarding devices and is almost entirely under the control of hardware vendors. This makes the addition of custom, application specific network functionality extremely difficult, as the process involves a time consuming design, implementation and testing loop. For small scale projects the incentive for device vendors to implement specific functionality is practically non-existent.

Over the last decade, the research community has worked towards revising the traditional network architecture by evaluating both evolutionary and clean-slate architectures. This development culminated in SDN paradigm which aims to alleviate the aforementioned issues. The

basic idea of SDN is the separation of the network forwarding and control planes through well-defined, open interfaces. Specifically, in SDN the control functionality is outsourced to a centralized *controller* entity. The data plane consists of a collection of externally programmable forwarding elements, with arbitrary interconnections. The first widely supported SDN technology is OpenFlow. The main benefit of this approach is that it introduces a new interface for dynamically controlling network elements as well as a new level of abstraction for approaching network problems. These changes are expected to speed up the innovation cycle.

1.1 THESIS CONTRIBUTIONS

The aim of this work is to further the understanding of the abstractions required to enable virtualization in software defined networking (SDN). In addition, our goal is the development of techniques which enrich the network view with statistical metrics, allowing SDN applications to make better use of the available network resources. This thesis is structured in two main parts according to these objectives. Our key contributions are summarized in the following. We provide an analysis of the requirements for virtual networks and identify fundamental constraints for virtual network topologies which stem from the corresponding application scenario. From these requirements, we derive a general architecture for SDN virtualization which supports the instantiation of arbitrary, programmable virtual topologies. We propose a versatile classification scheme for associating physical resources with a specific virtual context. As a consequence the virtualization layer may be concealed from the virtual network tenants. In addition, we outline a virtual network abstraction which provides a connectivity service between multiple tenant points of presence (PoPs), defined by demands at the end-points. We show that the embedding cost of such connectivity services may be significantly optimized by obtaining a tight characterization of the tenant traffic requirements.

Motivated by these findings, in the second part of the thesis we develop mechanisms for extracting QoS metrics from passive network observations. The proposed methodologies may be integrated into a logically centralized controller framework in order to provide a comprehensive global view of the network state. Such a representation is a fundamental element of the SDN paradigm, enabling operators and SDN applications to optimally utilize the available resources. Given the bursty nature of Internet traffic we consider the characterization of the traffic correlation structure as a key metric for the derivation of QoS bounds. In this thesis we provide an analytical evaluation of random sampling approaches for extracting the autocovariance of individual flows from network observations. To this end, we consider both random packet sampling as well as randomized queries of switch counters. We demonstrate that the use of such sampling approaches enables fine grained estimates of the traffic autocorrelation structure while reducing the monitoring

overhead. We prove that distortions caused by the utilized sampling strategy may be reversed to obtain unbiased estimates of the flow autocorrelation and quantify the effects of finite sampling durations. Finally, we present techniques which exploit the derived estimates to extract QoS bounds for specific switch interfaces.

1.2 THESIS STRUCTURE

In Chapter 2 we review the related work on virtualization, SDN and performance evaluation. We present the research challenges addressed in this work and highlight the significance of the presented contributions in Chapter 3.

In Chapter 4 we identify the requirements towards virtual networks and evaluate the feasibility of virtual network topologies. Next, we define mechanisms for mapping virtual entities to physical resources. We use these findings to design a scalable platform for SDN virtualization, which enables the instantiation of arbitrary virtual SDN topologies with live-migration support. As a result, network tenants may deploy arbitrary, isolated SDN applications on top of a virtualized SDN substrate. Finally, we propose connectivity service abstraction which we exemplify through a virtual router architecture which enables infrastructure operators to offer layer 3 connectivity between tenant PoPs as a service.

In Chapter 5 we provide mechanism for performance evaluation in SDN environments. To this end, we evaluate how the global network view in SDNs may be augmented with QoS metrics. We propose approaches for the extraction of flow-level traffic autocorrelations from packet samples as well as switch counter observations. These metrics are processed by an SDN controller to derive stochastic bounds on the backlog and delay at salient points in the network. The focus of this chapter lies on the use of random sampling as a means for minimizing the monitoring overhead without sacrificing the fidelity of QoS estimates. We quantify the effects of the sampling strategies and demonstrate how these may be reversed. Further, we quantify the impact of LRD traffic on the autocovariance estimators.

We conclude the thesis and discuss future research directions in Chapter 6.

2 BACKGROUND AND RELATED WORK

Today virtualization is ubiquitously employed across a wide range of domains. Further, the scale over which the virtualization abstraction is encountered varies significantly: from virtual memory abstraction within individual computer systems, to virtual machines hosted on shared hardware, to entire data centers which process data using a highly distributed pool of computing and networking resources. The focus of this thesis lies on the virtualization of network infrastructure. We seek to identify suitable network abstractions and to develop a virtualization architecture which efficiently makes use of the available network resources while supporting a wide range of use-cases. The optimization of the resource utilization in a multi-tenant environment necessitates mechanisms for generating a detailed and up-to-date view of the system state.

In this section we provide an overview of the state of the art and related work in the areas of network virtualization, resource allocation and network monitoring. We begin with an overview of the emerging SDN paradigm and associated virtualization frameworks. Next, we provide a review of the related work in network abstractions and network embedding approaches. Finally, we discuss the related work in the fields of admission control, performance evaluation, network measurements and sampling.

2.1 SOFTWARE DEFINED NETWORKING

In the last years the SDN architecture has been proposed as a means for facilitating the deployment of network functions and services. Essentially the approach relies on a clear, network-wide separation between the network control plane and the data plane. An SDN infrastructure consists of a pool of interconnected, programmable forwarding devices. By itself each device contains only the minimal amount of control logic, necessary to receive configuration commands from an external entity. The SDN control plane is implemented as a separate, logically centralized entity, e.g., running on a dedicated server (or a cluster of servers). Each substrate switch is connected to the control plane layer over a well-defined southbound API (e.g., OpenFlow). The SDN control plane (or SDN controller) is responsible for calculating and deploying the application specific forwarding logic across all forwarding devices in the substrate. In addition, it performs topology discovery and maintains a consistent global view of the network state. SDN applications implementing network services are deployed on top of the controller layer using an appropriate northbound interface. As a result of the global network view the need for complex distributed algorithms is eliminated. As an exam-

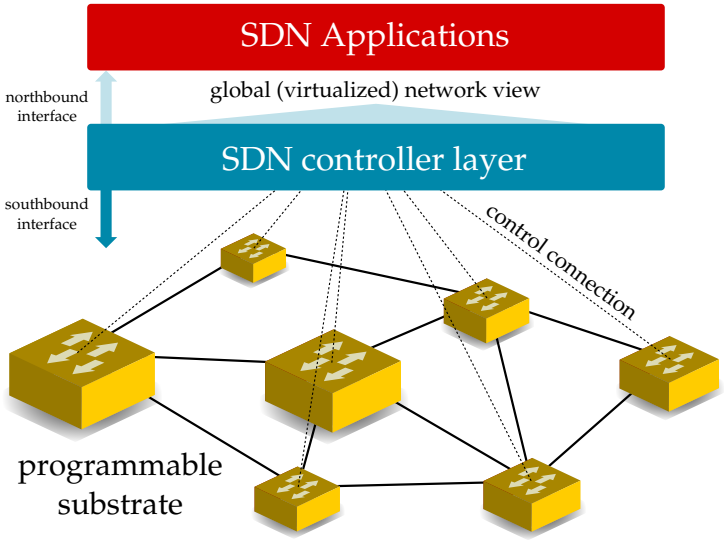


Figure 1: SDN architecture.

ple, a network application may use Dijkstra's algorithm to calculate the shortest paths in the network rather than relying on the distance vector algorithm.

The SDN paradigm promises a higher level of abstraction which facilitates the development and deployment of network services by providing programmability over the network resources, while hiding the underlying complexity of the network devices. The SDN framework enables novel approaches, such as automated verification of network policies [52, 7], or the development of high-level languages for programming SDN infrastructures [58, 118].

The development of SDN based architectures is still in an early stage. A large number of active research areas exist, which address challenges such as ensuring the consistency of the network state [86] and the implementation of distributed SDN controllers [41]. A number of works address scalability issues in SDN platforms [154, 101].

Currently, no predominant northbound interface has emerged. For the southbound interface OpenFlow (OF) [99], which we describe in the sequel, is the most widely supported and frequently used API. However, alternative interfaces such as ForCES [43], or PCEP [143] are also feasible.

OPENFLOW Today, the most widespread southbound interface in SDN is OpenFlow. OpenFlow is an open specification which enables a flow-level configuration of the forwarding logic in commercial switches through an external controller. Each OF-enabled switch maintains a flow table which is populated over a secure control connection. The OpenFlow specification defines a corresponding control protocol. The implementation specifics of the switch flow table are hidden from the controller, i.e., vendors may use ternary content-addressable memorys (TCAMs) as

well as arbitrary, custom memory designs without running risk of leaking intellectual property. Each flow table entry consists of a match *rule* which is associated with a list of packet processing *actions* and, optionally, a data counter. Each rule defines a traffic flow based on a bit pattern across one or more of the following packet header fields:

OpenFlow header fields									
input port	Ethernet source	Ethernet dest.	Ethernet type	VLAN ID	IP source	IP dest.	IP protocol	source port	dest. port

These fields are supported by all devices implementing version 1.0 of the OF specification. When using higher OpenFlow versions flows may match additional fields, such as MPLS or IPv6, as well as optional vendor specific headers. Unspecified header values act as wildcards (often depicted as '*'), matching any bit value at the corresponding location. The flow actions determine how a matched packet should be forwarded (e.g., drop, to port X, to controller) and which processing operations should be applied (e.g., add or strip headers, rewrite header fields). Each flow entry is assigned a priority to ensure a deterministic forwarding behavior for packets matching multiple (wildcarded) rules. Depending on the switch configuration incoming packets which do not match any flow table rule are either forwarded to the controller for further processing, or dropped.

CONTROLLER FRAMEWORKS In addition to a pure implementation of the southbound interface, SDN controllers typically offer an extendable software platform which provides a wide range of supporting features, such as topology discovery, distribution, or inter module messaging mechanisms. In addition, these so-called controller frameworks target different deployment environments and programming languages. The first widely available OpenFlow controller was NOX [66]. Hence, NOX has been extensively employed by the research community for prototyping SDN concepts, testing new OpenFlow features and evaluating the feasibility of new controller designs. NOX applications are implemented in C or Python (POX) using an event driven model, i.e., module functions are triggered by the arrival of specific OpenFlow protocol messages. While NOX/POX is extremely versatile it is not primarily aimed for production use, as it is not optimized for performance and stability and lacks resilience features.

Other controller frameworks aimed for deployment in production environments, include Beacon [50], Maestro [29], FloodLight and OpenDaylight [90], all of which are implemented in Java. FloodLight is the open source basis for a commercial OpenFlow controller. Other controllers include Ryu [107], Trema and MUL. Currently OpenDaylight is the youngest and probably also largest SDN controller platform. It is backed by the Linux Foundation and developed by an industrial consortium, which includes Cisco, Juniper and IBM, among many others. OpenDayLight includes numerous functional modules which are inter-

connected by a common service abstraction layer. Further, OpenDaylight provides a flexible northbound interface using REST APIs, and includes support for the OpenStack cloud platform. A comparison of the performance of key controller frameworks is provided in [142].

SDN VIRTUALIZATION The concepts of virtualization and SDN are closely related as a centralized control plane which possesses a global network view of the substrate network may be used to generate multiple concurrent, abstract representations of the underlying physical resources. FlowVisor [132] was the first framework which aimed to virtualize the access to the switch flow tables, enabling multiple users to independently install flow entries. To this end, FlowVisor assigns each user a set of header match patterns, called flowspaces, which the user is permitted to modify. FlowVisor ensures that the assigned flowspaces are non-overlapping and acts as an intermediate layer between the substrate OF devices and the user controllers in order to enforce the access policy for each flowspace.

Part of this thesis is dedicated to extending the concept of FlowVisor to enable the deployment of fully virtualized SDN network slices, where each network tenant is presented with a virtual SDN topology. Each vSDN should be programmable without notable restrictions, and completely unaware of potential concurrent vSDN deployments. In addition, we aim to identify principle abstractions necessary for implementing a hypervisor layer for SDN.

In [30] the authors present use cases and outline a design for an SDN hypervisor. Our work elaborates and generalizes several aspects identified in [30], such as the encoding scheme and mechanism necessary for mapping physical resources to virtual contexts and addresses flowspace isolation issues. In FlowN [46] database technology is used to efficiently store and manipulate the mapping between physical and virtual resources. In VeRTIGO [44] FlowVisor is extended to support virtual topology slices, however without full flowspace isolation. A recent proposal [2] for SDN topology virtualization uses layer 2/layer 3 addresses to bind resources to a logical context.

The scalability of SDN technologies is crucial for the deployment in production systems. Scalability issues have been identified both in the control plane [101, 154, 70], and in the data plane where the flow table size in OF switches may become a limiting factor [154, 130]. In Onix [84] and [41] distributed SDN controller architectures are proposed to improve the control plane scalability.

2.2 ENABLING TECHNOLOGIES

A significant appeal of the SDN paradigm is that it facilitates the reuse of network functions and the integration of existing technologies. The open nature of a number of system virtualization and networking technologies have facilitated a rapid development of new solutions.

In the context of network virtualization, server virtualization solutions are frequently used to isolate network functions running on commodity servers. Virtualization technologies can be classified into three types: full virtualization [78], paravirtualization [8] and container-based virtualization [108, 96]. Full (or hardware) virtualization provides a fully emulated machine abstraction and offers the highest level of isolation. Paravirtualization solutions rely on a virtual machine monitor (VMM) to achieve a higher performance. The VMM provides an optimized interface to the host hardware but require modifications to the guest operating system (OS). Container-based virtualization employs isolated resource slices within the host OS without a dedicated guest OS instances. Containers exhibit a small overhead and therefore offer high performance.

UNIX routing packages such as XORP [69] or Quagga [117] offer a stable and flexible basis for deploying a wide range of routing protocols (e.g., BGP, OSPF, IS-IS). The Click Modular Router [83] provides a low level platform for implementing arbitrary network functions in Linux systems. So-called Click elements can be chained together to implement high performance forwarding and custom packet processing operations, in kernel- and user-space. A highly flexible software switching implementation which implements OpenFlow as well as various management and monitoring interfaces is provided by OpenVSwitch [114].

2.3 NETWORK ABSTRACTIONS AND TOPOLOGY EMBEDDING

The concept of using network abstractions to hide complexity of the underlying network has been used in various contexts.

In [28] a platform for centralized inter-domain routing was developed. The hose model presented in [47] introduced the concept of using edge demands to define point-to-cloud virtual private networks (VPNs). In the position paper [75] the authors advocate the platform as a service paradigm in the context of virtual networks. Moreover, the authors propose a single router abstraction as a means for facilitating management. In the context of SDN a single node abstraction is proposed in [44].

Given an arbitrary virtual topology the embedding problem, i.e., the task of mapping each virtual resource to some set of physical resources, subject to some cost metric, is known to be NP-hard. As a result a number of heuristics have been developed which approximate the optimal solution. The problem is aggravated in the case of constrained physical resources. Topologies consisting of a backbone path with edge nodes attached to the nearest backbone node are evaluated in [94]. The authors present an approach aiming to minimize the cost of such virtual backbone networks, given pairwise traffic demands and unlimited network capacity. For arbitrary edge demands the authors find that tree-like backbone topologies yield lowest allocation costs. A commonly used strategy for network embedding is to perform node and link mapping separately. The authors of [135] present an algorithm which aims to max-

imize the number of embedded virtual networks (VNs). In [156] multiple arbitrary VN topology requests are mapped to the substrate network aiming to achieve a low and balanced utilization of substrate resources while taking link and node stress into account. Additionally, the online mapping problem is addressed by periodically recalculating the embedding. A number of VN embedding approaches [91, 32] aim to merge the node and link mapping phases. An approach advocating link splitting in the substrate network and the use of path migrations for periodic re-optimizations of the embedding is presented in [152]. The proposed heuristic takes both node and link constraints into account. An adaptive provisioning approach is presented in [73] which can cope with network failures.

2.4 QUALITY OF SERVICE AND NETWORK MEASUREMENTS

Providing QoS guarantees in traditional networks has been a long-standing challenge for the networking community. Given the best effort nature of the Internet architecture, which contributed significantly to its widespread success, the implementation of QoS mechanisms has attracted the attention of the research community culminating in a wide range of theoretical and practical results. In the context of virtual networks QoS is crucial as a means for providing isolation between virtual networks and enabling the deployment of network services with varying demands on a shared substrate.

In the last decades, two well studied QoS architectures have emerged: differentiated services (DiffServ) [11] offer a coarse-grained mechanism which classifies traffic into classes with different priorities. The classification is typically carried out at the network edge and the DiffServ class is encoded in the IPv4/IPv6 packet header. At each router packets are forwarded based on their assigned class, such as default per-hop behavior (PHB) (i.e., best effort) or expedited forwarding PHB (for traffic with the highest priority). On the other hand, integrated services (IntServ) [22] is a fine-grained architecture which operates on the level of flows. In this scenario, applications define requirements for their traffic and attempt to reserve the necessary resources along the desired network path using the RSVP protocol [23]. Each router may decide whether to accept or deny the reservation request depending on its available resources. The traffic specification (TSPEC) is defined in terms of token bucket parameters, where the token rate corresponds to the average rate of the flow and the bucket depth indicates the maximum burst size. Additionally, a request specification (RSPEC) defines the level of service desired. To effectively provide end-to-end QoS guarantees both DiffServ and IntServ require that all associated network routers support the corresponding QoS architecture, making an Internet-wide deployment highly challenging. Nevertheless, QoS mechanisms are successfully deployed within individual Internet service provider (ISP) networks, to provide specific network services (e.g., IPTV, VPNs).

As network resources are finite, enforcing a specific QoS policy requires admission control, i.e., the ability to grant or deny traffic access to the network subject to the current network utilization and the characteristics of newly arriving flows. However, in general it is difficult for network applications or users to accurately describe these flow characteristics a priori. In addition, as Internet traffic is bursty a deterministic flow characterization (e.g.using token bucket parameters) yields overly pessimistic (worst-case) bounds. To this end, a large body of work [80, 81, 65, 25, 26, 74] deals with measurement based admission control (MBAC) where admission decisions are based on measurements of the network utilization rather than an explicit user specification. Effectively, the traffic characterization task is shifted from the application to the network operator. In [25] the authors compare a range of MBAC algorithms and find that these yield essentially equivalent results. In [65] the performance of admission control schemes is analyzed, evaluating the effects of measurement uncertainty and flow dynamics. The authors identify a critical time scale over which the impact of admission decisions persists.

Traffic engineering aims to optimize specific network parameters such that the network performance is w.r.t. some given metric. Traditionally traffic engineering has focused on tuning link weight of routing protocols to minimize the maximum link load across all links [56, 57, 4]. More recently traffic engineering with MPLS [149, 147], which facilitate the estimation of traffic matrices [128, 138], has attracted the interest of ISPs.

INTERNET TRAFFIC A number of measurement studies performed in the 90s [85, 111, 37, 53] revealed that Internet traffic exhibits long range dependency (LRD) and self-similarity. Self-similar stochastic processes exhibit the same distribution over different time scales up to a rescaling factor, known as the Hurst parameter. Further, LRD processes are characterized by a slow decay of the autocovariance function which manifests itself as traffic burstiness. In [137] it was shown that self-similar LRD traffic emerges given an aggregation of a large number of on-off sources with heavy tailed on and off periods. The relationship between self-similarity and heavy-tailed distributions was recently verified experimentally in [93]. The fundamental impact for strongly correlated traffic processes on network performance is demonstrated in [85, 51]. As an example, the inefficiency of buffering LRD is evaluated in [106], supporting recent arguments on reducing buffer sizes [3].

More, recently the framework of probabilistic network calculus has been applied to study the impact of traffic processes on network performance [87, 54, 95]. In [61] effective envelopes motivate the use of so-called rate-interval curves as a means for estimating the properties of self-similar network traffic. In [87] the authors relate the notions of effective bandwidth [76] and effective envelopes. The authors of [122, 121] derive end-to-end performance bounds for networks carrying LRD traffic.

SAMPLING Estimators for extracting the Hurst parameter from contiguous-time series are presented in, e.g., [10, 136, 144]. However, in many practical scenarios capturing full traffic traces for the estimation of traffic characteristics is not feasible, e.g., due to high forwarding rates and processing and storage limitations of the monitoring infrastructure. In such scenarios, deterministic or random sampling approaches are frequently used to reduce the amount of captured data.

A fundamental sampling result [148], known as Poisson arrivals see time averages (PASTA), states that the portion of Poisson arrivals that see a system in a certain state corresponds, in average, to the portion of time the system spends in that state. In [100] the authors derive general conditions for arrivals see time averages (ASTA), showing that bias free estimates are not limited to Poisson sampling. In the context of network measurements the authors of [5] use an argument on joint ergodicity coining the term non-intrusive mixing arrivals see time averages (NIMASTA). Theorems provided in [146, 150] show that deterministically sampled, continuous-time long memory processes retain their statistical properties after sampling. For a class of self-similar processes [151], the authors of [150] prove that any band limited scale stationary process is determined by a sampled version, where the samples are taken at exponentially spaced sampling intervals. Estimators for the self-similarity parameter under deterministic, exponentially-spaced sampling intervals are derived based on least squares in [150] and on maximum likelihood in [146]. The work in [146] shows that within deterministic sampling techniques of continuous-time LRD processes, exponential sample intervals yield better results than equidistant sampling.

In the context of SDN, an algorithm for identifying heavy hitters in the switches is presented in [40]. In [141] a framework for the estimation of traffic matrices is proposed. In [102] accuracy trade-offs in SDN measurements are evaluated. The authors of [33] develop an OpenFlow API for collecting flow statistics at different aggregation levels.

3

PROBLEM STATEMENT

In the previous sections we described the state of the art in network virtualization and reviewed the related work in the area of performance evaluation. Next, we outline the research problems addressed in this thesis and present our contributions.

The notion of network virtualization encompasses a broad range of concepts including the slicing of physical resources for concurrent use, and conversely the abstraction of resource pools as a single virtual entity. Furthermore, virtualization approaches may be employed at the level of individual devices, such as switches and links, as well as at the level of entire network topologies.

As shown in the previous section, several solutions which address different aspects of virtualization in SDN have been proposed [132, 44, 46, 2]. Throughout these works several recurring concepts such as packet tagging and flowspace segmentation are employed. However, the requirements for encoding the virtual contexts within physical resources and the relationship between different types of context identifiers have not been analyzed systematically. Such an analysis, enables an abstraction of the implementation of a given virtualization platform from the underlying dataplane technology. As a consequence, the design of a portable SDN virtualization framework with support for arbitrary topologies and live-migration mechanisms is greatly facilitated.

Another aspect which has not been sufficiently studied is how the choice of vSDNs topology affects optimization strategies employed in the physical and virtual domain. In a virtualized environment which supports arbitrary vSDN topologies simultaneous optimizations carried out by the operator and the tenant may have a negative impact on network performance. Therefore, it is crucial to identify and implement constraints which prevent such scenarios. Insights in this area lead to a better isolation of resources and may be used to simplify the deployment and operation of vSDNs.

Finally, the SDN paradigm is built around the assumption that controllers possess an accurate and comprehensive global view of the network resources. Ideally such a network representation should encompass a wide range of QoS parameters, in order to enable SDN applications to tailor their forwarding logic to the network state. While several monitoring mechanisms have been recently proposed [153, 141, 102] for improving the visibility in SDNs, to the best of our knowledge, the extraction of advanced statistical metrics, such as backlog and delay bounds, has not been addressed. Such metrics can be expected to play an increasingly important role in upcoming network designs which rely heavily on SDN abstractions, e.g., through automation.

RESEARCH QUESTIONS In this work we revisit the concept of network virtualization in the context of the increasing abstraction levels advocated by emerging networking paradigms such as SDN. We address the following fundamental questions:

- What is the role of network topologies in the virtual domain?
- Which virtual network abstractions are feasible and what constraints must be enforced in order to deploy these?

Answering these questions is crucial for defining the requirements for virtualization architectures. Insights about the differences between traditional network topologies and virtual networks provide guidelines for the implementation of network functionality, such as provisioning of resilience or elasticity, in the virtual and physical domains. Such considerations pave the way for reducing the complexity in virtual topologies enabling, e.g., virtual networks which are specified in terms of a connectivity between a set of geographical points of presence (PoPs) with specific QoS demands. Such network abstractions facilitate the deployment and operation of network services by reducing configuration complexity and minimizing the likelihood of configuration errors.

Given an understanding of the requirements for virtual networks the following questions arise:

- What are the architecture requirements for enabling the concurrent deployment vSDNs on top of a shared infrastructure?
- Which minimal set of mechanisms is required to implement arbitrary virtual topologies in an SDN environment?
- Which embedding strategies minimize the capacity allocation cost for the deployment of virtual connectivity services?

The answers to the above questions enable the development of a scalable SDN virtualization architecture which supports both the instantiation of arbitrary virtual SDN topologies as well as the deployment connectivity services.

In the second part of the thesis we focus on performance evaluation approaches which enable network operators to optimize the utilization of the available physical network resources. A prerequisite for optimization is a detailed and up-to-date view of the current network state. Due to the properties of LRD Internet traffic over-provisioning is inevitable for maintaining reliable performance in computer networks. Thus, an accurate characterization of the traffic requirements of virtual networks, may be used to allocate capacities and to derive bounds for the QoS experienced by a tenant. To this end, the substrate network components must be monitored leading to the following research questions:

- Which traffic metrics enable the derivation of useful QoS parameters for arbitrary network flows?

- How can these metrics be efficiently extracted from observations of the network traffic?
- What are the effects of random sampling on the considered flow metrics?
- What is the impact of finite measurement durations on the quality of the estimates?

The answers to these questions enable the design of a monitoring framework which uses traffic observations, obtained from packet or counter samples, to derive QoS metrics, such as stochastic delay, backlog bounds and queue length distributions. These metrics may be exploited by network operators and SDN applications to evaluate alternative resource allocation strategies before committing changes to the network.

4

VIRTUALIZATION ARCHITECTURES FOR SOFTWARE DEFINED NETWORKING

In this chapter we evaluate the challenges of deploying virtualization in network environments with programmable forwarding resources. We develop a general framework for the instantiation and operation of concurrent SDN topologies. Additionally, we develop a systematic labeling scheme for working with resource slices in software defined networkings.

We begin this section with an analysis of the requirements of virtualized network environments. From there we derive two main use-cases for network virtualization: programmable tenant networks and connectivity services. We develop a platform for virtualizing arbitrary SDN topologies in Section 4.2. Next, we propose a design for deploying connectivity services on top of the aforementioned platform in Section 4.3. We exemplify this approach using routing as a service. The results presented in this chapter are based on prior work by the author [16, 17], as well as joint work with P. Papadimitriou [18, 20, 19].

4.1 REQUIREMENTS FOR VIRTUAL NETWORK ARCHITECTURES

A fundamental prerequisite for the successful deployment of virtualization in the networking domain is the design of an architecture which leverages the benefits of logical resource abstraction without incurring additional complexity. In fact, a well designed platform will enable operators to reduce management overhead and facilitate the network operation.

In this section, we aim to identify suitable architecture abstractions. To this end, we first consider the requirements towards a computer network and evaluate how these requirement are altered in the context of virtualization. In the following, we represent network topologies as a set of nodes arbitrarily connected by edges, specifically as directed graphs. Nodes perform forwarding or packet processing operations (e.g., switches, routers, middleboxes), while the edges represent network links (fiber, copper, or wireless). In the sequel, we assume that the physical network topology is managed by a network operator (or infrastructure provider (InP)) which acts as a substrate for multiple virtual network topologies, leased and operated by tenants (or virtual network operators (VNOs)). We assume that such virtual networks should be functionally indistinguishable from a physical network. To motivate the virtualization approaches presented in this thesis, we define the following requirements for traditional networks, ordered by importance:

CONNECTIVITY The raison d'etre, and hence primary requirement, of a computer network is the provision of point-to-point connectivity between geographically distributed end-systems. As a consequence of the layered Internet architecture, end-hosts do not have to deal with the implementation details of the connectivity topology. The network path connecting two hosts may span multiple intermediate nodes. Furthermore, the path over which data packets destined to a specific destination are transmitted, may vary over time.

The current Internet architecture is based around the principle of *best effort* connectivity. This means that no guaranties are given about the successful delivery of the transmitted data, the arrival time and arrival order of the data packets as well as the QoS. A consequence of this permissive approach is a significant gain of flexibility for the design of novel network mechanisms.

An additional key requirement for connectivity is a common, globally unique addressing scheme which is understood by all participating end-systems, with the Internet protocol (IP) being the predominant addressing and routing mechanism in the Internet today, while Ethernet is typically used in LANs.

RESILIENCE A prerequisite for maintaining network connectivity is the ability to ensure a disruption-free network operation in the case of network failures or abrupt changes in utilization. Thus, we consider resilience as an additional requirement for computer networks. Although the definition of connectivity is independent from resilience, carrier grade networks typically mandate connectivity with high availability requirements. To this end, large scale network deployments must implement mechanisms which ensure that no single segment of the network can disrupt the operation of the entire network. Such single points of failure can be due to hardware or software failure, as well as an overload of the network components.

Resilience is typically implemented through redundancy, and more specifically through the design of a suitable network topology. Hence, network topologies typically consist of multiple ring segments in order to provide multiple disjoint paths between sources and destinations, thereby minimizing the impact of link and node failures or changing demands. In the case of network failure automated fallback [103, 88, 109] mechanisms are used which redirect network traffic over alternative paths. Additionally, path redundancy is also exploited to achieve a uniform utilization of the available network resources. As a result, network operators can avoid hot-spots which cause a degradation of the network connectivity service. To this end, load-balancing techniques [140, 71, 59] aim to utilize all available network paths.

MANAGEABILITY Computer networks may provide varying degrees of configuration and monitoring functionality in order to support and enforce the above mentioned requirements. Configuration refers to a wide range of mechanisms [116, 49] for deploying, manipulating and updating network devices and their control plane logic, while monitoring refers to the instrumentation for querying the state of network components. While the impact of these auxiliary requirements is hard to quantify it is clear that they may significantly benefit network operation. A continuous and timely monitoring of system components enables operators to quickly identify end react to network problems and service degradation. Further, suitable configuration interfaces may be used to minimize the need for frequent operator intervention which may be slow and prone to errors. Monitoring and configuration mechanisms may be combined to define and automate network processes.

QUALITY OF SERVICE The notion of quality of service may be regarded as an optional class of requirements which is often desired in computer networks. Due to the best effort nature of the Internet architecture today, enforcing QoS aspects on a large scale is challenging and often neglected. We consider available bandwidth guarantees, as well as bounds on packet loss and queueing latencies to be a part of this requirements subset. Note that such QoS parameters rely on the connectivity and resilience requirements outlined above.

In this work, we argue that in the context of network virtualization the roles of the requirements described above are shifted significantly. As a result, we must develop clear definitions of the concept of virtual networks and outline feasible deployment scenarios for virtualized infrastructures. Clearly, connectivity remains the primary requirement in the virtual domain. However, in our view in a virtual network the concepts of redundancy and topology are ambiguous and are therefore unsuitable as a means for implementing resilience. We elaborate on this point in the sequel. On the other hand, the deployment of a wide range of virtualized network services on top of a shared physical substrate requires connectivity with well defined QoS guarantees. As a result, the importance of topologies is reduced in the virtualized domain, while the provision of connectivity and QoS requirements become primary requirements.

Next, we identify some key properties and limitations of the topology abstraction in the virtual domain, discuss trade-offs, and derive two models for designing and deploying virtual networks. Consider a *virtual* network topology depicted in Fig. 2 that is mapped onto a physical topology using some virtualization technology. Note, that in the virtual topology two disjoint paths exist between nodes A and B. In this scenario, without the introduction of further constraints, the network embedding algorithm may potentially map a segment of the disjoint virtual network

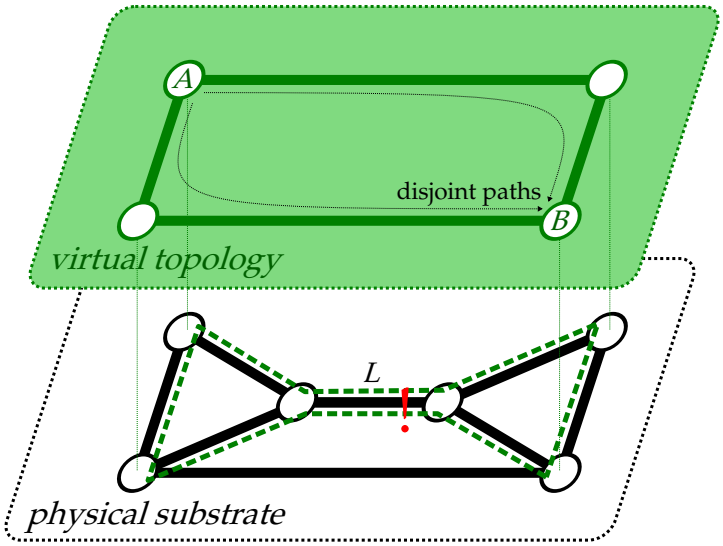


Figure 2: A virtual topology with N:1 Mapping. Note that the redundancy in the virtual topology may be negated by the mapping onto a single link.

paths, onto a single link in the physical substrate (denoted L in Fig. 2). Hence, if the embedding logic is not aware of the specific resilience requirements of the virtual topology, any benefits from maintaining redundant network paths in the virtualization layer may be rendered void. We denote a resource embedding where one or more entities belonging to a specific virtual network topology (e.g., link or node) may be mapped to a single entity in the physical substrate, as an *N:1 embedding*. From the example above, it follows that the use of a simple N:1 embedding is not sufficient to implement resilience within a virtual environment without additional constraints for the embedding algorithm.

On the other hand, an N:1 embedding does not preclude the use of load distribution mechanisms in the virtual network topology. As long as a strict isolation of the virtual resources is guaranteed by the substrate virtualization technology, any load distribution strategy employed by the tenant remains effective. However, the allocated resource characteristics (e.g., available capacity) must remain constant. Otherwise any optimization approach in the virtual domain may be negated by optimization mechanisms employed concurrently by the operator of the physical substrate. As an example, consider the case where the substrate operator performs a live-migration to remap a virtual network link onto some alternative physical path (e.g., to optimize the substrate utilization). In the virtual topology this transparent migration operation may manifest itself, e.g., as an increase in the propagation delays, triggering a load balancing algorithm. As a consequence, simultaneous optimizations performed by the tenant and the operator may leave the network in an oscillating state. Hence, we conclude that resource optimization should be performed either only in the virtual or only in the physical

domain. This relationship motivates the virtualization designs outlined in the sequel.

INFRASTRUCTURE VIRTUALIZATION First, we introduce an approach, which we denote *infrastructure virtualization*. Here, we formulate the redundancy requirements of the virtual topology as constraints of the topology embedding algorithm. The aim is to enable the protection of virtual network elements from hardware failures by enforcing physical redundancy in the substrate. To this end, any tenant wishing to deploy a virtual topology must explicitly request that all virtual network nodes and links should be mapped to distinct physical entities. This strategy implies a *1:1 embedding* between the virtual and physical network resources: i.e., each virtual entity is mapped to exactly one physical resource[1]. Note, that it is valid to map elements from different virtual network instances to a common physical resource. To further increase the level of resilience the operator may also employ a *1:N embedding* where, e.g., a single virtual link is mapped to multiple paths in the substrate network.

In addition, in order to enable the deployment of load-balancing techniques in the virtual network, a VNO may request guarantees regarding various metrics of the allocated resources, such as available capacity, memory or CPU, as well as various QoS parameters. For any topology request accepted by the substrate operator sufficient resources must be provisioned to ensure that the characteristics of the allocated resources will persist throughout the lifespan of the virtual topology. Any remapping performed by the infrastructure operator must remain transparent with respect to the allocated QoS characteristics. A consequence of the above approach is that the VNO maintains full control over the employed resilience mechanisms. Hence, it is critical that sufficient resources are requested before instantiating of the virtual network. We present an architecture for infrastructure virtualization in Section 4.2.

CONNECTIVITY VIRTUALIZATION Next we propose an alternative approach, which is motivated by a high level abstraction of a virtual network as a connectivity providing entity. Specifically, the implementation of resilience mechanisms, i.e., redundancy, is completely offloaded to the physical network substrate eliminating the need for complex virtual topologies. In this scenario tenants formulate VN requests as demands with respect to the connectivity between of a set of nodes. Each node represents a geographic PoP where the tenant attaches to the virtual network. Each attachment point is characterized in terms of service level agreements (SLAs) which define metrics such as capacity, QoS parameters and availability requirements. In this scenario, the InP is responsible for provisioning sufficient resources in the substrate to meet the tenant demands, as well as for implementing adequate redundancy

1 A virtual link may span multiple physical hops if these are not used by another virtual link from the same topology.

and transparent failover mechanisms in order to fulfill the negotiated SLAs. We denote this approach *connectivity virtualization* and present an corresponding architecture in detail in Section 4.3.

We highlight that an InP may simultaneously deploy virtual slices using both virtualization approaches, depending on the requirements of each VNO. In the following sections we present an SDN based architecture for implementing the approaches outlined above using OF as a salient building block. Initially, we describe a general framework for SDN infrastructure virtualization. Subsequently, we show how this framework may be used to implement connectivity virtualization using routing as an exemplary connectivity service.

4.2 A FRAMEWORK FOR SDN INFRASTRUCTURE VIRTUALIZATION

In this section, we develop a framework which enables the deployment of arbitrary virtual topologies on top of a physical SDN substrate, comprised of arbitrarily interconnected programmable switches. We consider a scenario where VNOs may easily request and acquire VN topologies from InP on a lease basis, enabling a rapid deployment of new services. Our goal is to enable virtual network tenants to deploy fully customized network slices, which are adaptable to their specific application scenario. To this end, tenants should be able to make unrestricted use of the existing SDN mechanisms to configure the switching logic of the allocated resources. Thus, we aim to provide virtual networks which are indistinguishable from a physical SDN substrate from the tenant's point of view. In the following we use the term vSDN to denote such a fully programmable virtual network topology.

Further, we aim to minimize the overhead associated with instantiation and operation of vSDN. In order to setup a virtual topology the infrastructure operator must allocate and map virtual resources to physical network devices, configure appropriate traffic encapsulation actions on all relevant switches and segment the switch flowtables. To facilitate management and minimize the risk of human error, these steps should be automated. All operation necessary for implementing the virtual SDN abstraction are concealed from the VNO. Finally, we address scalability challenges for the control and forwarding planes.

In the following, we assume that all SDN substrate components support the OpenFlow (OF) specification [99]. Today, OF is supported by a large number of hardware vendors and is the predominant SDN technology. However, note that the mechanisms outlined in the following are applicable to any flow-based SDN technology.

We define three key requirements for our SDN virtualization infrastructure.

DATA AND CONTROL PLANE VIRTUALIZATION In order to implement a true virtual SDN, tenants must be able to deploy custom controllers to program the forwarding entries of the switches in their

virtual topology. The virtual topology abstraction should provide a well-defined SDN interface, such that a tenant controller cannot distinguish the virtual SDN substrate from the physical one. Moreover, as the flow tables in physical switches are shared between multiple tenants, mechanisms are required to ensure that the table entries of each tenant are fully isolated and cannot be accesses or modified by other tenants.

INFRASTRUCTURE INSTANTIATION Mapping an arbitrary virtual topology onto a subset of the physical network graph requires a mechanism for unambiguously mapping resources to the corresponding virtual context. To this end, all tenant resources are marked using a suitable labeling scheme. The associated network operations (e.g., packet tagging) must be transparently carried out by the InP. Moreover, the instantiation of virtual links spanning multiple physical hops requires the installation of forwarding entries in intermediate physical switches, which are not visible in the vSDN topology. Finally, in order to provide isolation of the virtual resources the substrate operator may allocate queues and configure suitable schedulers across a large number of infrastructure devices.

LIVE-MIGRATION The virtualization platform must allow a transparent migration of virtual nodes and links, in order to achieve versatility in the infrastructure. Such a mechanism can be used by the operator to remap virtual entities in the case of failure of physical components, maintenance works, or to provision tenant requests for additional virtual resources. To this end, the reconfiguration of the necessary forwarding rules must be orchestrated in a way which minimizes disruption.

4.2.1 *SDN Hypervisor*

To address the aforementioned requirements we introduce a transparent virtualization layer, called SDN hypervisor, that coordinates the embedding, deployment and management of vSDNs. The SDN hypervisor is positioned between the physical data plane layer and the tenant controllers, as depicted in Fig. 3. It exposes a number of virtual OF switch interfaces to each VNO, while acting as a single controller instance when communicating with the physical switches. The hypervisor performs the following main functions:

TOPOLOGY EMBEDDING. vSDN requests are specified as a network graph with a set of assigned node and edge attributes, such as backplane and link capacity, geographical location, flow table requirements as well as QoS constraints. The hypervisor maps the resources of each vSDN topology request to physical components, taking resource utilization and VNO constraints into account. To this end, the hypervisor main-

tains a global view of the available resources in the substrate topology (e.g., by periodically querying network elements) and keeps track of the reserved resources. The resource mapping allocation is carried out using existing virtual network embedding algorithms (e.g., [32, 152]).

VSDN SETUP AUTOMATION. Once the resource mapping has been computed, the hypervisor automatically generates auxiliary forwarding and encapsulation flow entries in the substrate devices, necessary to instantiate the virtual topology (in the sequel these are called infrastructure flows). Furthermore, for each virtual node, the hypervisor allocates flow table space in the substrate switches. We discuss labeling schemes which ensure the isolation of virtual links and virtual flow tables in Section 4.2.2. Moreover, the hypervisor configures rate limiters and traffic schedulers on associated switch ports to enforce bandwidth isolation and QoS constraints.

CONTROL MESSAGE PROCESSING. The hypervisor manages the VNO access to the substrate forwarding elements, transparently enforcing resource isolation and access control. The hypervisor intercepts all control messages generated by the tenant controllers and the substrate switches, and performs filtering in order to limit the visibility of each vSDN topology to its owner. Further, for each vSDN node the hypervisor exposes a logical flowtable abstraction to which only the VNO has dedicated access. To emulate this, the hypervisor transparently rewrites the control messages exchanged between VNO controllers and the physical switches and replaces all references to physical switch, table, port and queue identifiers with the corresponding virtual topology identifiers and vice versa.

RESOURCE MIGRATION The hypervisor automates vSDN node and link migration by coordinating the necessary switch flow table updates. The hypervisor triggers the installation and removal of flows entries across the substrate devices, using an order which minimizes the disruption of traffic. Further, it updates all relevant resource identifiers and updates the mapping used for control message translation. We discuss virtual node and link migration mechanisms in Section 4.2.4.

4.2.2 *Framework Design*

In this section we propose a design for implementing the aforementioned functionality on top of an SDN substrate. We consider a generic SDN technology, analogous to OpenFlow, in which each SDN forwarding element, both physical or logical, possesses a single logical lookup table populated by a remote controller entity. Each entry in this so-called flow table contains a match rule (with wildcards) associated with a pointer to a set of packet processing actions. The actions are applied whenever the rule conditions are matched by an arriving packet. The

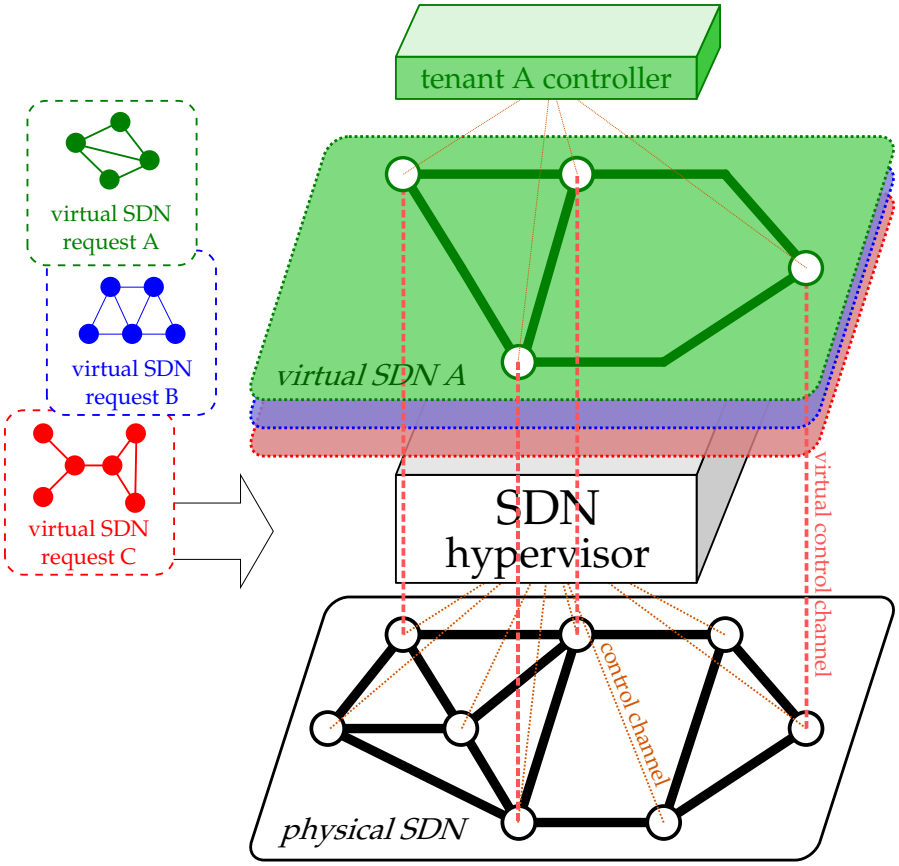


Figure 3: Operator substrate network hosting multiple virtual SDN slices. Each virtual topology is isolated and programmed by a tenant controller.

key for the table lookup is generated from the packet header bits as well as metadata such as the input port.

Similarly, in the sequel we represent a virtual SDN node as a logical lookup table associated with multiple virtual links (attached to virtual ports). Initially, in Section 4.2.3 we address the question whether this simple SDN forwarding element model, which hides the implementation complexity, is a suitable abstraction for virtualized SDNs. In order to implement a virtualized SDN forwarding element, two main mechanisms are required. First, the ability to segment the physical flow table into multiple logical tables, which can be concurrently modified by different VNOs in an isolated manner. Second, a packet tagging scheme to emulate virtual links spanning multiple physical hops. Next, in Section 4.2.4 we consider mechanisms for live migration of virtual resources, specifically the migration of virtual nodes and virtual links. Finally, we discuss the scalability of the hypervisor used in our SDN virtualization platform in Section 4.2.5.

4.2.3 *Context Identifiers*

In a virtualized environment all assets of a VNO must be unambiguously associated with a single logical context, which is valid throughout the lifetime of the virtual network instance. We use the term assets to denote resources which are controlled solely by the tenant, i.e., flow table entries in the switches as well as data packets which these entries act upon. In the following, we utilize the concept of context identifiers (XIDs) which bind tenant assets to a specific logical context. An XID is associated with a tenant resource using a suitable encoding, which may change depending on the asset type or the location in the network. The logical context of each VNO is comprised of a unique set of XIDs. In FlowVisor [132] the authors introduce the notion of flowspaces to allocate segments of the flow table to a specific user. A flowspace is defined as a bit-pattern in the flow table which matches a unique set of traffic packets. To ensure isolation, FlowVisor assigns each user a unique (non-overlapping) flowspace and polices the access to the flow table. Hence, flowspaces act as XIDs for the users. In this work we extend this approach, by completely hiding the XIDs from the tenant. To this end, the hypervisor allocates XIDs and transparently inserts these to the associated resources (e.g., through encapsulation flow actions and augmenting the lookup keys). Furthermore, the hypervisor rewrites all control messages of the SDN protocol which target a specific virtual resource, by appending/removing the corresponding XID such that the encoding is not exposed to the tenant. Finally, the hypervisor only permits tenants to access assets tagged with their own set of XIDs. As a result, each tenant's view of the virtual network is limited to their own logical context and an unrestricted abstraction of a dedicated network is obtained.

The XID for packets traversing a physical link is encoded as a set of bits in the packet header, which are used only by a single tenant. This tunnel or virtual link tag is typically inserted by some encapsulation/decapsulation mechanism at the ingress and egress of a switch, respectively. Numerous tunnelling protocols exist which differ in the header location of the tag as well as the number of used bits. Common examples include VLAN tagging at layer 2, MPLS at layer 2.5, or VxLAN at layer 4 [97]. However, the identifier may also be defined using existing header fields, such as MAC headers, if the deployment scenario permits such a restriction. More generally, the packet XID encoding may span an arbitrary number of bits distributed across one or more header fields of a packet. We refer to XIDs encoded inside the packet header as tunnel identifiers (TUIDs). Note, that the virtual tunnels may be directed, i.e., a separate TUID is assigned for each direction, or duplex, i.e., both link directions share the same context identifier.

Next, we consider the XID encoding of the virtual flow table. In an SDN switch the flow entries of multiple tenants are stored in a single physical lookup table, as depicted in Fig. 4. Each tenant may define arbitrary match rules with wildcards and corresponding actions, whose

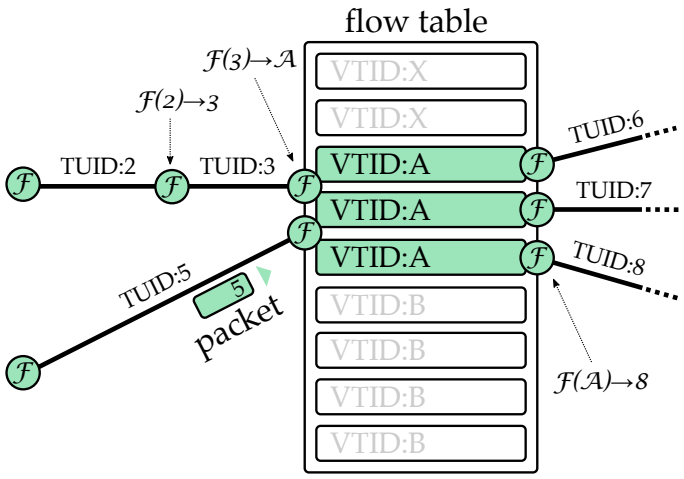


Figure 4: The tunnel identifiers encoded in the headers of incoming packets are mapped to virtual flow table identifier A. At the table egress new TUIDs are assigned using flow actions.

scope is limited to the tenant's logical context. For instance, within their virtual networks two VNOs may use overlapping IP ranges with different routing policies. In this case the hypervisor must provide isolation, i.e., guarantee that the actions associated with a tenant's flow entry will not overwrite and invalidate any entries previously installed by another VNO. To ensure that the match attributes are unique for all rows of the physical lookup table, the hypervisor attaches an XID to each flow table entry, by transparently modifying the attributes of the match rules generated by the tenant's controller. We refer to XIDs encoded inside the flow table entries as virtual flow table identifiers (VTIDs).

Match	Actions
`1001 1001 1010`	drop
`00** 10** *110`	forward port 1
`11*1 1010 0001`	forward port 2
`0110 001* 0101`	forward port 4
`0101 0011 1101`	drop
`11** 00*0 0000`	forward port 3

Table 1: Flow table consisting of a 12-bit flowspace. Wildcard values are represented by $*$.

To illustrate the need for dedicated VTIDs consider the flow table example in Table 1, where the first three flow entries are installed by tenant A while the remaining flows are installed by tenant B. Clearly, in this scenario all entries are unambiguous, i.e., an incoming packet will match exactly one of the six entries. Moreover, a hypervisor could uniquely map each flow to a specific tenant as the flowspaces defined by the

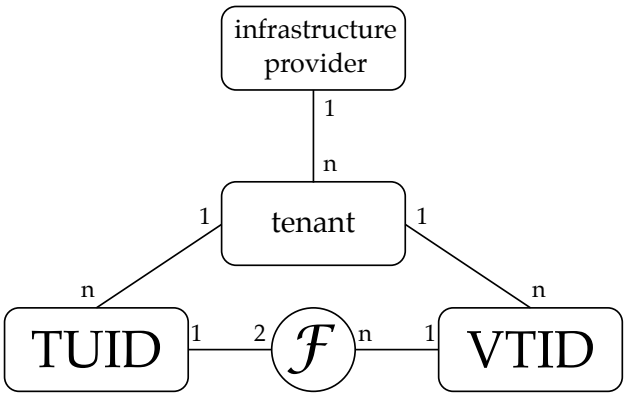


Figure 5: Relationship between virtual context identifiers.

match fields are non-overlapping. Specifically, the last three bits of the flowspace may be interpreted as XIDs for each table row, as these values compose two disjoint sets, each corresponding to a specific tenant. Note, that this property does not hold for an XID defined over the first two bits of the flowspace. Next, consider that the controller of tenant A is allowed to install an additional flow entry [`1111 *00* 000*` $\rightarrow$ drop]. With this entry incoming packets can no longer be unambiguously mapped to a specific tenant, as the new entry may also match traffic belonging to tenant B (entry 6). To prevent such violations of the flow table partitioning, the hypervisor must either restrict the insertion of entries which disrupt the XID encoding scheme, or must automatically append an addition XID to the flowspace. As our goal is to enable fully virtualized network slices we use the latter option which does not limit the type of entries which may be installed by the tenants.

So far, we defined TUIDs which bind packets to specific virtual links, and VTIDs which bind flow table entries to a specific flow table slice. We now evaluate the relationship between these two XID types. Recall, that each virtual node is comprised of a virtual flow table and a set of associated virtual links. Thus, the set of TUIDs connecting to a virtual switch could potentially be used as VTIDs in order to identify the flow entries of a specific tenant. However, such an encoding approach is only feasible if tenants are not allowed to use the input virtual link in combination with wildcards as an attribute in the flow match field. Consider a tenant flow entry which matches packets on *any* incoming virtual link. Such an entry effectively clears the tenant VTID and results in an ambiguous partitioning of the flowspace. To deal with this issue the hypervisor could automatically expand wildcards and generate an individual flow for each possible input tunnel (i.e., all tenant TUIDs connected to the virtual switch) as depicted in Table 2. However, such an approach scales poorly: assuming n virtual tenant links, the number of tenant flow rules may be increased by a factor of n in the worst case.

Such issues are avoided if the logical context of the flow entries is encoded using a VTID which is decoupled from the TUID encoded in the

Match	
Port/TUID	Header
2	1001 1010
*	**0110 0101**
5	0110 0101

(a)

Match	
Port/TUID	Header
2	1001 1010
1	**0110 0101**
2	**0110 0101**
3	**0110 0101**
4	**0110 0101**
5	0110 0101

(b)

Table 2: Equivalent flow tables before and after wildcard expansion. Tenant A is assigned virtual links with TUIDs=$\{1, 2, 3, 4\}$ and tenant B is assigned TUIDs=$\{5, 6\}$. The expansion of the wildcarded entry yields 4 flow entries.

packet headers. In this case, a conversion between the XIDs encoded in the different asset types must be carried out. To this end, we introduce the notion of an XID translator. We define the XID translator as a function $\mathcal{F}(\text{XID}) \rightarrow \text{XID}^*$ which exchanges a given context identifier with a new identifier XID^*. In practice, translator functions are implemented as additional table lookups, or are incorporated as part of an existing flow table. An XID translator will typically be used to map tunnel identifiers to virtual flow table identifiers, i.e., $\mathcal{F}(\text{TUID}) \rightarrow \text{VTID}$, and vice versa, i.e., $\mathcal{F}(\text{VTID}) \rightarrow \text{TUID}$. For the example illustrated in Table 2 the XID translator flow table depicted in Table 3a may be used to assign a VTID for each tenant in the switch. The resulting flow table of the physical switch are shown in Table 3b. We emphasize, that XID trans-

Match TUID	Actions
1	set_VTID 1
2	set_VTID 1
3	set VTID 1
4	set VTID 1
5	set VTID 2
6	set VTID 2

(a)

$\Rightarrow$

Match			Actions
TUID	VTID	Header	
2	1	1001 1010	...
*	1	0110 0101	...
5	2	0110 0101	...

(b)

Table 3: Table (a) implements an XID translator, which assigns each tenant a unique VTID which is used in the lookup table (b).

lators may also be employed to exchange identifiers of the same type, e.g., $\mathcal{F}(\text{TUID}) \rightarrow \text{TUID}^*$, for instance if different segments of a virtual

link are encoded using different TUIDs. In the naive scenario with wildcard expansion outlined above, the hypervisor encodes the translator function within the flow table. The relationship between the different context identifiers is depicted in Fig. 5. Note also, that the calculation of a hash value over some set of header fields may be regarded as an operation performed by a XID translator.

All translator lookup table entries are instantiated and managed by the SDN hypervisor. As the translator table lookup is performed immediately before the logical flow table lookup, the translator table may either be located inside on the penultimate intermediate switch or stored directly on the physical host switch. In the latter case, the translator lookup table may be mapped to additional tables exposed by the switch hardware (e.g., through OpenFlow). Alternatively, the translator flow entries may be stored alongside the regular tenant flow table flows. To this end, after the translator lookup, packets are redirected back to the ingress of the switch pipeline using a logical loopback interface[2].

SCOPE OF THE CONTEXT IDENTIFIERS Generally, a TUID may have a scope which is global, tunnel level or link level. With a global scope, packets traversing a specific virtual link are identifiable by a globally unique TUID. Using a tunnel level scope, the identifier is unique across all physical hops spanned by the virtual link. However, TUIDs may be reused in non-overlapping segments of the network. Finally, with a link level approach a different TUID is assigned at each physical hop. Tunnel and link level TUIDs have the advantage, that they may minimize the number of header bits required for encoding. However, a significant disadvantage is that the allocation of such non-overlapping identifiers is NP-hard (see routing and wavelength assignment (RWA) problem). Moreover, this constraint severely limits the flexibility of virtual link migration approaches. Therefore, in the following we assume a global TUIDs scheme, with encapsulation taking place at each virtual node and simple forwarding at intermediate physical nodes. On the other hand, VTIDs assigned by the hypervisor must be unique *at least* at the physical switch level in order to ensure the isolation of the logical table. The VTID may have a global scope, however this requires a substantially larger address space, which may be expensive to implement in the lookup tables of current SDN switches.

The encoding scheme outlined above enables the hypervisor to map arbitrary virtual topologies to to any physical substrate with SDN forwarding devices.

2 Version 1.0 of the OpenFlow specification supports logical interfaces as an optional feature.

4.2.4 *Resource Migration*

Live migration enables InPs to seamlessly remap allocated virtual network elements on to different substrate resources. The ability to transparently reassign network resources is a crucial element for optimizing network utilization. Moreover, in virtual SDN environments such optimizations may be automatically performed by the SDN hypervisor layer. Migration mechanisms must maintain the logical functionality of the virtual entity throughout the migration process. Furthermore, the migration must be carried out in a way which minimizes the disruption of both the physical and the virtual networks.

In this section, we discuss techniques which may be implemented by the SDN hypervisor to perform a seamless migration of virtual SDN resources. In a virtualized environment migration entails two operations: (i) virtual node migration, i.e., the transfer of the logical flow tables between physical switches, and (ii) virtual link migration.

VIRTUAL NODE MIGRATION. To migrate a virtual node the SDN hypervisor must transfer all tenant flow table entries stored on a source switch to the new destination. In addition, the hypervisor must update all context identifiers associated with the migrated flow entries. The order of the migration steps is crucial to avoid disruption of the traffic: initially, all flows which make up the logical flow table are cloned to the destination node. On the destination switch, the hypervisor updates the VTIDs to avoid conflicts with existing entries and configures any used queues and schedulers. Subsequently, the hypervisor temporarily maintains two instances of the flow table entries, i.e., any tenant modifications to the flow entries are propagated to both the source and destination switches, augmented by the appropriate VTIDs. As a result, the new table is logically correct but does not yet receive any incoming traffic. Next, each virtual link attached to the virtual node is asynchronously migrated using one of the mechanisms outlined in the next section. While the link migration is in progress, both switches process some part of the total virtual node traffic. Finally, after the link migration process is completed, the hypervisor deletes all relevant flow entries from the source switch.

Note, that the flow table migration may also be performed asynchronously, i.e., flow by flow, if all flow table entries are mutually independent. However, generally dependencies between the flow entries emerge whenever flow priorities and wildcard matches are used in the flow table. To exemplify this problem, consider a scenario in which longest prefix matching is used for IP lookup. Given two entries with overlapping prefixes at the source switch, the packet forwarding logic is altered if any one of the two entries is migrated individually to the destination switch. The dependencies between flows may, however, be diminished using appropriate transformations of the rules [119, 92].

VIRTUAL LINK MIGRATION. The migration of virtual links involves transferring all associated tunnel context identifiers to a new set of intermediate nodes. We outline two migration schemes. In the first approach the hypervisor effectively instantiates a new virtual link between the source and destination switches which is assigned a new TUID. To this end, forwarding entries containing the new TUID are installed on all intermediate nodes on the new link path. Subsequently, the hypervisor updates all tenant entries referencing the original TUID with the new TUID. The order of the update operations is crucial to ensure that the migration does not generate inconsistencies in the flow table which may result in packets loss. First, the hypervisor must augment the translator function implementation at the destination switch to include the new TUID. Note that the original translator entries are maintained in parallel as long as the original virtual tunnel remains in use. Next, the hypervisor begins updating all flow actions at the source switch which perform packet tagging using the original TUID. As a result the traffic begins traversing the new virtual link. After all actions at the source have been updated, the translator entries at the receiver which reference the original TUID can be discarded. Finally, the hypervisor completes the migration by removing all old forwarding entries on the intermediate nodes. The process is carried out for both directions of the virtual link[3].

In order to avoid the duplication of translator entries the virtual link migration may be performed without introducing a new tunnel identifier, if the forward and reverse directions of each virtual link are encoded using separate context identifiers. To this end, the hypervisor migrates each directed virtual link asynchronously. The key aspect of this migration technique is that the forwarding entries for each hop along the path are installed by the hypervisor in the direction opposite to the traffic flow. As a result, a valid forwarding path is maintained throughout the migration process and no traffic is dropped. Note, that this differs from the previous approach where the installation of the intermediate node flow entries may be carried out without any ordering.

The migration mechanism is exemplified in the scenario depicted in Fig. 6, where a virtual link mapped to path P with TUID=101 is migrated to a new destination path P′. The direction of traffic flow is from node A to node B and the intermediate nodes are denoted n_k. When the migration is initiated, a forwarding entry is first installed at intermediate node n_5. Subsequently, corresponding rules are installed at n_4, n_3, n_2, n_1. Note that the rule at (3) overwrites the previously installed flow entry at the intermediate node n_3, redirecting traffic from port *p1* to port *p2*. As a result, the traffic flows now traverses the path n_2, n_3, n_4, n_5. Subsequently, the remaining flow entries are installed (steps (4) and (5)). In the final step (6), all flow actions entries associated with TUID=101 at node A are updated to point to the new physical port *p2* and any forwarding entries associated with the old link path are discarded.

3 Note, that both link directions may be encoded using the same TUID assuming that input ports are encoded in the flow table.

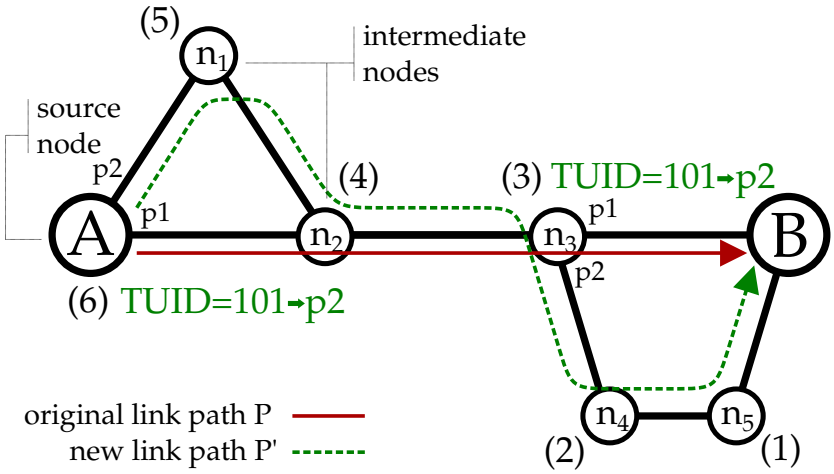


Figure 6: Migration of a directed virtual link with TUID=101. Numbers in brackets indicate the flow rule installation/update order.

It is important to highlight that while the techniques outlined above guarantee that no packets are lost during the migration process, packet reordering may occur in cases where the destination path is longer than the original path. Specifically, let d denote the one way delay of the original path and let d' denote the one way delay of the new path P'. Neglecting queueing delays and assuming a fixed capacity $C_P = C_{P'}$ of the virtual link, if $d' > d$ the maximum amount of data that may arrive out of order may be approximated by $\frac{1}{C_P}(d' - d)$.

4.2.5 *Scalability of the SDN Hypervisor*

In the following, we address factors which affect the scalability of the proposed approach. As the controller traffic from all tenants must be processed by the SDN hypervisor it is important to ensure that this module does not become a bottleneck of the architecture. In order to cope with large volumes of control traffic, we propose a distributed hypervisor design depicted in Fig. 7. The distributed hypervisor layer is comprised of a central resource manager module (RMM) and multiple controller proxies (CPX). The main idea behind the approach is a segmentation of the substrate network into multiple non-overlapping SDN domains. Each SDN domain is assigned a dedicated CPX which is responsible for all resources located within the domain. Essentially each CPX performs the operations described previously for the non-distributed hypervisor for a subset of the substrate network graph, i.e., the CPX configures devices and performs all necessary translation operations of the control traffic exchanged between the forwarding plane and the tenant controllers. The RMM is responsible for the allocation of substrate resources and coordination of the controller proxies.

The key functions of the hypervisor modules are summarized in Table 4. To illustrate the functionality of the distributed hypervisor in the

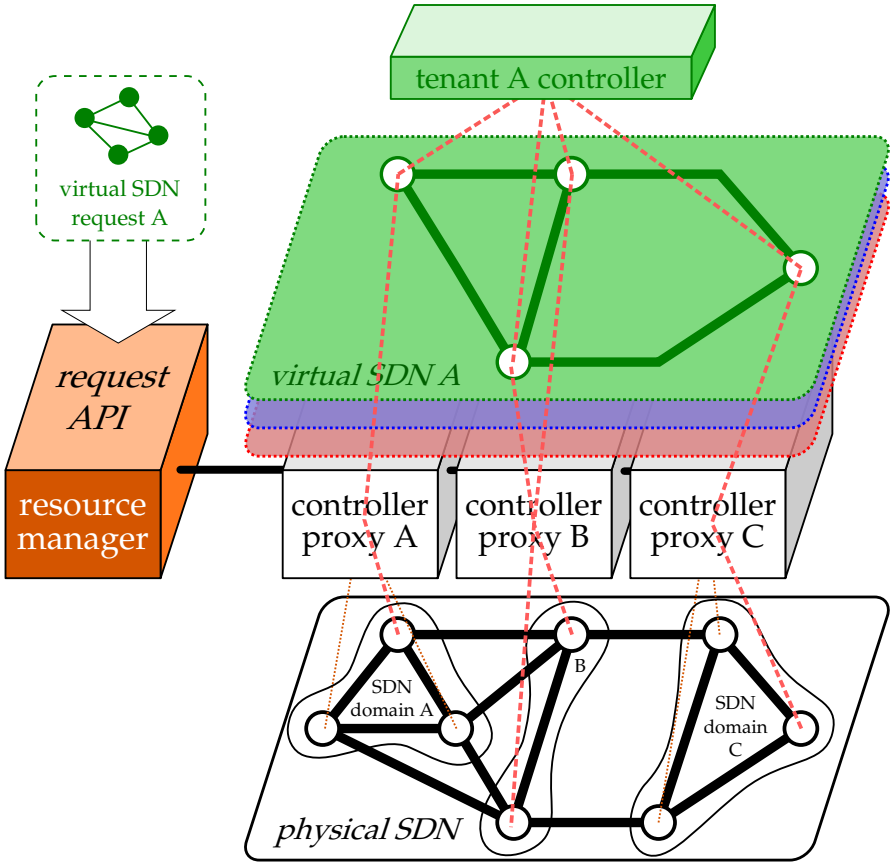


Figure 7: Distributed SDN hypervisor architecture. Each controller proxy is responsible for an independent SDN domain.

following we outline the steps performed during the deployment and operation of a virtual SDN.

VSDN DEPLOYMENT. In order to deploy a virtual SDN slice the tenant generates a vSDN request and forwards it to the RMM. The RMM evaluates the resources available in the substrate network and computes an initial mapping between the logical topology and the physical network. The general VN embedding problem is NP-hard and therefore, a large number of heuristics have been proposed [32, 152, 91, 39] which yield near optimal solutions to the problem subject to various constraints (e.g., capacity, delay, forwarding table size). Subsequently, the RMM notifies the relevant CPX about the newly allocated vSDN resources within their SDN domain. Moreover, the RMM assigns globally valid XIDs for the logical resources and distributes these to the CPX. Finally, each CPX installs the infrastructure flow entries into the relevant switches of its SDN domain. At this point the vSDN is ready to be configured by the tenant.

VSDN OPERATION. After the instantiation of the vSDN topology all control traffic between a tenant controller and a specific virtual switch is forwarded to the CPX responsible for the associated physical switch. As

Manager module
SDN domain segmentation
Virtual topology embedding
CPX resource assignment and coordination
Global optimization (reprovisioning/migration)
Assignment of context identifiers
Controller proxy
Infrastructure flow setup
Control message translation
SDN domain optimization

Table 4: Summary of the functions performed by the manager module and controller proxy.

in the non-distributed case each CPX transparently rewrites the control messages, inserting/removing context identifiers as necessary. Additionally, each CPX performs policy control to ensure the isolation of the flow table slices.

Each SDN domain operates independently from the remaining domains. Communication between virtual resources distributed across domains relies on the XIDs assigned by the RMM. Consequently, the state information associated with any virtual node may be maintained solely by the responsible CPX. Furthermore, each CPX may optimize the resource utilization within its domain by migrating logical resources. On the other hand, the RMM may reassign substrate resources between SDN domains in order to optimize the substrate utilization globally.

4.3 CONNECTIVITY AS A SERVICE

In this chapter we focus on the concept of connectivity virtualization proposed in Section 4.1 - a collection of virtual network resources which provide connectivity between multiple PoPs as a service to tenants based on some requested network technology (e.g., Ethernet, IP, MPLS). From the tenant perspective the connectivity service acts as a single logical entity which mimics the behaviour of a corresponding physical device. Possible use-cases include forwarding devices such as switches and routers as well as middle-boxes which perform specialized network function such as firewalling or intrusion detection. We argue that such an abstraction covers a wide range of use cases for network virtualization. The approach eliminates the need for complex virtual topologies and uses familiar network concepts abstractions thereby significantly simplifying the operational overhead for tenants, as well as the integration into an existing network infrastructure. From the infrastructure provider (InP)

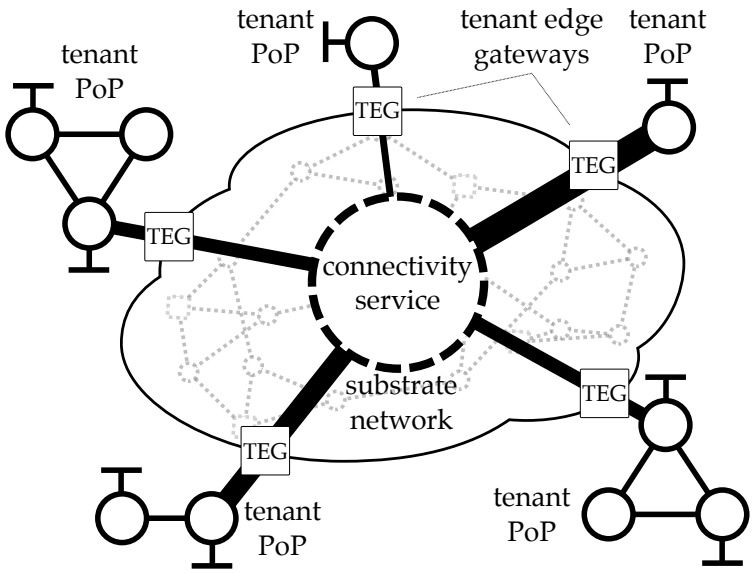


Figure 8: A connectivity service between five tenant PoPs hosted on top of a provider substrate. The tenant capacity demands for each PoP are defined as SLAs. Each tenant PoP is attached to the infrastructure provider (InP) network over a tenant edge gateway (TEG).

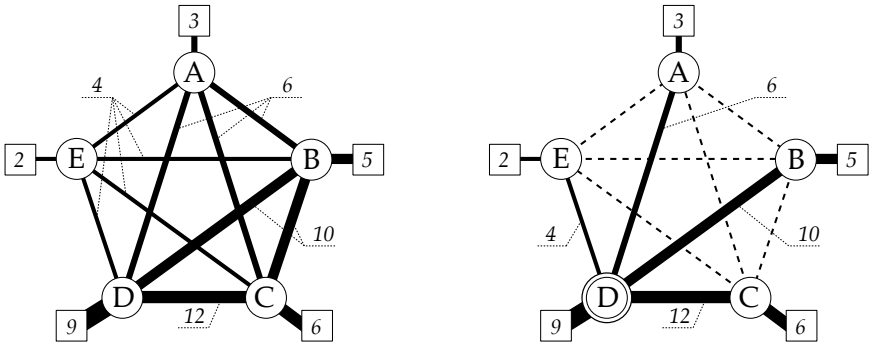
perspective the simplicity of the abstraction facilitates the embedding of virtual resources onto the physical substrate.

A connectivity service is defined in terms of a series of demands, such as capacity and delay, at the tenant attachment points. In addition, tenants may define requirements of the service itself such as the amount of packets which it must be able to process. The demands are formulated as a request to the substrate operator, which in turn attempts to allocate the necessary resources using an embedding algorithm. In the sequel we exemplify this concept using a layer-3 connectivity service which we refer to as virtual router service (VRS) depicted in Fig. 8. The tenant is presented with a virtual router instance which spans an entire network segment.

The concepts presented in this section are based partly on our previous works [16, 17] as well as joint work with P. Papadimitriou [19, 20]. The structure of the chapter is as follows: we begin by motivating a simplified topology for connectivity services defined by edge demands in Section 4.3.1. Next, we outline a corresponding network embedding algorithm which yields an *optimal* allocation of resources in capacity constrained substrate networks in Section 4.3.2. Finally, we present our framework for virtual routers as a service (VRaaS) in Section 4.3.3.

4.3.1 *Connectivity Service Embedding*

In this section, we evaluate alternatives for virtual topologies which implement connectivity between multiple tenant PoPs. Consider a tenant request for a connectivity service between a set of N tenant attachment



(a) Fully connected topology (b) Rooted tree topology

Figure 9: Equivalent embeddings for a connectivity service between five customer PoPs. The unidirectional capacity demand for each PoP is denoted in the corresponding box. Link labels indicate the bidirectional capacity reservation costs. The total capacity allocation costs are $S_K = 66$ (a) and $S_{star} = 32$ (b).

points (PoPs). Initially, assume that the tenant specifies bidirectional capacity demands b_u at each $u \in N$. First, we evaluate the maximum capacity which the substrate operator must reserve in order to satisfy the tenant request, i.e., we examine the worst case scenario where the tenant may transmit traffic using the maximum possible transfer rate between any two attachment nodes, which are not known a priori.

Without loss of generality, we model the substrate network as a fully connected graph connecting all PoPs u, where each graph edge i represents a minimum cost network path between two PoPs with respect to some cost metric, as depicted in Fig. 9. We assume that all links are bidirectional and have infinite capacity (we consider capacity constrained topologies in the next section). Moreover, we assume that the capacity allocated to each tenant is guaranteed (i.e., the resources are reserved without oversubscribtion). Further, let S denote the capacity allocation cost for a connectivity service request defined as the (weighed) sum of capacities on each substrate path which are allocated in each direction in order to connect all possible pairs of tenant attachment points.

The graphs depicted in Fig. 9 illustrate the two edge cases for embedding the virtual topology of the connectivity service. Both topologies offer equivalent connectivity and fulfill the tenant capacity demands. In Fig. 9a all attachment points are interconnected directly resulting in a complete graph. To satisfy the capacity demands between any pair of nodes $(u, v) \in N$, $\min(b_u, b_v)$ units of bandwidth must be allocated along the corresponding path i. Hence, denoting the path cost between nodes u an v as c_{uv}, the allocation cost for the fully connected virtual topology S_K is given as

$$S_K = 2 \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \min(b_i, b_j) c_{ij}. \tag{1}$$

Using unit link costs and capacity demands yields $S_K \sim n(n-1)$ and therefore the allocation costs increase with the number of attachment points as $O(n^2)$. Hence, the use of a complete graph as an allocation scheme for connectivity services may quickly become unfeasible for scenarios involving a large number of tenant PoPs.

Alternatively we can use a rooted tree topology to connect all tenant attachment points using a central hub node as depicted in Fig. 9b. Specifically, we may select any node $k \subset N$ as a root and forward traffic between all remaining PoPs over least cost paths connected to this node[4]. As both the input and the output traffic rates at each attachment point u are limited by b_u the topology satisfies the capacity requirements if each path connected to the root is assigned $\min(b_u, b_k)$ units of bandwidth. The corresponding allocation cost S_{tree} is given by

$$S_{tree} = 2 \sum_{i=1}^{n-1} \min(b_i, b_k) c_{ik}. \tag{2}$$

Clearly S_{tree} grows linearly with the number of customer edge nodes. Moreover, for any $k \in N$ the tree topology is a subgraph of the complete graph and therefore $S_{tree} < S_K$. For connectivity services which are defined in terms of demands at the attachment points, tree topologies yield the lowest capacity allocation costs. For an arbitrary substrate network, i.e., if we consider all intermediate substrate nodes and not only least cost paths, the position of the root node has a significant impact on the cost of the tree topology. We address the optimal location of root nodes in Section 4.3.2.

Next, we consider the impact of the virtual topology on network latency. Assume that the costs c_{uk} represent latency, and therefore the edge between any two nodes $(u, v) \in N$ of the substrate graph represents the network path with the smallest latency. It follows that the complete graph scenario yields the smallest possible delay for the connectivity service as all attachment points are connected directly. In a tree topology, the tenant traffic must additionally traverse the root node, resulting in a higher delay.

Finally, we consider the memory requirements of the two approaches. A significant benefit of implementing the connectivity service using a tree topology is that the (virtual) flow table (e.g., for routing table lookups) may be stored at a single, central location at the root node. At each attachment point only a small number of forwarding entries is sufficient to tunnel the traffic to the root node. On the other hand, the complete graph topology requires copies of the flow table entries at each tenant PoP.

The considerations above show that a trade-off exists between the total capacity allocation costs, the latency and the memory requirements for the two worst case embedding approaches outlined above. Next, we

4 Note, that in general the root is not required to be one of the attachment nodes and may be located anywhere in the substrate network

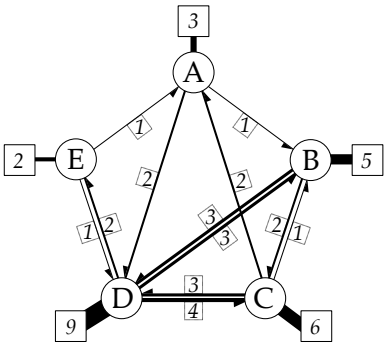


Figure 10: Capacity allocation for a connectivity service with point-to-point demands. The total capacity allocation costs are $S_{P2P} = 25$.

consider the ideal case where the tenant is able to exactly predict how much bandwidth will be required between any pair of TEGs, such that the maximum capacity demand b_u at each attachment point is not exceeded. An example of such a request is given in the following connectivity matrix which contains the directed capacity demands between any two nodes:

$$\begin{array}{c} \\ A\,(b_A{=}3) \\ B\,(b_B{=}5) \\ C\,(b_C{=}6) \\ D\,(b_D{=}9) \\ E\,(b_E{=}2) \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \\ (b_A{=}3) & (b_B{=}5) & (b_C{=}6) & (b_D{=}9) & (b_E{=}2) \end{array} \\ \begin{pmatrix} - & & 2 & & 1 \\ 1 & - & 1 & 3 & \\ & 2 & - & 4 & \\ 2 & 3 & 3 & - & 1 \\ & & & 2 & - \end{pmatrix} \end{array}$$

A corresponding point-to-point embedding is depicted in Fig. 10. Observe that the capacity allocation cost for this scenario is given by the total sum of the matrix elements $S_{P2P} = 25$. As each pair of nodes is connected using the shortest path and the required bandwidth is known exactly, this embedding is optimal. It is clear that inserting a root node to obtain a rooted topology will increase the cost. However, like the complete graph embedding case, this approach requires that a significant segment of the connectivity service forwarding table is maintained at each attachment point (entries which are unreachable from a specific host may be pruned). Furthermore, in practice the exact point-to-point demands are hard to predict in advance. Therefore, we expect that tenants will prefer to define the connectivity service using edge demands.

In this work we propose a hybrid approach which aims to exploit the benefits of all aforementioned topology embedding strategies. To illustrate this approach we consider the rooted topology embedding from Fig. 9b, and assume that the actual traffic being generated by the tenant is described by the connectivity matrix above. Next, we assume that the InP determines, e.g., through measurements, that the tenant is transferring a significant amount of traffic from nodes C to A and from B to E (i.e., 2 capacity units for each edge). The substrate operator then reserves

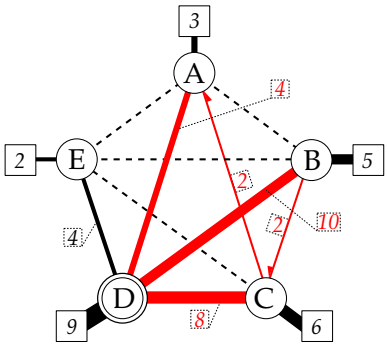


Figure 11: Capacity allocation for a connectivity service using a hybrid embedding strategy. The total capacity allocation costs are $S_{\text{hybrid}} = 28$.

capacity along the shortest paths between the source nodes C and B and the corresponding destination nodes A and E, and installs appropriate forwarding entries at the source nodes in order to redirect the traffic over the new direct paths. The operator may now free a corresponding amount of capacity from the rooted tree reservation. The resulting embedding is depicted in Fig. 11. For the given example the approach reduces the capacity allocation costs from 32 to 28. As the number of direct paths is increased the allocation cost converges to the optimal value.

We expect that such a reservation strategy will be effective in environments, where the aggregate tenant traffic is dominated by so-called "elephant" flows, which make up a large proportion of the total carried traffic. By forwarding such large volume flows over shortest paths the bandwidth which must be allocated by the operator is reduced. At the same time only a small subset of the connectivity service forwarding tables must be exported to the attachment points. We dedicate the second part of this thesis to methods for monitoring of traffic flows and the estimation of resource requirements, which may be employed for implementing the hybrid strategy outlined above. In the remainder of this chapter we develop an algorithm for embedding connectivity services using a rooted tree topology and provide a detailed description of a layer 3 connectivity service implementation in Section 4.3.3.

4.3.2 *Rooted Tree Embedding Algorithm*

In this section, we present an algorithm which allocates substrate resources for a connectivity service defined by capacity demands at the tenant attachment points. Based on the considerations above we implement the service as a rooted tree, where a root node performs the service logic and TEGs are connected to the root node using virtual links with sufficiently reserved bandwidth.

If we ignore the capacity constraints of the substrate network, a minimum cost allocation of a tree topology rooted at node k is obtained by reserving bandwidth along the shortest paths from r to each tenant PoP $e \in E$. Fulfilling edge demands in uncapacitated networks is discussed

in [47] in the context of point-to-cloud VPNs. In the sequel, we assume a substrate network with a finite capacity. In this case, a simple shortest path approach is not guaranteed to minimize the allocation cost.

We divide the rooted tree embedding into two distinct operations: the selection of an optimal root node and the allocation of optimal forwarding paths to the TEGs. Given a specific utilization of the substrate network we aim to minimize the total allocated capacity for each incoming connectivity service request. To this end, we formulate the path allocation task as a flow network problem where a requested amount of flow[5] must be transported from a source to a sink over a network with capacity constraints. First, we assume that the location of the root node k is known.

We model the substrate network graph as a flow network $G = (V, N)$ with nodes N and links V, and let $E \subseteq N$ denote the tenant attachment points. Further, we define the set E as a traffic sink with a flow demand of $b_E = \sum_{e \in E} b_e$ and the root node k as a traffic source with a flow supply of $b_k = -\sum_{e \in E} b_e$. We seek to find the so-called minimum cost flow (MCF) with respect to a given root node k. We may calculate a set of least cost paths which satisfy the capacity constraints using the successive shortest paths (SSP) algorithm [1]. The algorithm solves the MCF problem in pseudo-polynomial time by iteratively adding flow along the shortest path from arbitrary source/sink pairs until all TEG demands are satisfied. It has the advantage that it can efficiently handle edge demand changes or attachment of new TEGs. Note, that for demands $b_i \neq b_j$ for $(i, j) \in E$, the resulting optimal flow may be split along multiple paths [6] as advocated in [152]. Approximations for the unsplitable flow case are provided in [79, 133].

Next, we consider the choice of the root node. As the node which yields the minimal allocation cost is not known in advance, we may iterate over every feasible root node candidate and select the one for which the SSP algorithm yields the lowest allocation cost. To reduce the number of nodes which must be checked we propose the following approach.

Consider an instance of the substrate graph G with infinite capacity, denoted G_∞. For all possible root node candidates $n \in G_\infty$ we can obtain the path cost $c(e, n)$ between n and each tenant attachment point e using Dijkstra's algorithm and calculate the allocation costs $S_\infty(n) = \sum_{e \in E} c(e, n) b_e$. It follows that $S_\infty(n)$ is a lower bound for the allocation costs $S(n)$ of the capacity constrained scenario. Next, we sort the costs S_∞ and denote the corresponding ordered set of root node candidate as K. We iterate over the nodes in $k \in K$ starting with the smallest unconstrained allocation cost and calculate the allocation costs $S(k)$ for the capacity constrained case using the SSP algorithm. At each step the root node candidate k_{min} is updated if a smaller cost $S(k)$ is found.

The iteration may be terminated when the capacity constrained cost $S(k_{kmin}) \geqslant S_\infty(k)$, as the lower bound states that no better solution

5 Here we use the term "flow" in a graph theoretical context.

Algorithm 1 VRS embedding

1: prune nodes with insufficient resources
2: $S_\infty \leftarrow 0$ // initialize empty array of lower bound costs
3: **for** $n \in G_\infty$:
4: **for** $e \in E$: // iterate through all CEGs
5: calculate shortest path cost $c(e, n)$ between e and n
6: $S_\infty(n) \leftarrow S_\infty(n) + c(e, n)b_e$ // allocation cost with infinite capacity
7: $S_\infty \leftarrow \text{sort}(S_\infty)$ // sort S_∞ by descending cost
8: $s_\infty \leftarrow 0$, $s_{min} \leftarrow \infty$
9: $k_{min} = \emptyset$
10: **while** $s_\infty < s_{min}$:
11: $(k, s_\infty) \leftarrow \text{pop}(S_\infty)$ // get next least cost (node, cost) tuple
12: $s = \text{MCF}(k)$ // get least cost for capacity constrained case
13: **if** $s < s_{min}$:
14: $s_{min} \leftarrow s$, $k_{min} \leftarrow k$ // new minimum allocation (node, cost) tuple
15: **return** (k_{min}, s_{min})

can exist for the remaining nodes. The algorithm is outlined in Alg. 1. While the algorithm might still need to iterate over all candidate nodes times in the worst case, in practice the number of iterations is reduced substantially.

The search space can be significantly reduced by pruning substrate nodes and links with insufficient resources such as bandwidth, switching capacity or forwarding table space. Furthermore, latency bounds between VRS edges can be enforced by eliminating core node candidates exceeding predefined SLAs.

We note, that the algorithm described above yields an optimal embedding for the rooted tree for a given utilization of the substrate network. If an operator wishes to globally optimize all allocated virtual resources in the network, the resource reservation task may be formulated as a multi commodity flow problem, which is NP-hard and may be addressed using suitable solvers. However, in such a scenario a remapping of the virtual resources may trigger a large number of migrations resulting in a negative impact on network performance. As a simple (suboptimal) strategy, the operator may periodically evaluate whether sufficient resources have been freed in the substrate which warrant the re-embedding of a randomly chosen connectivity service.

In order to react to network failures the InP must provision resources for use by failover mechanisms. To this end, the operator may reserve additional capacity over disjoint network paths and maintain shadow instances of the associated flow entries. An algorithm for constructing redundant trees is presented in [9].

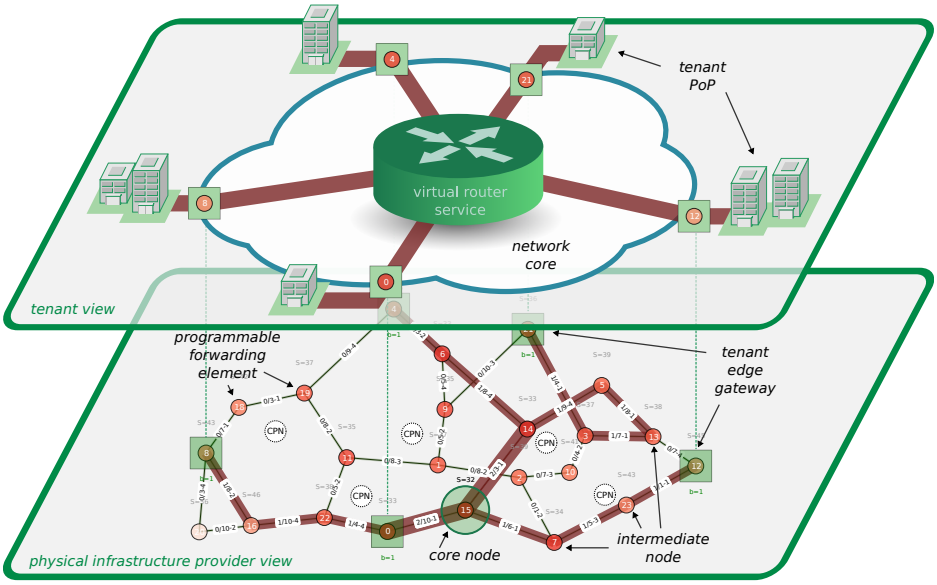


Figure 12: OpenVRoute: a layer 3 virtual connectivity service.

4.3.3 *Virtual Router Architecture*

In this section, we outline OpenVRoute, our architecture for a layer 3 virtual connectivity service. The architecture uses the concepts and algorithms outlined in the previous section to provide tenants with a virtual router instance which interconnects multiple point of presence (PoP) with specific capacity demands. Our goal is to demonstrate the feasibility of the virtual connectivity service approach using a real world implementation which provides flexibility and high performance. To this end our platform couples the performance of commercial programmable switches with the flexibility of software routers. We use off-the-shelf components and open source software which results in a cost-effective and extendable design.

Our previous work [15] has shown that software routers executed within a virtualization environment struggle to sustain high packet forwarding rates. Several works [42, 68] have achieved significant performance improvements by exploiting the massive level of parallelism provided by general-purpose CPUs and GPUs. Nevertheless, dedicated forwarding hardware is expected to continue offering significant performance benefits for the foreseeable future. In addition, the port density of hardware switches is typically an order of magnitude higher than what is achievable using general purpose PC architectures. To address these shortcomings, OpenVRoute aims to offload the packet forwarding of software routers to low cost, programmable OpenFlow devices which provide high performance. In addition, we address the problem of limited flow table sizes in hardware switches by introducing a secondary datapath executed on commodity hardware.

To provide isolation between the virtual router instances of multiple tenants, OpenVRoute uses the SDN virtualization framework described

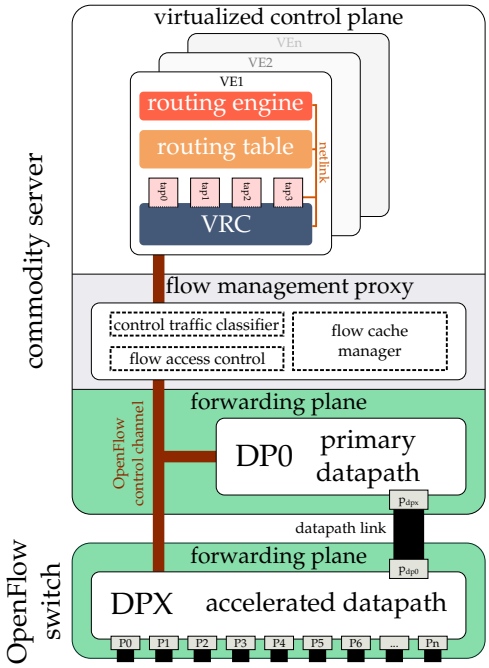


Figure 13: OpenVRoute architecture overview.

in Section 4.2. The architecture extends and elaborates on ideas presented in our previous work [16, 17] and addresses limitations identified in [127]. A prototype implementation of the architecture was designed within the BMBF G-LAB VirtuRAMA project [55]. Within the project a successful test deployment connecting PoPs in several cities was carried out in the network of a large German backbone carrier. Furthermore, demonstrators of the platform were presented at the BITKOM Future Internet Workshop 2012 and CeBIT 2012.

The remainder of this section is structured as follows. Next, we describe the functionality of the key architecture modules in a top down manner. We present the employed flow-table management operations and describe a set of exemplary packet forwarding operations in Section 4.3.4. In Section 4.3.5, we evaluate the performance and scalability of OpenVRoute using a prototype implementation.

OPENVROUTE OVERVIEW In the following, we detail the implementation of the root node of the layer 3 connectivity service. To instantiate the virtual links that connect the TEG with the root node, the InP installs a flow entry at each path node which encapsulate the tenant traffic using an appropriate TUID. The root node architecture is comprised of three main components as illustrated in Fig. 13. A *virtualized control plane* hosts the control logic of all deployed virtual router instances. Each tenant is assigned a dedicated control plane which is used to configure virtual interfaces, execute routing daemons and processes control traffic. A *flow management proxy* provides a transparent binding layer between the virtualized control plane and the forwarding plane. It coordinates the control traffic, performs access control and manages the available flow

table resources. In the *forwarding plane* OpenVRoute employs a dual datapath design consisting of a fast, hardware-based datapath and a slower, software-based datapath. This separation is motivated by the limited number of entries in the TCAM tables of OpenFlow switches. However, each control plane instance uses a single forwarding element abstraction. The communication between the components takes place over a control channel using the OpenFlow protocol. Therefore the components may be distributed across multiple physical hardware. In the sequel, we describe the components in detail.

VIRTUALIZED CONTROL PLANE. Each virtual router is assigned a virtual environment (VE) which executes all control plane processes and maintains a local routing table. The VEs provide a standard Linux environment and are hosted on a commodity server using an OS virtualization technology, such as KVM [78]. As a result, a tenant may deploy arbitrary routing daemons, such as XORP or Quagga [69], within the virtual router instance in order to populate the routing table. Further, each VE contains a set of user-space dummy interfaces, implemented using *TAP* devices, which allow the user and routing software to configure IP addressing. According to the identifier classification introduced in Section 4.2.2, each virtual interface is associated with a tunnel identifier (TUID) which uniquely maps traffic to the appropriate virtual context. Moreover, each TUID is mapped to a physical port in the datapath by the flow management proxy (FMP).

A key component executed within each VE is the virtual router controller (VRC) daemon. This process is responsible for handling the communication between each virtual control plane and its (virtual) flow table in the forwarding plane. Note, that all data exchanged between the VRC and the forwarding plane is mediated by the FMP. The VRC is implemented as an OpenFlow (OF) controller and uses OF control messages to communicate with the other components of the system. Essentially, the VRC acts as a glue between the OF protocol and the Linux kernel, and vice versa. Specifically, the VRC processes intercept all control traffic packets (e.g., ARP, ICMP, BGP) written to a TAP interface by the VE network stack and encapsulates these inside a `PacketOut` OF protocol message, instructing the datapath to transmit the packet over the corresponding physical datapath port. Conversely, all incoming control traffic, encapsulated in `PacketIn` OF messages, is extracted and injected into the appropriate TAP interface. As a consequence, the control packets are received and processed by the network stack of the VE operating system. In addition, the VRC transparently duplicates the routing table computed by the VE routing daemon onto the datapath flow table. To achieve this, the daemon monitors the Linux Netlink socket for notifications about routing table changes. Whenever a routing process adds, deletes or modifies a routing entry the VRC generates an OF message containing a corresponding forwarding rule and sends it to the datapath.

FLOW MANAGEMENT PROXY. The FMP acts as a transparent layer between the control plane and the forwarding plane, providing each virtual router instance with an abstraction of a dedicated datapath. Specifically, the FMP functions as an OF controller for the forwarding plane and exposes a slice of the flow table as a separate datapath to each control plane instance. The proxy performs three key functions implemented as modules.

1. A *control traffic classifier* acts as a de-multiplexer which inspects the incoming control traffic from the forwarding plane (which is encapsulated inside OF protocol messages) and forwards it to the associated VE for further processing. The mapping to a specific logical context is performed based on the TUID encoded in the control packet.

2. A *flow access control* module ensures that each VRC can only manipulate datapath flow table entries which belong to its logical context. Consequently, only valid OF insertion and deletion commands are forwarded to the forwarding plane. Again, the control is implemented using the known mapping between TUIDs and virtual environments. In addition, the module monitors and limits the rate at which control messages associated with each virtual router instance are generated in order to ensure that the control channel capacity is not consumed by a malicious tenant or a denial of service (DoS) attack.

3. A *flow cache manager* optimizes the utilization of the available flow table and forwarding resources in the dual datapath. To this end, the manager aims to identify large volume flows which should be cached in the accelerated datapath. The flow cache manager collects flow statistics from packet counters in the two datapaths using the OF protocol. The extraction of relevant traffic characteristics from counter observations is the focus of Section 5.5 in this thesis. For specific traffic offloading strategies refer to [130] and the references therein.

FORWARDING PLANE. The forwarding plane of the OpenVRoute architecture uses two datapaths to overcome the problem of small flow table sizes in dedicated OpenFlow switches and the limited port density and backplane capacity in commodity servers. To address theses limitations our design, depicted in Fig. 13, consists of a primary datapath (DP0) which is hosted on a commodity server coupled with an additional, high-performance, accelerated datapath (DPX) hosted on a dedicated OF switch. The datapaths are interconnected using a datapath link which comprises one or more high-capacity Ethernet connections that essentially act as a backplane. Both datapaths are controlled using the OF protocol and maintain separate flow tables.

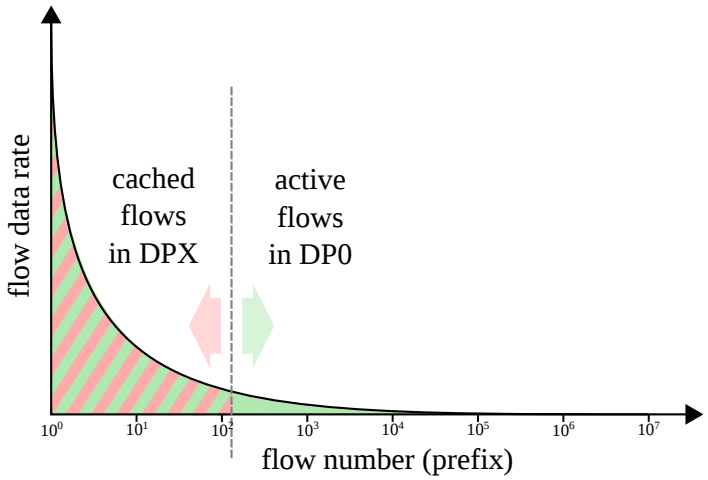


Figure 14: Distribution of flows vs. data rate: a primary datapath DP0 stores the forwarding entries of all virtual router instances, while the accelerated datapath DPX caches only a subset of flows with the highest data rates. The area beneath the curve corresponds to the total traffic rate.

The combination of the two datapath types enables us to exploit an inherent property of network traffic, namely the observation that given an aggregate of network flows, a small number of flows typically carry a large share of the total traffic volume. These dominating flows are known as "elephant" flows while the remaining low volume flows are referred to as "mice". This effect has been demonstrated repeatedly and shown to follow a Zipf distribution [155].

To take advantage of this relationship we use the fact that commodity servers offer abundant memory resources. Thus, a large number of forwarding entries can be stored in a software implemented OF flow table, at the expense of increased lookup times. Consequently, the raw forwarding performance of software datapaths is limited. On the other hand, dedicated OF switches which use specialized TCAM for fast lookups can sustain high forwarding rates but can typically only accommodate several tens of thousands forwarding entries (in contrast: today a full BGP routing table contains over 500 000 entries). Therefore, our architecture stores the forwarding entries of all virtual router instances in a software switch DP0 and caches elephant flows in the accelerated datapath DPX. As a result, the traffic load which must be processed by the software datapath DP0 is reduced significantly while the number of flow table entries which must be stored DPX is minimized.

The selection of cached flows as well as the update rate is performed by the flow cache manager (FCM). Note, that the primary datapath always maintains a full copy of the forwarding entries of all virtual routers. Flows cached in DPX are assigned a higher priority, such that they mask the corresponding entries in the primary datapath. The Zipf distribu-

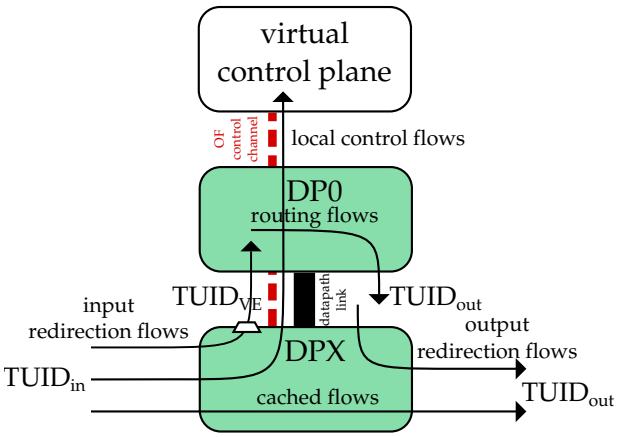


Figure 15: Flow entry types.

tion of traffic flows and the allocation to the two datapaths is illustrated schematically in Fig. 14.

4.3.4 *Flow Table Configuration*

In this section, we discuss the flow-level implementation specifics of the OpenVRoute architecture. We classify the flow table entries which implement the functionality of the platform into five distinct types. All flows are installed into one of the two datapaths by the FMP module using the OF protocol. Recall that the root node which performs the layer 3 forwarding decisions is connected to the TEGs using virtual links. Each virtual link is defined by a tunnel identifier (TUID) which uniquely maps the corresponding traffic to the logical context of a specific tenant. Depending on the operator setup and hardware deployed in the substrate network a TUID may be encoded in a VLAN, an MPLS header or even layer 2 addresses as outlined in our previous work [19]. In the following, we denote the tunnel identifier of an input virtual link as TUID_{in}, and the tunnel identifier of an egress link as TUID_{out}.

In each datapath flow table the forwarding entries of each tenant are unambiguously identified by a virtual flow table identifier (VTID). In the accelerated datapath we directly use the input TUIDs as VTIDs. In the primary datapath we encode an intermediate TUID_{VE} and use this identifier as a VTID. The motivation for this context translator function is described in the sequel. We use the following flow entry types, illustrated schematically in Fig. 15, to implement the encoding and decoding of the virtual resource identifiers:

INPUT REDIRECTION FLOWS These entries forward incoming traffic from any physical port at DPX to DP0 over the datapath link. To achieve this, each input redirection flow matches the identifier TUID_{in} corresponding to a connected virtual link. Furthermore, input redirection flows act as a XID-translator, multiplexing all

traffic associated with a specific virtual environment (VE) into a single tunnel identifier denoted TUID_{VE} before forwarding it over port P_{DP0}. The new identifier acts as a virtual flow table identifier (VTID) at the primary datapath, enabling the isolation of the tenant flow table slices, while ensuring that the forwarding table lookup for any IP prefix destination can be performed using a single flow table entry, regardless of the input TUID.

Input redirection flows have the lowest priority providing a default/fallback "slow" path for packet forwarding at DP0 (see flow entry A in Table 5 for an example).

ROUTING FLOWS (DP0) The primary datapath stores the IP forwarding entries for all hosted virtual routers. These routing flow entries match against a packet's destination IP, as well as the TUID_{VE} which acts as a VTID and associates each entry with a specific virtual router instance. Each flow table entry is assigned a priority by the VRC, such that destination IP addresses with the longest prefix are matched first. The actions associated with each flow entry assign an output tunnel identifier TUID_{out} to each packet, which is mapped to a specific physical port at DPX. Further, the source and destination MAC addresses are updated to match the addresses of the outgoing VE port and next-hop router, respectively (see entry G in Table 5). Finally, each packet is forwarded to the accelerated DPX via the datapath link connected to port P_{DPX}.

OUTPUT REDIRECTION FLOWS These entries stored at DPX decode the output port of the packets routed at DP0. Specifically, each entry matches packets arriving at the datapath link port P_{dp0} and uses the output tunnel identifier TUID_{VE} to select the physical interface over which the packet is forwarded (see flows B and C in Table 5). The mapping between the TUID_{VE} and physical ports is maintained in the FMP.

CACHED ROUTING FLOWS The flow cache manager may install routing entries which forward "elephant" flows over the accelerated datapath DPX. These entries use the input tunnel identifier TUID_{in} as VTID and additionally match the destination IP and MAC addresses of the incoming packets. The source and destination MAC addresses are updated analogously to the DP0 routing case. Finally the output tunnel identifier TUID_{out} is set and the packet is forwarded over the associated physical port. Note, that the FMP ensures that the logical consistency of the virtual router routing tables is not disturbed by the caching operation. To this end, the FMP generates non-overlapping IP prefixes for the cached flow entries.

Cached flow entries are assigned a higher priority than the redirection flow entries and therefore the matched traffic does not traverse the slow path of the system (see entry D in Table 5).

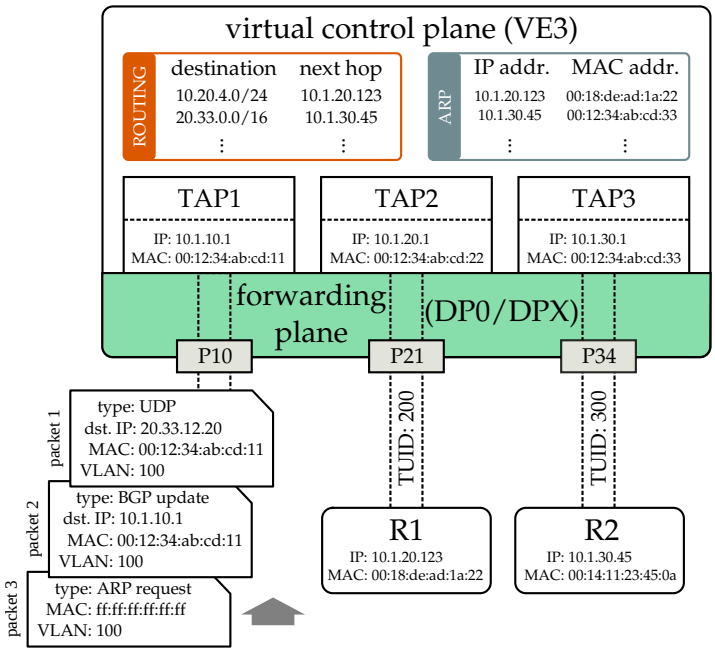


Figure 16: Exemplary virtual router instance VE3.

LOCAL CONTROL FLOWS These flows are responsible for forwarding packets destined to the control plane of a virtual router, such as routing messages and broadcast traffic, to the corresponding VE. To this end, the entries match the input tunnel identifier TUID_{in} and the destination layer 3 address belonging to a specific virtual interface. Additionally, an entry matching the layer 2 broadcast address is installed for each virtual interface in order to process broadcast traffic. The matched packets are encapsulated in an OF message and forwarded directly to the appropriate VE over the control channel where they are injected to the corresponding TAP by the VRC. These flows are assigned the highest priority in the flow table. The local control flow entries may be stored in either the accelerated or the primary datapath flow table. In the following, we assume that the entries are installed in DPX to minimize the processing overhead (see entries E and F in Table 5)).

We now outline a set of exemplary packet forwarding operations which illustrate the path of a packet traversing the different stages of the architecture. Consider a virtual router hosted within a virtual environment VE3 as depicted in Fig. 16. The router has three virtual interfaces TAP1, TAP2, and TAP3, each connected to a virtual link with TUID 100, 200 and 300, respectively. Assume that the TUIDs are encoded in VLAN headers. Further, the three virtual links are mapped to the physical ports P10, P21, P34, respectively. The layer 2/3 addresses of the virtual interfaces and an excerpt of the routing and ARP tables are given in Fig. 16. The virtual router is connected to two neighbor routers R1 and R2 reach-

able over the ports P21 and P34, respectively. Table 5 contains a set of relevant flow table entries installed in DP0 and DPX by the FMP.

FORWARDING AT DPX First, we consider `packet 1` with destination IP address 20.33.12.20. Assume that the packet belongs to an elephant flow and that the corresponding forwarding entry has been copied to the accelerated datapath (flow D in Table 5) by the FMP. According to the virtual routing table the next hop for the packet is router R2.

When the packet arrives at the switch, it will match the cached flow table entry D as this flow has the highest priority among all matching entries in the flow table (i.e., entry A). The associated flow action will rewrite the packet's source and destination MAC addresses, set the VLAN ID of the packet to 300 to correspond to the TUID of the egress link, and forward the packet to port P34.

FORWARDING AT DP0 Next, assume that flow entry D has been dropped from the flow table of accelerated datapath. Consequently, the IP lookup for `packet1` must be performed in primary datapath DP0. The packet will first match flow entry A in the accelerated datapath. The datapath will update the VLAN ID of the packet to 1003 to indicate that it belongs to the logical context of VE3. The packet will then be forwarded to DP0 over the datapath link.

At the primary datapath the packet VLAN ID and destination IP address will match entry G. Note that the packet will not match entry H even though it contains an identical IP prefix and layer 2 address as entry G due to the different VLAN ID of 1004. This VTID indicates that the flow entry belongs to a different logical context. Flow table entry G will trigger an update of the packet's MAC header fields. In addition packets VLAN ID will be set to 300, corresponding to the TUID of the output virtual link, and the packet will be sent to DPX over the datapath link. Finally, at the accelerated datapath entry C will cause the packet to be forwarded over port P34 on which router R2 is attached.

ROUTING UPDATES We now consider the processing of a border gateway protocol (BGP) message, denoted `packet2`, advertising a new route to the virtual router hosted in VE3. We assume that the packet is associated with a pre-established BGP session with the virtual router instance. At DPX flow entry E matches all packets with an IP destination address corresponding to the address of the virtual interface TAP1, and the corresponding TUID of the virtual link. Hence, the BGP packet will be encapsulated inside an OF message and forwarded to the FMP through the control channel. There the message will be analyzed and forwarded to the associated VE (VE3) by the control traffic classifier based on its TUID. The VRC daemon in VE3 will strip the OF protocol encapsulation and inject the contents into `TAP1`. Assuming that the new route is accepted by BGP, the routing daemon will generate a routing table update in the VE. The VRC will receive a notification about the routing table

(a) DPX flow entries

#	Priority	Match rule	Action
input redirection flow entries			
A	0	VLAN_ID: 100	SET_VLAN_ID: 1003 OUTPUT_PORT: P_{DP0}
⋮			
output flow entries			
B	0	IN_PORT = P_{DP0} VLAN_ID: 200	OUTPUT_PORT: P_{21}
C	0	IN_PORT = P_{DP0} VLAN_ID = 300	OUTPUT_PORT: P_{34}
⋮			
cached routing flow entries			
D	108	VLAN_ID: 100 DST_IP = 20.33.0.0/16	SET_SRC_MAC: 00:12:34:00:00:33 SET_DST_MAC: 00:14:11:23:45:0a SET_VLAN_ID: 300 OUTPUT_PORT: P_{34}
⋮			
local control flow entries			
E	255	SET_VLAN_ID: 100 DST_IP = 10.1.10.1	TO_CONTROLLER
F	255	SET_VLAN_ID: 100 DST_MAC = ff:ff:ff:ff:ff:ff	TO_CONTROLLER
⋮			

(b) DP0 flow entries

#	Priority	Match rule	Action
routing flow entries			
G	108	IN_PORT = P_{DPX} VLAN_ID: 1003 DST_IP = 20.33.0.0/16	SET_SRC_MAC: 00:12:34:00:00:33 SET_DST_MAC: 00:14:11:23:45:0a SET_VLAN_ID: 300 OUTPUT_PORT: P_{DPX}
H	108	IN_PORT = P_{DPX} VLAN_ID: 1004 DST_IP = 20.33.0.0/16	SET_SRC_MAC: 00:12:34:00:00:33 SET_DST_MAC: 00:14:11:23:45:0a SET_VLAN_ID: 400 OUTPUT_PORT: P_{DPX}
⋮			

Table 5: Exemplary flow table entries in DP0.

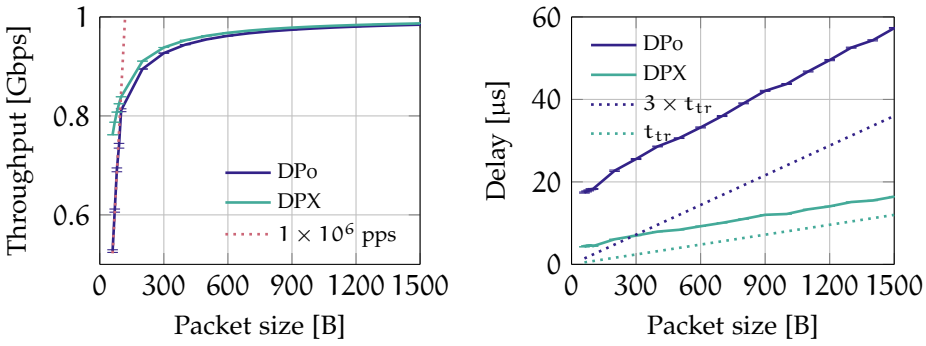
update, triggering the generation of an OF message containing a corresponding flow table entry (similar to flow entry G). This OF message will be forwarded to FMP over the control channel. Subsequently, the flow access control module, verifies whether VE3 is permitted to install the flow entry in DP0. Since the new flow entry contains a valid TUID the flow table is updated.

4.3.5 *Performance and Scalability Evaluation*

In the following, we evaluate several key performance metrics and discuss the scalability of the proposed architecture. We implemented a prototype of the root node using a commodity server with an Intel Nehalem CPU with four 2.27 GHz cores and 4 GB of RAM and a non-blocking Pica8 3290 OF switch with 48×1 Gbps. The virtualized control plane is hosted using the Linux KVM virtualization infrastructure [78]. The primary datapath (DP0) is implemented using the OpenVSwitch [114] software. The Pica8 switch acts as the accelerated datapath (DPX) of the system. The primary and accelerated datapaths are connected using four 1G datapath links.

We used NetFPGA [36] cards to generate the test traffic and an Endace DAG capture card to capture packet timings with nanosecond accuracy. First, we measured the achievable forwarding rates for traffic traversing the fast path (DPX) and the slow path (DPX $\rightarrow$ DP0 $\rightarrow$ DPX) of the virtual router platform. We generated 1 Gbps constant bit rate traffic with packet sizes ranging from 64 B to 1500 B. For each packet size we measured the rate at the egress of DPX for 100 s. Each experiment was executed 25 times. The averages of the obtained forwarding rates are depicted in Fig. 17a with 99% confidence intervals.

As expected, the accelerated datapath DPX forwards traffic at line rate regardless of the used packet size. On the other, traffic traversing the software datapath DP0 only achieves the maximal throughput for packets larger than 100 B. For small packets the forwarding rate is limited by the number of packets per second (pps) which can be processed by the software datapath. The measurements indicate that the maximum processing rate for OpenVSwitch is $\approx 1 \times 10^6$ pps. (in Fig. 17a this rate is indicated by a dotted line). Our results match the findings of [125] where several I/O techniques are proposed to improve the forwarding rate of OpenVSwitch. The Netmap platform developed in [126] achieves a forwarding rate of nearly 3 Mpps. GPU-supported platforms such as [68] have been shown to achieve forwarding rates of up to 15.6 Gbps with two 2.66 GHz quad-core CPUs. Most recently, prototype implementations of OpenVSwitch using Intel's DPDK platform [45] have claimed forwarding rates of up to 10 Mpps. Finally, our measurements show that with VLAN tagging, the multiplexing/demultiplexing operations required to encode the context identifiers do not have a notable impact on the forwarding performance.



(a) Throughput (b) Delay

Figure 17: Measured performance for the DP0 and DPX datapaths (99% confidence intervals).

To investigate the effect of the number of available CPUs on the packet forwarding capability of commodity servers we replace the Open-VSwitch datapath in the experiment above with a simple IPv4 forwarding module implemented using the Click Modular Router [83]. Our measurements showed that in this case a single CPU core is able to forward 1 Gbps of CBR traffic with 64 B packets. Moreover, we find that the forwarding performance scales linearly with the number of cores. As memory access latency is the main limitation of shared-memory architectures [48], the achievable throughput is expected to scale further as the number of non-uniform memory access (NUMA) nodes, which have dedicated memory controllers, is increased.

Finally, we measured the latency of the fast and slow paths of the system. We generated single packets and measured the arrival times at the ingress and egress of DPX, respectively. Sufficient time was left between runs in order to ensure that all system queues are empty. For each packet size the experiment was repeated 1000 times. The average latency for each packets size is depicted in Fig. 17b with 99% confidence intervals. For 64 B packets the forwarding operation takes approximately 4 µs over the fast path DPX. The latency increases linearly with the packet size with a slope corresponding to the transmission delay of the DPX port.

The minimal measured latency for a packet traversing the slow path over DP0 is 18.6 µs. Again we observe that this latency grows linearly as the packet size is increased. However the rate of increase is greater. This is due to the fact that, in addition to the PDX port, each packet must be transmitted over the datapath link twice. Hence, the total transmission delay for a packet with length l is given as $t_{tr} = \frac{l}{C_{DPX}} + \frac{2l}{C_{dl}}$, where C_{DPX} is the port capacity of the accelerated datapath and C_{dl} is the capacity of the datapath link. The theoretical transmission delays for the fast and slow paths are indicated by dotted lines in Fig. 17b, assuming 1 Gbps links.

In the sequel we discuss the scalability of the OpenVRoute architecture. The potential constraints of the platform are summarized below:

1. The available flow table size in the primary and accelerated datapaths and the number of flow table entries of auxiliary flow table entries which is installed by the FMP.
2. The rate at which new flows table entries can be generated by the FMP and the rate with which flow table modification are processed by the control logic of the datapaths.
3. The packet forwarding capability of the software switch at DP0 and the capacity of the datapath link.

First, we quantify the flow table space requirements. To this end we consider a root node which hosts N virtual router instances, where each corresponding VE $n \in N$, has M_n virtual interfaces and a routing table containing R_n entries. Furthermore, assume that the forwarding plane of the root node is connected to the substrate network over P physical ports on the accelerated datapath DPX.

The primary datapath DP0 must accommodate the routing flow entries ($\sum_{n=1}^{N} R_n$) for all virtual router instances. Given that current software OpenFlow switches support multiple flow tables with millions of entries each this requirement can be easily met.

The flow entries stored in DPX can be classified as static and dynamic entries. Static entries are responsible for redirecting traffic to and from the primary datapath over the datapath link. Such entries are created when a virtual router is instantiated and removed when the instance is taken offline. On the other hand, dynamic entries are comprised of cached routing flows that have a limited lifetime which depends on the statistical properties of the forwarded traffic.

As outlined in Section 4.3.4 the FMP installs one input and one output redirection flow, as well as a two local control flows for each instantiated virtual interface. Consequently the total number of static flow entries installed in DPX is $4 \sum_{n=1}^{N} M_n$ (the number is halved if the local control flows are hosted in the primary datapath). The number of entries increases linearly with the number of virtual interfaces. We expect that in a typical deployment this number will be in the order of several hundred entries. For the current generation of OF switches, this leaves several thousand flow entries for the installation of dynamic flow entries.

FLOW INSERTION RATE We evaluate the frequency at which flow entries may be cached in the accelerated datapath. To this end, we measure the *insertion* rate R at which the flow cache manager may install or modify flows in the datapaths. The measurement methodology used in the sequel is based on joint work with Amr Rizk [21], where we use the flow insertion rate as a metric to enable a consistent behaviour of SDN applications in heterogeneous SDN substrates.

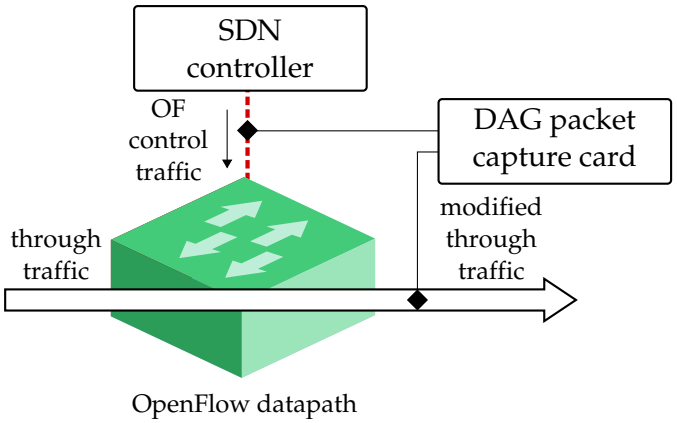


Figure 18: Experimental setup for the measurement of the flow insertion rate.

Our measurement setup is depicted in Fig. 18. It consists of a controller module, which mimics the functionality of the FMP, connected to a datapath. We use a NetFPGA card to inject a constant bit rate (CBR) traffic flow, denoted "through traffic", into the datapath and record the packet times at the ingress and egress of the datapath using an Endace DAG capture card. The test traffic consists of 64 B UDP packets with an inter packet interval of 10 µs. To obtain the maximum flow insertion rate the controller generates a burst of flow modification OF messages, set to increment a predefined header field of the through traffic. As a result, at the egress of the switch we can measure the intervals at which the packet header values were incremented to obtain the rate at which subsequent flow modification messages were processed by the datapath. Before the rate measurement is started we ensure that the TCP parameters of the control channel have converged to a steady state by generating a preliminary burst of control messages followed by a barrier request message which guarantees that the datapath processing queue is empty.

The flow insertion rates for DP0 and DPX over a 1 s interval are depicted in Fig. 19 with 99% confidence intervals (note that the confidence intervals are very small). The message processing rate of the software datapath is $R_{DP0} \sim 5700\,msg/s$. This rate is sufficient to support multiple concurrent virtual router instances. On the other hand, the rate of the accelerated datapath is significantly lower due to the limited processing capabilities of the switch control plane. Moreover, the tested switch exhibits two modes of operation. For $t > 0.37\,s$ the rate is $R_{DPX} \sim 1950\,msg/s$. As the number of elephant flows is expected to be small and the associated churn rate low [130] the reduced processing capability is sufficient for our architecture. We expect that flow insertion rates will increase further as the next generation OF switch implementations become available. We do not present measurements of the FMP control message generation rate since it exceeds the processing rate at the datapaths by several orders of magnitude. All major OpenFlow frameworks (e.g., NOX, OpenDaylight, Beacon), which may be used to

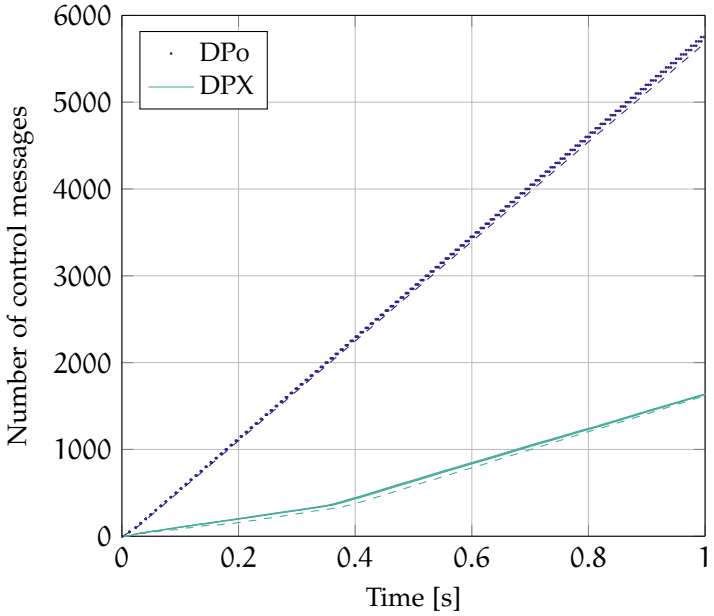


Figure 19: Number of control messages processed by the primary (DPo) and accelerated datapaths (DPX) over a one second interval. Dashed lines indicate 95% lower bounds.

implement the FMP, are capable of generating several millions of messages per second [35, 142].

Finally, we consider the scalability of the primary datapath. The performance of the commodity server hosting DP0 must be sufficient to forward the long tail of the flow distribution of aggregate Internet traffic. As discussed above, recent advancements in packet I/O performance [126, 125, 42] enable modern commodity servers to achieve throughput rates higher than 10 Gbps.

To quantify the requirements for DP0 we consider a trace captured on a 1 Gbps link of an access router of a residential ISP reported in [130]. The authors show that 100 prefixes account for more than 50% of the total traffic, whereas 1000 prefixes carry 80% of the total traffic. We extrapolate from these numbers and assume that 20% of the total traffic are "mice" flows which must be processed at DP0 while 80% of the traffic is forwarded at DPX. Assuming that the mice traffic is transported to DP0 over a 10 Gbps datapath link leaves 40 Gbps of "elephant" traffic to be processed at the accelerated datapath. For such a scenario a current generation OF switch with 48 1 Gbps ports and capable of storing 50 thousand flow table entries (~1000 flow entries for each physical port) is sufficient to handle the cached flows.

In closing, we note the resource requirements for hosting a virtualized control plane are minimal, as each VE only needs to process a relatively modes number of control traffic messages. Thus, the number of virtual router instances that our architecture can support depends largely on the specific volume an flow distribution of the carried traffic.

4.4 CONCLUSIONS

In this chapter, we evaluated the differences in the requirements for physical and virtual network topologies. We showed that in a virtual environment the traditional concept of a network topology is only meaningful if the corresponding VN specification mandates a suitable set of constraints. These constraints serve as guidelines for the development of a virtual network architecture. We showed that by offloading standard topology objectives, such as the provision of resilience or elasticity, to the physical domain the need for complex virtual network topologies may be eliminated. As a result, virtual networks may be specified in terms of a connectivity service between as set of geographical PoPs with specific QoS requirements. For scenarios where tenants desire full control of the network functionality infrastructure operators may instantiate arbitrary, programmable virtual topologies. In such a use case all optimization and resilience mechanisms are handled by the tenant, while the InP guarantees the isolation of the allocated resources as well as 1:1 mapping between the virtual and physical resources.

We implemented an infrastructure virtualization platform which supports both of these use cases. To this end, we evaluated techniques which enable the SDN hypervisor to bind physical resources to a corresponding virtual context. Specifically, we showed that the use of tunnel identifiers, virtual flow table identifiers and translator functions serves as a versatile mechanism for encoding context identifiers. This outline mechanism enables the SDN hypervisor to conceal the virtualization layer from the virtual network tenants, while allowing the installation of arbitrary flow entries by tenant controllers. Additionally we discussed mechanisms for resource migration and addressed the scalability of the hypervisor layer by segmenting the SDN substrate into independent domains managed by controller proxies.

We evaluated the capacity allocation costs for the special case of connectivity services. Assuming that VN requests are formulated as capacity demands at each tenant attachment point we showed that rooted tree topologies yield the minimal capacity allocation costs. We showed that a further reduction of the reservation is only possible if the exact point-to-point traffic requirements are known. However, in general such demands are difficult to specify a priori. We showed that if the traffic demands between two tenant PoPs are known, e.g., estimated through measurements, a hybrid embedding scheme which forwards individual flows over the shortest paths may be applied to reduce the reservation costs.

Finally, we presented a virtual router architecture as an example of a layer 3 connectivity service. The proposed OpenVRoute platform combines off-the-shelf components to offer tenants virtual router instances which are functionally and logically indistinguishable from traditional routers. We addressed the memory limitation in current generation SDN devices, by exploiting the Zipf distribution of aggregate traffic flows.

Specifically, we proposed a dual datapath approach which enables us to take advantage of the abundant memory resources in commodity servers and the high packet forwarding rates and port density of dedicated switching hardware. The proposed architecture relies on readily available, open technologies, and provides a template for the implementation of more specialized connectivity services. We expect that the presented concepts are flexible enough to integrate emerging software, hardware and protocol developments. For example a modification of the platform to implement an OF-enabled switch abstraction is straightforward. Moreover, we expect that the versatility of virtual connectivity services will be further enhanced through long term OF developments such as support for user-defined packet matching and processing. Finally, the scalability of the architecture will benefit from the advent of OF-enabled switches with larger flow table sizes, enabling higher levels of resource consolidation.

The results presented in the area of topology embedding highlights the necessity for an accurate view of the network traffic demands. A tight characterization of the actual capacity requirements of a virtual link enables InPs to better utilize the available physical resources. Thus, we dedicate the following chapter to the design of techniques for extracting relevant flow characteristics from network measurements.

5

PERFORMANCE EVALUATION IN CENTRALIZED NETWORK ARCHITECTURES

5.1 MOTIVATION

Software defined networks are based on the concept of a separation of the control and data planes, wherein a logically centralized controller instantiates the forwarding logic of a pool of forwarding devices. Consequently, SDN network services operate using a global network view generated by the controller framework. This implies that the controller must possess a detailed and up-to-date representation of the state of the substrate network. Given this global view, SDN enables a fine-grained, automated optimization of network operations.

A major benefit of the SDN approach is the abundant availability of processing resources in the centralized control plane which is typically hosted on high performance commodity servers. We believe that these resources should be exploited to perform complex evaluations of monitoring data collected by an SDN controller, which can be beneficial to the services deployed on top of SDN environments. To this end, we propose a monitoring framework which automates the derivation of statistical characteristics of network flows. The framework is embedded in the SDN control plane enabling an augmented global view of the substrate network. The extracted information may be leveraged by SDN applications to perform admission control or compute optimized forwarding policies which improve QoS or the utilization of the available network resources.

In the previous chapter we presented approaches for virtualizing full topologies and network services in an SDN environment. Topology embedding and optimization strategies, e.g., using live-migration, rely on an accurate description of the tenant traffic requirements as well as a comprehensive view of the current utilization of the physical resources. In practice both requirements are difficult to fulfill. In the case of embedding, a static allocation strategy in which fixed capacities are reserved for each link in the virtual topology may be used to provide bandwidth and QoS guarantees. To this end, a suitable scheduling discipline is configured at the corresponding egress switch interfaces. However, such an approach leads to a poor utilization of the substrate resources as multiplexing gains, which are a key benefit of packet switched networks, are not exploited. In other words, the strategy does not scale well for scenarios where the substrate network must accommodate a large number of VN requests. This problem is aggravated by the fact that tenants typically cannot accurately predict their capacity requirements in advance. Therefore, the specified vSDN demands may exceed the actual traffic

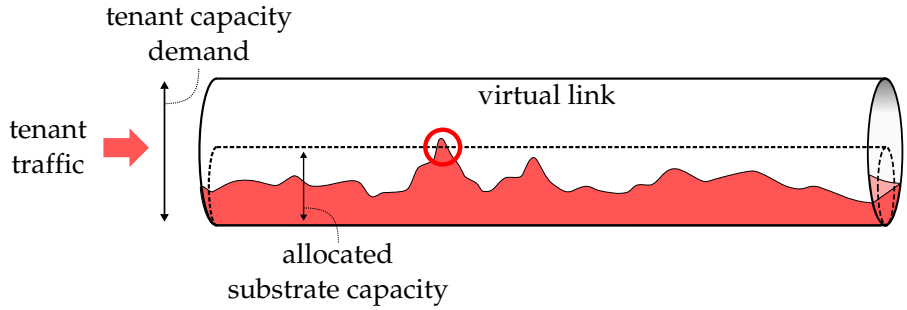


Figure 20: Virtual link with requested and allocated capacity. The operator is interested to determine the probability that the tenant traffic will exceed the allocated substrate capacity (indicated by circle above).

rate at the corresponding PoP by some over-provisioning factor. On the other hand, an over-subscription of the infrastructure resources necessitates an accurate characterization of the network utilization to avoid a degradation of the perceived VN performance or a violation of negotiated SLAs.

In this chapter we propose a monitoring framework which extracts flow characteristics at key network points with the goal of aiding SDN controller algorithms with the dimensioning of the resource allocation. A conceptual illustration of one use case for our methodology is depicted in Fig. 20. Consider a tenant virtual link at an ingress PoP with a requested capacity demand D. The actual traffic generated by the tenant is significantly lower than the requested capacity. In the substrate, an SDN controller may allocate a lower capacity D^* for this virtual link, such that the probability that the tenant flow will exceed D^* remains below a certain threshold. Similarly, the controller may asses the impact of allocating a smaller virtual link capacity on the queueing delay experienced by the tenant *before* performing the actual allocation.

It is well known that network traffic is bursty and exhibits strong correlations [85]. The adverse impact of these properties on the network performance, specifically the provided quality of service, has been shown in empirical and theoretical studies, e.g., [106, 51]. Bursty network traffic results in the buildup of large queues on the traversed network devices and consequently leads to large, highly variable latencies. To mitigate the effects of LRD traffic network operators must over-provision the network capacities. In modern networks (e.g., data centers) queueing delays are emerging as a primary target for network optimization [113].

The goal of this chapter is to enable the characterization of the impact of salient traffic flows on the network service using monitoring data collected from network switches by a logically centralized controller. As a result, the SDN control plane may generate a detailed view of the processes occurring inside a network which may be utilized by the SDN application layer to optimize the placement of flow routes within the network substrate. Specifically, we aim to equip the SDN control plane

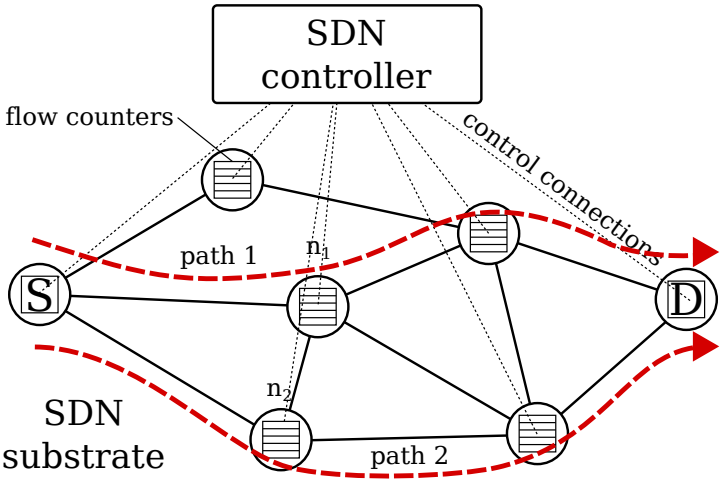


Figure 21: SDN monitoring scenario: a controller queries flow counters across a set of forwarding devices. The extracted performance metrics are exposed to interested SDN applications.

with the ability to monitor the correlation structure of traffic flows in addition to traditional metrics such as utilization. In the sequel, we provide the means for extracting relevant flow statistics in SDN with minimal intrusiveness using random sampling. In addition, we outline a suitable performance evaluation technique which exploits the collected information.

In the following, we assume a typical SDN architecture, depicted in Fig. 21, comprised of a centralized controller that monitors individual flows (or collections of flows) on all connected switches using either packet counters or sampled packets. The controller uses the collected data to extract per-flow performance metrics, such as traffic intensity, variance or queueing delays. In the sequel, we assume that virtual links are allocated a fixed capacity on all interfaces of the corresponding substrate path.

The example in Fig. 21 depicts an SDN application which uses the derived metrics to evaluate the effects of a virtual link migration between two network paths with respect to some prescribed QoS threshold.

To reduce the amount of monitoring data which must be collected in the network we rely on random sampling. Therefore, we require sampling strategies which guarantee that the extracted traffic characteristics are not distorted. We note, that while the presented work focuses on the SDN use case, the techniques outlined in the following are applicable to any centralized monitoring architecture which provides fine-grained, randomized querying mechanisms of individual network flows.

The chapter is structured as follows: In the next section we provide some background on key network traffic characteristics and describe the notation used in the sequel. In Section 5.3 we review strategies used for deriving QoS bounds from traffic traces. In Section 5.4 we present an analytical evaluation of random packet sampling for extracting the

traffic autocovariance as well as estimating the Hurst parameter which characterizes LRD traffic. In Section 5.5 we provide similar results for random sampling of flow counters, highlighting the inherent differences between the two approaches. Section Section 5.6 presents our approach for generating independent sample paths from the extracted autocovariances. We consider traffic with both Gaussian and non-Gaussian traffic increments and present results that verify our findings. We conclude the chapter with a controller strategies for sampling across multiple nodes and flows in Section 5.7.

5.2 NETWORK TRAFFIC CHARACTERISTICS: BACKGROUND AND NOTATION

We begin with a review of several key statistical properties of network flows used in the following sections and a description of the used notation.

Throughout this work we model network flows as discretized, wide sense stationary, stochastic increment processes, i.e., the mean and autocorrelation function of the flow are time invariant. For a given flow we denote the intensity of the traffic increments at time t, i.e., the amount of data within a discrete time slot with a length δ, as $X(t)$ for $t \in \mathbb{N}_0$. Where applicable, we represent the random process $X(t)$ as a column vector $\mathbf{x} = (x_1, x_2, \ldots, x_t)^T$ with random elements denoted $x_t = X(t)$. In the sequel, we make a distinction between the stationary increment process $X(t)$ and the corresponding cumulative process $\overline{X}(t) = \sum_{k=0}^{t} X(k)$ (similarly we denote the cumulative vector process $\overline{\mathbf{x}}$).

A primary metric for resource allocation is the mean rate of the traffic flow characterized by the expected value $E[X(t)]$. In the following, we frequently use $\mu_{(\cdot)}$ to indicate the mean rate $E[(\cdot)]$ of a process. For a link with a fixed capacity C and carrying traffic process $X(t)$ the residual capacity $C - \mu_X$ is commonly referred to as the available bandwidth [134].

LONG RANGE DEPENDENT TRAFFIC PROCESSES. In the last two decades, numerous empirical measurement studies have shown that network traffic exhibits properties of self-similar processes [85, 93]. Self-similar stochastic processes may be strongly correlated, meaning that the traffic load at any point in time is dependent on the traffic intensity of a large number of previous values. Moreover, the statistical properties of self-similar processes persist, to a degree, regardless of the time-scale at which the process is observed. This so-called LRD, which has been observed, e.g., in [85, 67], manifests itself in the burstiness of Internet traffic. Network measurements and results from queueing theory have shown that strongly correlated traffic has an adverse effect on network performance [51, 106, 121].

As a result, a main focus of this thesis lies on the extraction of the correlation structure of salient network flows. The autocovariance of a stationary, stochastic process $X(t)$ is defined as $E[X(t)X(t+\tau)] - \mu_X^2$ for

$\tau \in [0, \infty)$. In the following we use the notation $c_{(\cdot)}(\tau)$ to denote the autocovariance of a process $(\cdot)$ at lag τ, and $\sigma^2_{(\cdot)} = c_{(\cdot)}(0)$ to denote its variance. In addition, we denote the square, positive definite autocovariance *matrix* of the process as $\mathbf{\Sigma}_{(\cdot)}$, where each matrix element at row i and column j is denoted as $\Sigma_{(\cdot)ij}$. For any stationary increment process the covariance matrix $\mathbf{\Sigma}$ possesses a Toeplitz structure and is specified fully by the process autocovariance as $\Sigma_{ij} = c_{(\cdot)}(|i-j|)$. We denote the autocovariance matrix of the cumulative process $\overline{X}(t)$ as $\overline{\mathbf{\Sigma}}$.

A stochastic process is said to be LRD if its autocovariance function exhibits a slower than exponential decay for increasing lags τ. In the context of performance evaluation, fractional Brownian motion (fBm) is a continuous-time stochastic process widely used to model the cumulative arrivals of network traffic. The corresponding increment process, known as fractional Gaussian noise (fGn), has Gaussian increments with an autocovariance function

$$c_X(\tau) = \frac{\sigma^2}{2}[(\tau+1)^{2H} - 2\tau^{2H} + (\tau-1)^{2H}], \tag{3}$$

where $H \in (0.5, 1)$ is the so-called Hurst parameter which characterizes the decay of the autocovariance function and σ^2 is the process variance. For large τ the relationship above is commonly approximated using a Taylor expansion as

$$c_X(\tau) \sim (2H-1)H\sigma^2\tau^{2H-2} \sim \tau^{2H-2}. \tag{4}$$

From Eq. (4) it follows that $\sum_\tau c_X(\tau) = \infty$ for $H > 0.5$.

A discrete time family of LRD processes with similar properties is described by autoregressive fractionally integrated moving average (ARFIMA) models [85]. In the sequel, we use the term LRD traffic to refer to flows which exhibit an autocovariance structure given by Eq. (4).

QUEUE LENGTH DISTRIBUTION. In this work we consider queueing delay as a key metric for characterizing network performance. Queueing delays are caused by buffered data within network devices, i.e., backlog. For any time t the backlog of a lossless system with FIFO scheduling is defined as the difference between the cumulated data arrivals up to time t $A(t) = \overline{X}(t)$ and departures $D(t)$ at that system up to time t

$$B(t) = A(t) - D(t).$$

Hence, for a work conserving, buffered link with constant capacity C the time at which the last bit at the end of the queue leaves the buffer is given as $B(t)/C$. For a wide range of applications it is desirable to specify the buffer occupancy in terms of a distribution, i.e., the probability ε that the buffer occupancy exceeds some threshold b, or $\mathbb{P}[B(t) > b] = \varepsilon$.

In this thesis, we study the estimation of backlog and delay distributions at a single SDN switch. Our main objective is the derivation of practical methods for the estimation of traffic characteristics necessary to

quantify the impact of a given traffic mix on queueing delay. We expect that future SDN applications seeking to optimize network performance will build upon our findings and utilize the large body of theoretical work dealing with performance evaluation of multi-hop systems.

In the sequel, we consider bounds [31] of the form $\mathbb{P}[B(t) > b] \leqslant \varepsilon(b)$ on the tail decay of the backlog distributions B and as $\mathbb{P}[W > d] \leqslant \varepsilon(d)$ for the virtual delay W. Further, we characterize the buffered links at any substrate switch using the well-known concept of exact service curves $S(t)$ [31, 13]. For a given interface the function $S(t)$ describes the relationship between the cumulative traffic arrivals $A(t)$ and departures from the system $D(t)$ over a time interval t. Specifically, we use a latency-rate server model, in which incoming data is delayed by a constant latency term T (e.g., processing delay) and subsequently processed with the output rate of the link C, i.e., $S(t) = C[t-T]_+$ with $[x]_+ = \max\{0, x\}$. We assume that the switch service curve is determined through measurements [24] or provided by the switch vendor.

A fundamental result from network calculus [13] states that the departures $D(t)$ from a system characterized by a service curve $S(t)$ are given by the min-plus convolution of the arrivals $A(t)$ and $S(t)$, i.e., $D(t) \geqslant (A \otimes S)(t) = \inf_{0 \leqslant \theta \leqslant t}\{A(\theta) + S(t-\theta)\}$. Substituting this relationship into the definition of the backlog yields

$$\begin{aligned} B(t) &\leqslant A(t) - \inf_{0 \leqslant \theta \leqslant t}\{A(\theta) + S(t-\theta)\} \\ &= \sup_{0 \leqslant \theta \leqslant t}\{A(t) - A(\theta) - S(t-\theta)\}. \end{aligned}$$

Note, that for so-called exact service curves, such as the latency rate model used in this work, the inequality above holds with equality [54]. As a result, a bound on the backlog distribution is given by

$$\mathbb{P}[B(t) > b] = \mathbb{P}\left[\sup_{0 \leqslant \theta \leqslant t}\{A(t) - A(\theta) - S(t-\theta)\} > b\right]. \tag{5}$$

Note, that given a bound $\mathbb{P}[B(t) > b] = \varepsilon(b)$ for Eq. (5), a corresponding bound for the virtual delay $\mathbb{P}[W > d] = \varepsilon(d)$ follows analogously [54] since we assume constant rate links. Extensions for more involved link models using the notion of service curves [27, 54] are readily obtained from the related work.

We highlight an alternative representation of Eq. (5) which we use in the following sections. Consider that the event $\sup_{0 \leqslant \theta \leqslant t}\{A(t) - A(\theta) - S(t-\theta)\} > b$ occurs only when at least one of the events $\{A(t) - A(\theta) - S(t-\theta) > b\}$ with $\theta \in [0, t]$ occurs. Therefore an equivalent formulation of Eq. (5) which uses the union of these events is

$$\mathbb{P}[B(t) > b] = \mathbb{P}\left[\bigcup_{\theta \in [0,t]}\{A(t) - A(\theta) > S(t-\theta) + b\}\right]. \tag{6}$$

Throughout this work we assume that the increment process associated with the arrival process $A(t)$ is stationary. Therefore, we may rearrange $\mathbb{P}[A(t) - A(\theta) \leqslant k] = \mathbb{P}[A(t-\theta) \leqslant k]$ and substitute $\eta = t - \theta$ to obtain

$$\mathbb{P}[B(t) > b] = \mathbb{P}\left[\bigcup_{\eta \in [0,t]} \{A(\eta) > S(\eta) + b\}\right] \tag{7}$$

$$= 1 - \mathbb{P}\left[\bigcap_{\eta \in [0,t]} \{A(\eta) \leqslant S(\eta) + b\}\right]. \tag{8}$$

APPROXIMATIONS OF THE QUEUE LENGTH DISTRIBUTION It has been shown [12] that backlog and delay distributions for general arrival traffic distributions and general service distributions are notoriously hard to analyze exactly. In the context of performance evaluation, a number of works, e.g., [31, 34, 89], analyze the backlog and delay under various assumptions. In this work we are interested in practical approximations of the probability $\mathbb{P}[B(t) > b]$ in Eq. (5). As this probability is difficult to evaluate analytically, a commonly applied approximation is obtained by changing the order of the supremum and probability operators. The resulting expression

$$\sup_{0 \leqslant \theta \leqslant t} \{\mathbb{P}[A(t) - A(\theta) - S(t-\theta) > b]\}, \tag{9}$$

is known as the max-approximation [120] of the queue length distribution. Again we use the stationarity of the increment process to rewrite Eq. (9) as

$$\sup_{0 \leqslant \theta \leqslant t} \{\mathbb{P}[A(\theta) > S(\theta) + b]\}. \tag{10}$$

Intuitively, the above approximation yields a point-wise violation probably, i.e., the maximum likelihood that a given sample path will exceed the buffer threshold for some specific point in time. We denote point-wise probability in Eq. (10) as $\mathbb{P}[B(t) > b]_{pw}$. In contrast, Eq. (5) specifies the probability that the sample path will exceed the threshold at least once over *all* points in time $0 \leqslant \theta \leqslant t$. From the fact that the max-approximation considers a single point in time, while Eq. (5) yields the violation probability for the entirety of points in time, it follows that $\mathbb{P}[B(t) > b]_{pw}$ is a lower bound for $\mathbb{P}[B(t) > b]$, i.e.,

$$\mathbb{P}[B(t) > b]_{pw} \leqslant \mathbb{P}[B(t) > b].$$

In other words, the resulting approximation of the buffer overflow probability is too optimistic.

GAUSSIAN INCREMENTS We now consider the cumulative arrivals of a zero mean[1] random traffic process with Gaussian increments, e.g., fGn. The corresponding cumulative arrival process, denoted $A(t)$, is fBm.

1 Here, we assume that the expected value is known and has been subtracted from the traffic process.

We model $A(t)$ as a random vector $\overline{\mathbf{x}}$ with a multivariate normal distribution, i.e., the vector elements $\overline{\mathbf{x}} = (\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_t)^T$ are the cumulative traffic increments and each element $\overline{x}_\theta$ for $\theta = [1, t]$ is normally distributed with variance σ^2_θ. Further, let $\overline{\mathbf{\Sigma}}$ denote the autocovariance matrix of the cumulative process $\overline{\mathbf{x}}$ with $\overline{\Sigma}_{ii} = \sigma^2_\theta$. The density function of the multivariate process $\overline{\mathbf{x}}$ is given by

$$f_{\overline{\mathbf{x}}}(\mathbf{x}) = f_{\overline{\mathbf{x}}}(x_1, x_2, \ldots, x_t) = \frac{1}{\sqrt{|\overline{\mathbf{\Sigma}}|(2\pi)^t}} e^{-\frac{1}{2}\overline{\mathbf{x}}^T \overline{\mathbf{\Sigma}}^{-1} \mathbf{x}}. \tag{11}$$

Consequently, to obtain $\mathbb{P}[B(t) > b]$ we define the threshold vector $\mathbf{r}$ with elements $r_\theta = S(\theta) + b$ for $\theta = [1, t]$ and write Eq. (7) as

$$\mathbb{P}[B(t) > b] = 1 - \mathbb{P}\left[\bigcap_{\theta \in [0,t]} \{\overline{x}_\theta \leqslant r_\theta\}\right] = 1 - F_{\overline{\mathbf{x}}}(\mathbf{r}),$$

where $F_{\overline{\mathbf{x}}}(\mathbf{r})$ denotes the cumulative distribution function $F_{\overline{\mathbf{x}}}(\mathbf{r}) = F_{\overline{\mathbf{x}}}(r_1, r_2, \ldots, r_t)$ of the arrival vector $\overline{\mathbf{x}}$. Therefore, for traffic processes with normally distributed increments we may express the queue length distribution as the complementary cumulative distribution function (CCDF) of a multivariate Gaussian random variable:

$$\mathbb{P}[B(t) > b] = 1 - \frac{1}{\sqrt{|\overline{\mathbf{\Sigma}}|(2\pi)^t}} \int_{-\infty}^{r_1} \int_{-\infty}^{r_2} \cdots \int_{-\infty}^{r_t} f_{\overline{\mathbf{x}}}(\mathbf{x}) d\mathbf{x}. \tag{12}$$

Unfortunately, no analytical solution is known for the relationship above. Therefore, numerical integration approaches, e.g.[60], are used to obtain an approximation of Eq. (12).

To obtain the point-wise violation probability in Eq. (10) we evaluate the likelihood of the event that the arrivals $A(\theta)$ at time θ exceed the threshold $S(\theta) + b$.

$$\mathbb{P}[A(\theta) > S(\theta) + b] = 1 - \Phi\left(\frac{S(\theta) + b}{\sigma_\theta}\right) \tag{13}$$

where Φ denotes the cumulative distribution function (CDF) of the standard normal distribution. Next we seek the time $\theta^* \in [1, t]$ at which this event is most likely to occur by setting the derivative of Eq. (13) to zero and solving for θ.

A widely used approximation of this relationship was derived in [105, 106] using Chernoff's bound to approximate the tail distribution in Eq. (13) as $1 - \Phi(x) \leqslant \frac{1}{2} e^{-x^2/2}$. Assuming an fBm traffic process with Hurst parameter H, variances $\sigma^2_t = \sigma^2 t^{2H}$, link capacity C and a mean rate μ and solving yields the following closed form expression [105]

$$\mathbb{P}[B(t) > b]_{pw} \approx \exp\left(-\frac{1}{2\sigma^2}\left(\frac{C - \mu}{H}\right)^{2H}\left(\frac{b}{1 - H}\right)^{2-2H}\right). \tag{14}$$

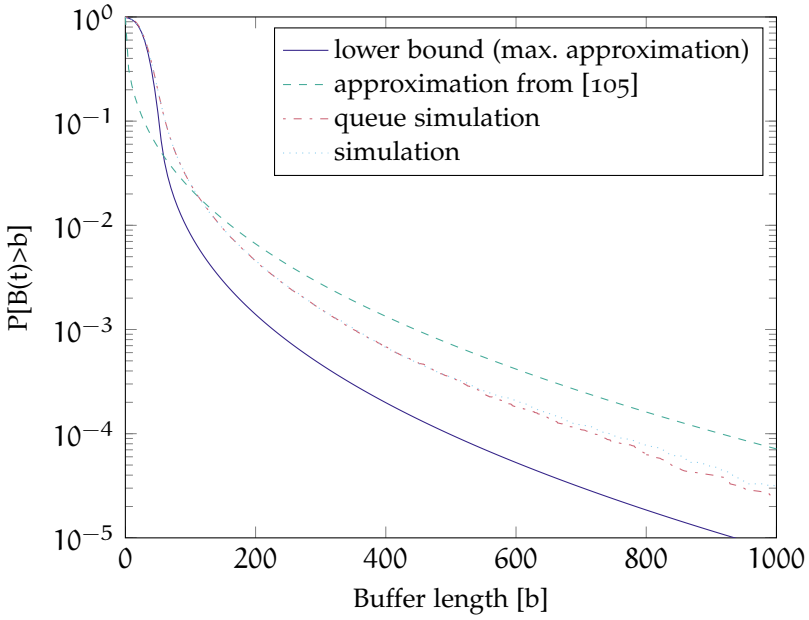


Figure 22: Comparison of approaches for estimating the complementary cumulative distribution function (CCDF) of the buffer occupancy.

This formulation shows the fundamental impact of fBm traffic on the queue violation probability. For LRD traffic with $H > 0.5$ it follows from the relationship above that the queue length distribution exhibits a Weibull tail behavior.

The differences between the approaches for estimating the queue length distribution are depicted in Fig. 22. We generated synthetic fBm traffic traces with Hurst parameter $H = 0.8$, mean rate $\mu = 6.6$ data units per unit time (dpt) and variance $\sigma^2 = 16$. Each sample path had a length of 4000 time units. We assumed a latency rate server with a link capacity of $C = 20\,\mathrm{dpt}$ and a latency of 5 time units. In Fig. 22 the curve denoted "lower bound" represents the point-wise violation probability calculated by taking the maximum value of Eq. (13) over the interval $[1, t]$. Next, the "approximation from [105]" was calculated using Eq. (14). The curve denoted "simulation" was derived by approximating the multivariate integral Eq. (12) using [60]. Finally the curve denoted "queue simulation" was obtained from a packet level queue simulation. Clearly, the two simulation approaches yield very similar results, both of which lie above the lower bound estimate. We emphasize that both simulation approaches converge to the true queue length distribution only in the limit for a large number of repetitions and for $t \to \infty$. The approximation obtained from Eq. (14) lies above the simulation results for buffer sizes $b > 200$ (note, that Eq. (14) assumes a constant rate server with capacity C).

Finally, we highlight the importance of a sufficiently high sampling rate for the accuracy of the CCDF estimate. Using the synthetic traces outlined above, we evaluate the queue length distribution for increasing

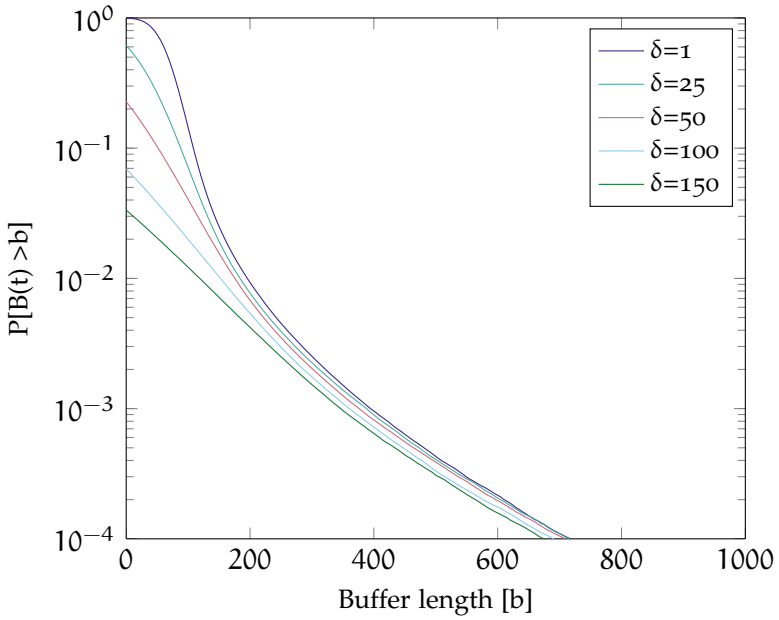


Figure 23: Effect on sampling interval δ on the CCDF estimate. Increasing the time slot length yields to a loss of precision.

time slot durations $\delta \in \{1, 10, 100\}$. The results are depicted in Fig. 23. Evidently, the estimate becomes increasingly distorted as the slot length is increased. Intuitively this loss of accuracy results from the unknown behaviour of the flows at small time scales. This example motivates the use of random sampling in this work, which reduces the sampling intensity while preserving the high resolution of the autocovariance structure.

5.3 PERFORMANCE EVALUATION STRATEGIES

The goal of this work is to provide methods for performance evaluation and optimization of resources in SDN networks. We envision an SDN controller component that collects fine grained flow statistics from data counters and computes corresponding characteristic flow metrics. We seek to enhance the controller's ability to make decisions which affect the performance of a given switch by including flow correlation information in the performance evaluation procedure. As a consequence, the controller can compute performance metrics such as backlog and virtual delay distributions and incorporate these into optimization algorithms. These calculations are carried out for designated switch interfaces where salient flows are multiplexed.

In this section we aim to empirically extract an approximation of the QoS level ε from measurements. To this end, we consider two possible strategies, which we denote *parametric approximation* strategy and *simulation* strategy, respectively.

In the first strategy we utilize the approximation Eq. (14) from [105, 106] and assume that the salient network traffic flows may be modelled as fBm processes. Consequently, each flow may be characterized by its mean rate, variance and Hurst parameter. Then, we obtain an approximation of the queue length distribution by estimating the flow parameters from measurements and substituting the estimates into Eq. (13). The main advantage of this approach is that it is lightweight and only requires estimates of the flow variance, mean rate and the Hurst parameter H. We provide approaches for estimating H in the following. However, the obtained estimate may be inaccurate due to the approximations used in the derivation of Eq. (13). Moreover, the fBm traffic model may not accurately capture the properties of the carried traffic.

On the other hand, the simulation strategy aims to achieve a more accurate estimate of the queue length distribution while minimizing the a priori assumptions about the structure of the observed traffic process. Specifically, we assume that each traffic flow possesses an arbitrary autocovariance structure $c(\tau)$ which is known over some range of lags $\tau \in [0, t]$. Further, we assume that the expected value and the variance of the process exist. Initially we assume that the process increments are normally distributed. We relax this assumption in Section 5.6.4.

For this scenario we make use of the queue length distribution as defined in Eq. (7):

$$\mathbb{P}[B(t) > b] = \mathbb{P}\left[\bigcup_{\theta \in [0,t]} \{A(\theta) > S(\theta) + b\}\right].$$

Intuitively the formulation above states that the buffer overflow probability is equal to the probability that a given sample path of traffic arrivals $A(\theta)$ will exceed the threshold $S(\theta) + b$ at *any* single point in time $0 \leqslant \theta \leqslant t$. Hence, it follows that an estimate of ε may be obtained by evaluating a large number of arrival sample paths and counting how many sample paths exceeds the function $S(\theta) + b$ for some point $0 \leqslant \theta \leqslant t$. Note, that due to the finite length of the sample paths the approximation may underestimate the true buffer overflow probability.

In order to realize this approach our controller framework includes a module which applies a general Monte Carlo (MC) approach to estimate the queue length distribution. Monte Carlo estimators utilize a large number of independent samples to obtain an unbiased estimate of the evaluated metric. For our use-case each sample consists of an independent traffic arrival sample path. However, in any practical scenario only a single sample path is observed for any given traffic flow. To circumvent this limitation we extract the statistical properties of the observed traffic process and use these to synthesize an arbitrary number of independent sample paths exhibiting identical statistical properties (if possible). We outline the details of the approach which is based on matrix factorization technique in Section 5.6. For now, we highlight that the key prerequisite for the factorization step is an estimate of the auto-

covariance matrix of the considered traffic flow. In summary, the simulation approach outlined in the sequel comprises the following steps:

1. Random sampling of packets/flow counters and estimation of the flow autocovariance matrix.
2. Cholesky decomposition of the estimated covariance matrix and generation of synthetic sample paths with identical covariance properties.
3. Monte Carlo estimation of the queue length distribution based on the generated sample paths and Eq. (6).

The main contribution of this work is the analysis of methods for extracting the process autocovariance from (random) observations. The covariance structure is used as a basis for deriving an estimate of the Hurst parameter for the parametric approximation strategy, as well as to construct the autocovariance matrix required for the simulation strategy. In addition, we aim to reduce the amount of observations necessary to obtain an unbiased autocovariance estimate through random sampling. We aim to minimize the monitoring traffic load and the associated processing overhead at the switch and the controller while maintaining a sufficient estimation quality.

In the following section we discuss two approaches for estimating the traffic correlation structure of salient flows using a centralized SDN controller. First, we evaluate packet sampling in Section 5.4. Subsequently Section 5.5 we extend the approach to observations of flow counters exposed in SDN switches.

5.4 PACKET SAMPLING

Given the continuous increase of network capacities, sampling has emerged as an essential technique for monitoring a wide range of traffic metrics. The infrastructure necessary to transport, process and store unsampled monitoring data at a central vantage point is typically unfeasible due to the vast volume of traffic traversing a network segment.

To mitigate this issue network device vendors and network operators rely on technologies which use some form of sampling to efficiently capture a representative subset of the network traffic. An example of widely deployed monitoring technology is *sFlow*, which uses random sampling as a primary means for ensuring scalability. Devices which implement sFlow randomly select individual packets from the aggregate traffic flow and forward these to a central server, called an sFlow collector, for further analysis[2]. Such an approach integrates seamlessly into the SDN framework where the collector may be implemented as an SDN controller component.

2 sFlow devices can also generate counter samples, which are discussed in Section 5.5.

In this section we address the question of how packet samples, collected using sFlow-like technology, may be used to obtain unbiased estimates of the covariance structure of a given flow. Specifically, we show that the autocovariance of a traffic flow estimated from packet samples differs from the true autocovariance of the traffic flow and depends on the specific sampling strategy utilized at the switch. We demonstrate that this "observed" covariance structure contains a systematic error, which can be corrected under specific conditions. Furthermore, we investigate the accuracy of covariance estimates for finite sample sizes. In addition, in Section 5.4.3 we show that for fBm traffic the Hurst parameter can be accurately estimated even for small sample sizes using an aggregation of the values of the autocovariance function.

The results presented in this section are partly based on joint work with A. Rizk and M. Fidler [123, 124].

5.4.1 *Estimating the Flow Autocovariance from Packet Samples*

In the following we model the packets sampled randomly from a given flow using three stationary, discrete time processes. We consider the monitored flow as an increment process $X(t)$, discretized at some fixed time slot δ with $t \in \mathbb{N}_0$. Next, we model the strategy employed by the monitoring system to select whether a packet is sent to the central server as a sampling process denoted $A(t)$. The sampling process $A(t)$ is a point process taking the value of one whenever a sample is generated, and zero otherwise. We assume that this process is statistically independent from the traffic process $X(t)$. Furthermore, we assume that the times between subsequent samples are independent and identically distributed (i.i.d.), and are drawn from a known probability distribution F_A. We denote the mean rate, or sampling intensity, of the sampling process $A(t)$ by $\mu_A = E[A(t)]$. Note, that $\mu_A \in [0, 1]$ because $A(t)$ contains only binary values.

Finally, we define the random process observed at the sample collector as

$$W(t) = A(t)X(t). \tag{15}$$

The process contains the values of the monitored process $X(t)$ for all times at which a sample was generated, and zeros otherwise.

In the following we use the observed process $W(t)$ to infer the autocovariance of the underlying traffic process X which is not directly observable. However, the extraction of the autocovariance of $X(t)$ from $W(t)$ is not straightforward, as the distribution of the sampling process distorts the covariance structure observed at the collector. Specifically, the impact of the sampling process on the autocovariance of the observed process $c_W(\tau)$ is shown by the following lemma:

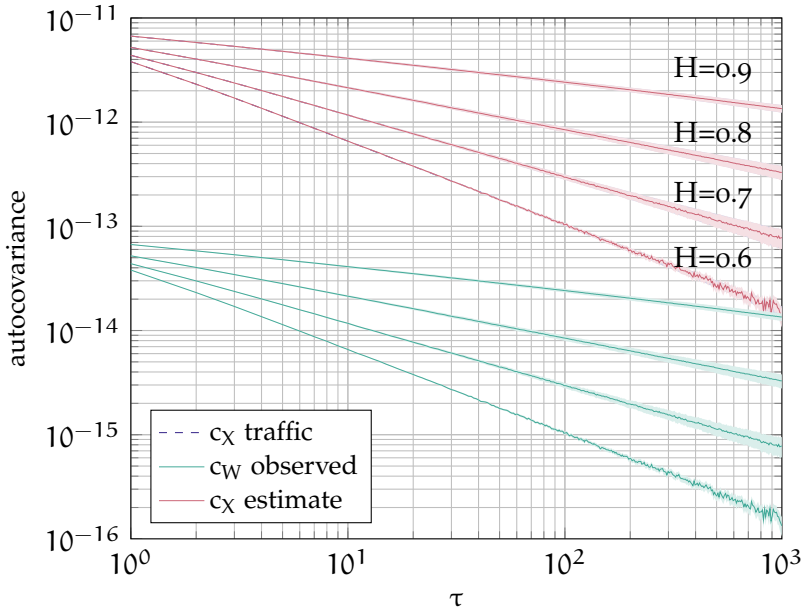


Figure 24: Observed and reconstructed autocovariance of a LRD traffic processes under geometric sampling. The observed "$c_W(\tau)$" maintains the autocovariance structure of the traffic process. The covariance of the original process "$c_X(\tau)$ (traffic)" is exactly covered by the reconstructed "$c_X(\tau)$ (estimate)".

Lemma 5.1. *Given the stationary and independent stochastic processes,* $A(t)$, $X(t)$ *and* $W(t) = A(t)X(t)$, *the autocovariance of* $W(t)$ *is comprised of the autocovariances of the processes* $A(t)$ *and* $X(t)$ *with*

$$c_W(\tau) = \big(c_A(\tau) + \mu_A^2\big)c_X(\tau) + c_A(\tau)\mu_X^2.$$

Proof. Consider the independent and stationary processes $A(t)$ and $X(t)$. For $W(t) = A(t)X(t)$ it follows that

$$\begin{aligned} c_W(\tau) &= \mathrm{E}[A(t)A(t+\tau)]\,\mathrm{E}[X(t)X(t+\tau)] - \mu_A^2\mu_X^2 \\ &= \big(c_A(\tau) + \mu_A^2\big)\big(c_X(\tau) + \mu_X^2\big) - \mu_A^2\mu_X^2 \\ &= \big(c_A(\tau) + \mu_A^2\big)c_X(\tau) + c_A(\tau)\mu_X^2 \end{aligned}$$

where $c_{(\cdot)}(\tau)$ denotes the covariance of process $(\cdot)$ at lag τ. □

It is evident that the distribution of the employed sampling-process, i.e., specifically the sampling intensity μ_A and the inter-sample covariance $c_A(\tau)$, directly influence the observable covariance structure $c_W(\tau)$. Furthermore, rearranging Lemma 5.1 shows that the true covariance structure $c_X(\tau)$ of the traffic flow can be recovered from $c_W(\tau)$ as

$$c_X(\tau) = \frac{c_W(\tau) - c_A(\tau)\mu_X^2}{c_A(\tau) + \mu_A^2}. \tag{16}$$

In order to extract $c_X(\tau)$ from the observed covariance we require knowledge of the intensity and the covariance of the sampling process $A(t)$,

as well as the mean rate of the process $X(t)$. While the mean traffic rate μ_X must be estimated from measurements, the first two properties are typically known by the operator, or may be estimated empirically when closed-form expressions are not available.

In this thesis we focus on the special case of a Bernoulli sampling process, i.e., a point process with geometrically distributed sample times. As the increments of the process are uncorrelated, its covariance is zero for $\tau > 0$. Substituting $c_A(\tau) = 0$ into Eq. (16) yields

$$c_X(\tau) = \frac{c_W(\tau)}{\mu_A^2}. \tag{17}$$

This shows that memoryless sampling processes preserve the covariance structure of $X(t)$. An additional desirable property of such processes is that knowledge of the traffic process mean μ_x is not required.

In general, note that in cases where the autocovariance of the sampling process $c_A(\tau)$ decays faster than $c_X(\tau)$ we can use Lemma 5.1 to show that the asymptotic tail decay of $c_W(\tau)$ is the same as $c_X(\tau)$ except for a scaling factor. Dividing both sides of Lemma 5.1 by $c_X(\tau)$ and taking the limit for $t \to \infty$ yields

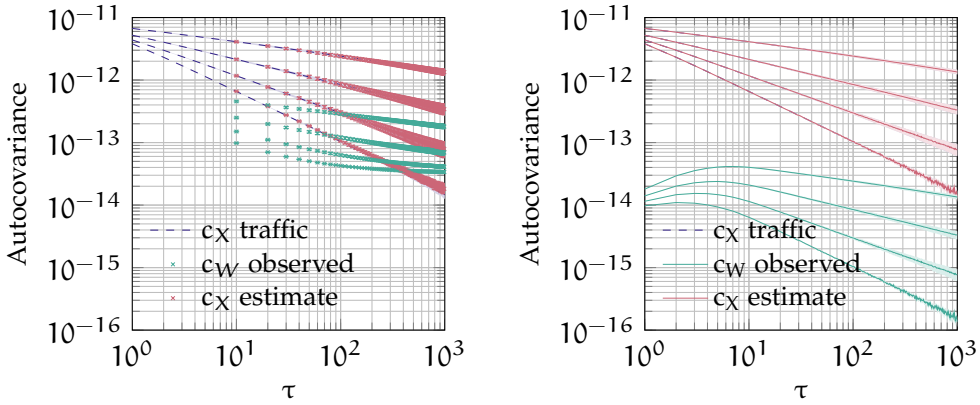
$$\lim_{t\to\infty} \frac{c_W(\tau)}{c_X(\tau)} = \lim_{t\to\infty} c_A(\tau) + \mu_A^2 + \frac{c_A(\tau)}{c_X(\tau)}\mu_X^2 = \mu_A^2. \tag{18}$$

Examples of such sampling processes are SRD, or Markovian processes with exponentially decaying covariances. This observation matches the asymptotic result from [115].

In addition to Bernoulli sampling, in [124] we investigated three additional inter-sample distributions: periodic, Gamma, and uniform. For each case we derived analytical expressions for $c_A(\tau)$ which we substitute into Eq. (16) in order to recover $c_X(\tau)$ from the observed covariance $c_W(\tau)$. The reconstruction results are depicted in Fig. 25 as a reference, for the corresponding derivations refer to [124]. In the sequel we consider Bernoulli sampling processes.

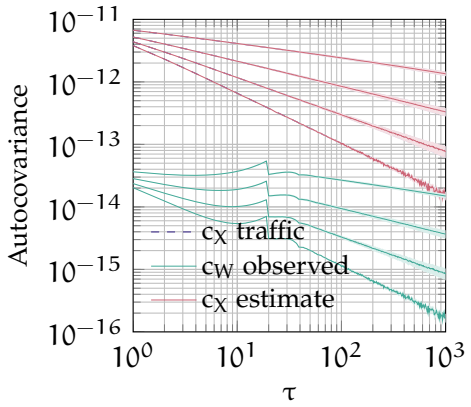
SIMULATIONS Figures 24 and 25 demonstrate the effects of the sampling distribution on the observed autocovariance, as well as the feasibility of the reconstruction operation. We performed simulations using synthetic LRD traffic traces with autocovariances $c_X(\tau) \sim \sigma_X^2 \tau^{2H-2}$ and $H \in \{0.6, 0.7, 0.8, 0.9\}$[3]. We sample the traffic traces using different probing strategies setting the probing intensity to $\mu_A = 0.1$ for all simulation scenarios. In Fig. 24 we see that for geometrically distributed inter-sample intervals the structure of the observed covariance $W(t)$ corresponds to the true autocovariance $c_X(\tau)$ of the process. The reconstructed covariance, denoted "c_X *estimate*", is derived by dividing $c_W(\tau)$ by μ_A^2. On a logarithmic scale this corresponds to a vertical shift of the covariance structure. Note that the estimate matches $c_X(\tau)$ exactly.

3 The length of each trace was 2.5×10^8 time slots. For each parameter H the simulation was repeated 25 times.



(a) Periodic sampling

(b) Gamma sampling



(c) Uniform sampling

Figure 25: Observed and reconstructed autocovariances for simulated LRD traffic processes using different sampling strategies. Note that the reconstructed "$c_Y(\tau)$ estimate" covers "$c_Y(\tau)$ traffic".

Figure 25 depicts the effects of periodic, Gamma, and uniform sampling processes, respectively. For these cases the observed covariance is clearly distorted. Nevertheless, the covariances reconstructed using Eq. (16) cover the original traffic autocovariances *"c_X traffic"* exactly. However in each of these scenarios an estimate of the mean traffic rate μ_X was required. Note, that for $\tau \to \infty$ the covariance structure for both gamma and uniform sampling exhibit the same decay as the $c_X(\tau)$. This is not the case for periodic sampling, as $c_a(\tau)$ does not tend to zero for large lags. Furthermore, using periodic sampling, $c_X(\tau)$ can only be recovered at lags which correspond to a multiple of the sampling period.

Throughout the remainder of this work we focus on geometric sampling due to its favorable reconstruction properties. Furthermore, the use of memoryless sampling for network probing is advocated by the IETF in [112].

5.4.2 *Impact of Finite Sample Sizes*

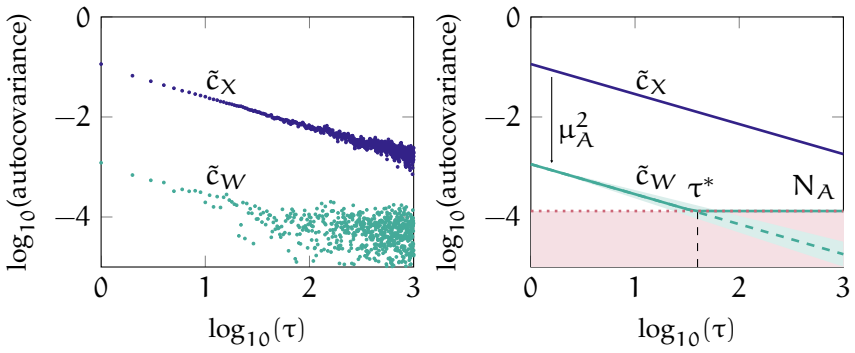
In any practical scenario the number of samples available for estimation of the autocovariance is finite. Hence, we now evaluate the impact of the sample size on the accuracy of the considered estimators. To this end, we derive analytical margins which enable us to assess the quality of the collected estimates. Moreover, understanding the effects of finite sample sizes enables us to relax the stationarity assumption by assuming only a piece-wise stationarity for the duration of a measurement.

In particular we investigate the impact of sampling duration and sampling intensity on the observations. In the previous section we assumed that for Bernoulli sampling the autocovariance $c_A(\tau) = 0$ for $\tau > 0$. However, this relationship holds only in the limit for $T \to \infty$ where T is the number of elements in the finite discrete sampling process $A(t)$.

In the following, we consider the sample autocovariance of a process $(\cdot)$, denoted $\tilde{c}_{(\cdot)}$, as an estimator of the population autocovariance $c_{(\cdot)}$. Analogously, we consider the sample mean $\tilde{\mu}_{(\cdot)}$ as an estimator of the population mean $\mu_{(\cdot)}$. We examine the effects of the involved sample covariance individually. In the next section we evaluate the impact of the sampling duration T on the visibility of the observed autocovariance $\tilde{c}_W(\tau)$. Subsequently, in Section 5.4.2.2 we assess the impact of the sample duration on the bias of the autocovariance estimators.

5.4.2.1 *Observation limit*

For decreasing observation periods T, the sample autocovariance of the observed traffic flow becomes increasingly noisy, making the extraction of the true flow autocovariance difficult. In this section, we aim to identify the range of lags in which the observed autocovariance differs significantly from the "noise" which is due to the limited number of samples available for the calculation. The sample autocovariance $\tilde{c}_X(\tau)$ of a LRD traffic process $X(t)$ calculated from a finite sample size T is depicted in Fig. 26a. In addition the figure contains the corresponding autocovariance of Bernoulli sampled observations $\tilde{c}_W(\tau)$. Clearly, in addition to a vertical shift of $\tilde{c}_X(\tau)$ the sampling operation produces an "observation noise" which becomes continuously more pronounced for increasing lags τ. We seek to identify the range $\tau \in [0, \tau^*]$ over which this noise does not significantly obstruct the observation of the underlying autocovariance structure. To this end, we use a standard technique [10] and compare the covariance of the observed process to the 0.95 confidence limit for the autocovariance of a Bernoulli sampled i.i.d. processes with the same mean rate and variance. We obtain a value for τ^* at which $c_W(\tau)$ from Eq. (17) and the 0.95 confidence interval intersect. The approach is illustrated schematically in Fig. 26b, where the shaded area indicates the 0.95 confidence interval. The upper confidence limit N_A may be interpreted as a "noise floor" which limits the visibility of the measured autocovariance structure for values smaller than N_A. The com-



(a) Sampling (b) Schematic description

Figure 26: Observation noise due to finite sampling.

parison with an i.i.d. process is motivated by the fact that for most processes the autocovariance becomes negligible for sufficiently large lags. Hence the observation noise can be regarded as an inherent property for any traffic process observed using a limited number of samples.

To obtain an analytic expression for the noise floor we consider the autocovariance function $c_V(\tau)$ of an observed process $V(t) = Y(t)A(t)$ where $Y(t)$ is an i.i.d. process sampled using a Bernoulli process $A(t)$ with sampling intensity μ_A. An estimator of the autocovariance is given by

$$\begin{aligned}\tilde{c}_V(\tau) &= \frac{1}{T-\tau}\sum_{t=1}^{T-\tau}(v(t)-\mu_V)(v(t+\tau)-\mu_V)\\ &= \mu_V^2 + \frac{1}{T-\tau}\sum_{t=1}^{T-\tau}v(t)v(t+\tau) - \frac{\mu_V}{T-\tau}\sum_{t=1}^{T-\tau}(v(t)+v(t+\tau))\end{aligned} \tag{19}$$

We assume that $T-\tau$ is large and use the central limit theorem (CLT) to approximate the summations by random Gaussian variables. We make use of the fact the elements of $V(t)$ are independent to obtain $E[v(t)v(t+\tau)] = \mu_V^2$ and $\mathrm{Var}[v(t)v(t+\tau)] = 2\mu_V^2\sigma_V^2 + \sigma_V^4$, as well as $E[v(t)-v(t+\tau)] = 2\mu_V$ and $\mathrm{Var}[v(t)-v(t+\tau)] = 2\sigma_V^2$. Applying the CLT to Eq. (19) yields the following approximation:

$$\begin{aligned}\tilde{c}_V(\tau) &\approx \mu_V^2 + \mathcal{N}\left(\mu_V^2, \frac{2\mu_V^2\sigma_V^2+\sigma_V^4}{T-\tau}\right) - \mathcal{N}\left(2\mu_V^2, \frac{2\sigma_V^2\mu_V^2}{T-\tau}\right)\\ &= \mathcal{N}\left(0, \frac{4\mu_V^2\sigma_V^2+\sigma_V^4}{T-\tau}\right).\end{aligned}$$

The 0.95 confidence levels for the observation noise correspond to two times the standard deviation of the expression above, i.e.,

$$N_{A^\pm} = \pm 2\sqrt{\frac{4\mu_V^2\sigma_V^2+\sigma_V^4}{T-\tau}}.$$

Let $\mu_Y = \mu_X$ and $\sigma_Y^2 = \sigma_X^2$ denote the mean, respectively, the variance of the i.i.d. process $Y(t)$. Further, the variance of a Bernoulli sampling process is $\sigma_A^2 = \mu_A - \mu_A^2$. Then using Lemma 5.1 to derive the variance of the observation process as $\sigma_V^2 = \sigma_A^2\mu_Y^2 + \mu_A^2\sigma_Y^2 + \sigma_A^2\sigma_Y^2 = \mu_A(\mu_Y^2 - \mu_Y^2\mu_A + \sigma_Y^2)$ and with the relationship $\mu_V = \mu_Y\mu_A$ we obtain the following upper confidence limit N_A assuming $T \gg \tau$:

$$N_A = \frac{2}{\sqrt{T}}\mu_A(\mu_Y^2 - \mu_Y^2\mu_A + \sigma_Y^2)\sqrt{\frac{4\mu_Y^2\mu_A}{\mu_Y^2 - \mu_Y^2\mu_A + \sigma_Y^2} + 1}. \tag{20}$$

Evidently an increase in the sampling duration $T \to \infty$ pushes the noise floor towards zero.

Finally, to obtain the lag τ^* at which the confidence limit intersects with the shifted autocovariance function of the considered LRD process we set $N_A = \mu_A^2 c_X(\tau)$ with $c_X(\tau) = K\sigma_X^2\tau^{2H-2}$ according to Eq. (4). For ease of exposition we ignore the effects of the sampling duration on the observed autocovariance $Y(t)$ and evaluate these separately in Section 5.4.2.2. Solving for τ yields:

$$\tau^* = \left(\frac{\sqrt{T}\mu_A K\sigma_Y^2}{2(\mu_Y^2 - \mu_Y^2\mu_A + \sigma_Y^2)\sqrt{\frac{4\mu_Y^2\mu_A}{\mu_Y^2 - \mu_Y^2\mu_A + \sigma_Y^2} + 1}} \right)^{\frac{1}{2H-2}}.$$

The formulation above shows that for given σ_Y^2 and μ_Y the autocovariance structure of LRD processes with a larger Hurst parameter H is more easily observable as τ^* is shifted to the right. The results of this section indicate that the observation noise must be taken into account to ensure the visibility of the autocovariance structure.

5.4.2.2 *Bias of the autocovariance estimators*

In the previous section we assumed that the estimated autocovariance structure of an observed traffic process is not effected by the finite measurement duration. We now evaluate the bias of the sample autocovariance which emerges when the number of samples available for the calculation is limited. Initially we consider a direct estimation (i.e., $\mu_A = 1$) of the process autocovariance $\tilde{c}_X(\tau)$ for a finite sampling duration with T samples. The estimator is unbiased iff $E[\tilde{c}_X(\tau)] = c_X(\tau)$. Given a sample path $x(t)$ with sample mean $\tilde{\mu}_{X_0} = \frac{1}{(T-\tau)}\sum_{t=1}^{T-\tau} x(t)$ and $\tilde{\mu}_{X_\tau} = \frac{1}{(T-\tau)}\sum_{t=1}^{T-\tau} x(t+\tau)$ we use the following autocovariance estimator:

$$\tilde{c}_X(\tau) = \frac{1}{T-\tau}\sum_{t=1}^{T-\tau}(x(t) - \tilde{\mu}_{X_0})(x(t+\tau) - \tilde{\mu}_{X_\tau}). \tag{21}$$

To estimate the bias we derive the expected value $\mathrm{E}[\tilde{c}_X(\tau)]$

$$\begin{aligned}\mathrm{E}[\tilde{c}_X(\tau)] =& \mathrm{E}\left[\frac{1}{T-\tau}\sum_{t=1}^{T-\tau} x(t)x(t+\tau) - \frac{1}{T-\tau}\sum_{t=1}^{T-\tau} x(t)\tilde{\mu}_{X_\tau}\right.\\ &\left. -\frac{1}{T-\tau}\sum_{t=1}^{T-\tau} x(t+\tau)\tilde{\mu}_{X_0} + \frac{1}{T-\tau}\sum_{t=1}^{T-\tau} \tilde{\mu}_{X_0}\tilde{\mu}_{X_\tau}\right] \end{aligned} \tag{22}$$

and evaluate its terms separately. Using the population parameters $c_X(\tau)$ and μ_X we write

$$\mathrm{E}\left[\frac{1}{T-\tau}\sum_{t=1}^{T-\tau} x(t)x(t+\tau)\right] = c_X(\tau) + \mu_X^2.$$

Further, using $\frac{1}{T-\tau}\sum_{t=1}^{T-\tau} x(t) = \tilde{\mu}_{X_0}$ we obtain

$$\mathrm{E}\left[\frac{1}{T-\tau}\sum_{t=1}^{T-\tau} x(t)\tilde{\mu}_{X_\tau}\right] = \mathrm{E}[\tilde{\mu}_{X_0}\tilde{\mu}_{X_\tau}].$$

Similarly, for the term $x(t+\tau)\tilde{\mu}_{X_0}$ we get

$$\mathrm{E}\left[\frac{1}{T-\tau}\sum_{t=1}^{T-\tau} x(t+\tau)\tilde{\mu}_{X_0}\right] = \mathrm{E}[\tilde{\mu}_{X_\tau}\tilde{\mu}_{X_0}].$$

Plugging these results into Eq. (22) yields

$$\begin{aligned}\mathrm{E}[\tilde{c}_X(\tau)] =& c_X(\tau) + \mu_X^2 - \mathrm{E}[\tilde{\mu}_{X_0}\tilde{\mu}_{X_\tau}]\\ =& c_X(\tau) - \mathrm{Cov}[\tilde{\mu}_{X_0}, \tilde{\mu}_{X_\tau}],\end{aligned} \tag{23}$$

where we used $\mathrm{E}[\tilde{\mu}_{X_0}\tilde{\mu}_{X_\tau}] = \mathrm{Cov}[\tilde{\mu}_{X_0}, \tilde{\mu}_{X_\tau}] + \mathrm{E}[\tilde{\mu}_{X_0}]\,\mathrm{E}[\tilde{\mu}_{X_\tau}]$ and the unbiased estimators $\mathrm{E}[\tilde{\mu}_{X_0}] = \mu_X$ and $\mathrm{E}[\tilde{\mu}_{X_\tau}] = \mu_X$. Furthermore, we can express the covariance of the sample mean in terms of the process autocovariance $c_x(\tau)$ to obtain an expression for the bias of the estimator:

$$\mathrm{E}[\tilde{c}_X(\tau)] = c_X(\tau) - \frac{1}{(T-\tau)^2}\sum_{i=1}^{T-\tau}\sum_{j=1}^{T-\tau} c_X(i-j+\tau). \tag{24}$$

The derivation of this expression is provided in Section A.1 of the appendix. Alternately, we can obtain an approximation of Eq. (23) for $T \gg \tau$ by taking advantage of the fact that the samples used to calculate $\tilde{\mu}_{X_0}$ and $\tilde{\mu}_{X_\tau}$ overlap over a range $T - 2\tau$. Thus, if we assume that the number of available samples is much larger than the considered lag, we may write $\tilde{\mu}_{X_0} \approx \tilde{\mu}_{X_\tau}$ and therefore approximate $\mathrm{Cov}[\tilde{\mu}_{X_0}, \tilde{\mu}_{X_\tau}] \approx \mathrm{Var}[\tilde{\mu}_{X_0}]$ to obtain

$$\mathrm{E}[\tilde{c}_X(\tau)] \approx c_X(\tau) - \mathrm{Var}[\tilde{\mu}_{X_0}]. \tag{25}$$

The investigations above show that for a finite number of samples T the autocovariance estimator exhibits a bias which corresponds to the variance of the sample mean. For LRD processes the variance of the sample

mean estimated from $T-\tau$ samples decays as $\mathrm{Var}[\tilde{\mu}_{X_0}] = \frac{\sigma_X^2}{(T-\tau)^{2-2H}}$. In other words, the rate at which the bias for LRD processes diminishes is characterized by its Hurst parameter H, i.e., higher H necessitate a larger number of samples T to obtain a negligible bias. Nevertheless, from Eq. (25) it also follows that the autocovariance estimator is asymptotically unbiased for $T \to \infty$ and $T \gg \tau$. As a consequence, the lag range over which the sample autocovariance is evaluated must be carefully chosen to ensure a negligible bias.

In Section 5.4.3 we provide an approach which yields unbiased estimates of the Hurst parameter for finite measurement intervals. This approach can be applied to obtain an estimate of the variance of the sample mean.

BIAS OF THE OBSERVED SAMPLE AUTOCOVARIANCE Next, we consider the bias of the sample autocovariance $\tilde{c}_W(\tau)$ of the observed process $W(t)$ calculated from a limited number of samples T. From Eq. (25) and using $\tilde{\mu}_{W_0} = \frac{1}{(T-\tau)} \sum_{t=1}^{T-\tau} w(t)$ we get

$$\mathrm{E}[\tilde{c}_W(\tau)] \approx c_W(\tau) - \mathrm{Var}[\tilde{\mu}_{W_0}] \tag{26}$$

We rearrange the variance of the sample mean in the expression above to obtain the following expression

$$\begin{aligned}\mathrm{Var}[\tilde{\mu}_{W_0}] = \frac{1}{(T-\tau)^2} \mathrm{Var}\left[\sum_{t=1}^{T-\tau} w(t)\right] &= \frac{1}{(T-\tau)^2} \sum_{i=1}^{T-\tau} \sum_{j=1}^{T-\tau} c_W(|i-j|) \\ &= \frac{c_W(0)}{T-\tau} - \frac{2}{(T-\tau)^2} \sum_{t=1}^{T-\tau-1} (T-\tau-t) c_W(t),\end{aligned}$$

where $c_W(0)$ denotes the variance of the observation process $\sigma_W^2 = \sigma_A^2 \mu_X^2 + \mu_A^2 \sigma_X^2 + \sigma_A^2 \sigma_X^2$. It follows that for Bernoulli sampling processes the impact of the sampling intensity on the bias is given by the following lemma:

Lemma 5.2. *The relationship between the expected value of sample autocovariance* $\mathrm{E}[\tilde{c}_W(\tau)]$ *of the observation process* $W(t)$ *obtained through Bernoulli sampling with probing intensity* μ_A *and the expected value of the sample autocovariance* $\mathrm{E}[\tilde{c}_X(\tau)]$ *of the sampled stationary process* $X(t)$ *for* $\tau > 0$ *is*

$$\mathrm{E}[\tilde{c}_X(\tau)] \approx \frac{1}{\mu_A^2} \mathrm{E}[\tilde{c}_W(\tau)] + \frac{(\frac{1}{\mu_A} - 1)(\mu_X^2 + \sigma_X^2)}{(T-\tau)}.$$

Proof. Substituting $c_W(\tau) = \mu_A^2 c_X(\tau)$ from Eq. (17) into Eq. (26) for $\tau > 0$ and rearranging yields

$$\begin{aligned}
\mathrm{E}[\tilde{c}_W(\tau)] &\approx \mu_A^2 c_X(\tau) - \frac{\sigma_A^2 \mu_X^2 + \sigma_A^2 \sigma_X^2}{T-\tau} + \frac{\mu_A^2 \sigma_X^2}{T-\tau} \\
&\quad - \frac{2\mu_A^2}{(T-\tau)^2} \sum_{t=1}^{T-\tau-1} (T-\tau-t) c_X(t) \\
&= \mu_A^2 c_X(\tau) - \frac{\sigma_A^2 (\mu_X^2 + \sigma_X^2)}{T-\tau} + \mu_A^2 \operatorname{Var}[\tilde{\mu}_{X_0}] \\
&\approx \mu_A^2 \mathrm{E}[\tilde{c}_X(\tau)] - \frac{\mu_A(1-\mu_A)(\mu_X^2 + \sigma_X^2)}{T-\tau},
\end{aligned}$$

where Eq. (25) was used in the last step. Solving for $\mathrm{E}[\tilde{c}_X(\tau)]$ completes the proof. □

This result shows that the additional impact of Bernoulli sampling on the bias of the observed sample autocovariance is tractable. We emphasize, that the bias which is due solely to the sampling process disappears for $T \to \infty$ and $T \gg \tau$. Furthermore, the bias tends towards zero as the sampling intensity $\mu_A \to 1$. Finally, we note that the sampling bias is not dependent on the correlation structure of the observed process. Thus, it may be corrected using unbiased estimates of the process mean μ_X and variance σ_X^2.

We conclude this section with a brief overview of the presented findings. We outlined a methodology for extracting the autocovariance of traffic flows using randomly observed samples. In particular, it was shown that the covariance structure of a randomly sampled observation process is distorted and that this distortion is dependent on the distribution of the inter sample times. However, these distortions may be reversed. For Bernoulli sampled processes the covariance structure is recovered from the observed autocovariance by a simple scaling operation $c_W(\tau)/\mu_A^2$. We quantified the impact of using a limited number of samples for the autocovariance estimation, identifying two limiting factors. Firstly, the sampling process produces a noise floor which may mask the true covariance structure. Secondly, the bias of the covariance estimators depends on the variance of the sample mean. For LRD processes this variance depends on the Hurst parameter, necessitating longer measurement intervals for strongly correlated processes. These observation constraints disappear in the limit for large measurement durations. Finally, we showed that the contribution of the Bernoulli sampling process on the total sample autocovariance bias may be eliminated using unbiased estimates of the mean and variance of the observed process.

5.4.3 *Unbiased Hurst Parameter Estimation*

In this section we focus on the estimation of the Hurst parameter which characterizes LRD traffic processes. This parameter is required for the

parametric approximation of the queue length distribution using Eq. (14). Furthermore, the Hurst parameter may be used to calculate the variance of the sample mean in order to eliminate the bias of the autocovariance estimator in (25). In the following we make use of the so-called aggregated variance estimator [136], to quantify the degree of correlation of a traffic process. The method exploits the fact, that the variance of strongly correlated processes, aggregated over increasing intervals M, decays slower than the variance of weakly correlated processes.

Specifically, consider an LRD process $X(t)$ segmented into consecutive blocks of M samples. Further, let $X^{(M)}$ denote the aggregated process obtained by taking the mean of each block, i.e.,

$$X^{(M)}(k) = \frac{1}{M} \sum_{t=1+(k-1)M}^{kM} X(t) \quad \text{for} \quad k \in \mathbb{N}. \tag{27}$$

The variance of the aggregated process $X^{(M)}$ is given by

$$\mathrm{Var}\left[X^{(M)}\right] = \sigma^2 M^{2H-2}, \tag{28}$$

where σ denotes the variance of the non-aggregated process $X(t)$, and H is the Hurst parameter with $H \in (0.5, 1)$. A derivation of Eq. (28) is provided in Appendix Section A.2. An estimate of the Hurst parameter H may be obtained by evaluating the decay of the aggregated variance $\mathrm{Var}\left[X^{(M)}\right]$ for increasing block lengths M (called aggregation intervals hereafter). Specifically, H may be estimated by performing a regression in the logarithmic domain and estimating the slope $2H - 2$. Note, that for uncorrelated processes with $H = 0.5$ we get the well known variance decay $\sim M^{-1}$. Furthermore, note that for processes with Gaussian increments the aggregated variance may be used to obtain point-wise effective envelopes [87].

In this section we show that the aggregated variance estimator offers several advantages for estimating the Hurst parameter compared to the approach based on the autocovariance decay discussed in the previous section. We evaluate the influence of stochastic sampling on the estimator and derive a modification which enables us to obtain an unbiased estimate of the process Hurst parameter, which is independent of the observation duration (i.e., the number of available samples).

In Section 5.4.2.1 we showed that for finite sample sizes, the range over which the covariance estimator produces useful estimates of H depends on the Hurst parameter itself. Specifically, for cases where H is small a significant number of samples is necessary in order to sufficiently reduce the noise floor which perturbs the estimate. In the following we will demonstrate that the aggregated variance estimator performs much better in such scenarios.

In addition to the aggregated variance estimator described above, where the process $X(t)$ is split into non-overlapping intervals with length M, in the sequel we employ a second estimator which is derived directly

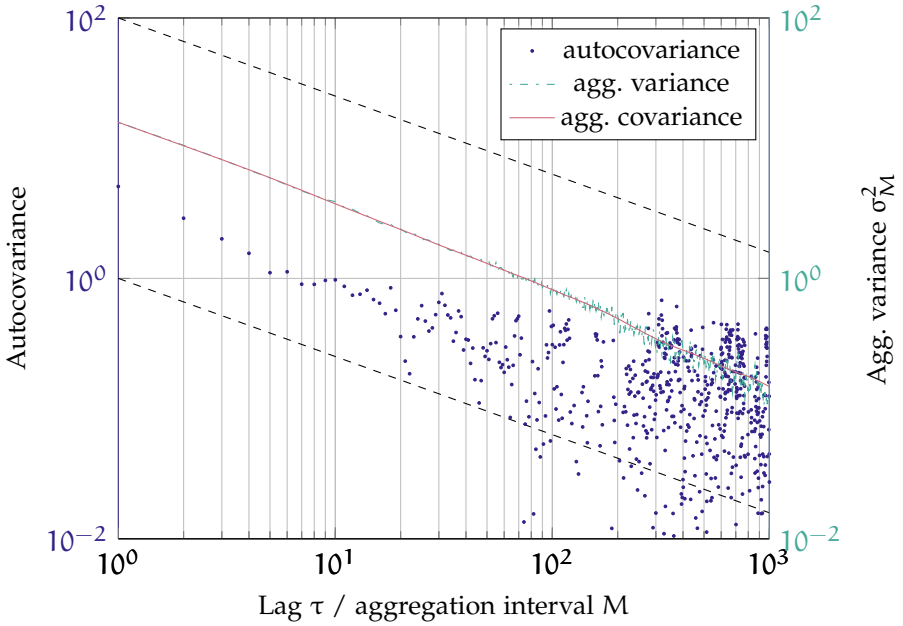


Figure 27: Autocovariance and aggregated variance estimates of an exemplary finite length process with $N = 10000$ points and $H = 0.7$. The autocovariance estimate is obstructed by the noise floor. The aggregated variance estimate exhibits less noise while the aggregated covariance estimator produces the smoothest estimate. Dashed lines correspond to a decay of $2H - 2$.

from the autocovariance $c_X(\tau)$ of the non-aggregated process $X(t)$. We will show that for finite length processes the estimates generated by this estimator are less noisy compared to the traditional aggregate variance estimate. We make use of the following relationship between the variance of the aggregated process $\mathrm{Var}\left[X^{(M)}\right]$ and the covariance of $X(t)$:

$$\mathrm{Var}\left[X^{(M)}\right] = \mathrm{Var}\left[\frac{1}{M}\sum_{t=1}^{M} X(t)\right] = \frac{1}{M^2}\sum_{i=1}^{M}\sum_{j=1}^{M}\mathrm{Cov}[X(i), X(j)]. \quad (29)$$

For stationary processes we can use the relationship $\mathrm{Cov}[X(i), X(j)] = c_x(|i-j|)$ to rearrange Eq. (29) yielding

$$\mathrm{Var}\left[X^{(M)}\right] = \frac{c_x(0)}{M} + \frac{2}{M^2}\sum_{\tau=1}^{M-1}(M-\tau)c_x(\tau). \quad (30)$$

It follows from Eq. (30) that the variance of the aggregated process can be obtained by estimating the autocovariance function and suitably summing its elements. In the sequel, we refer to the two estimators as *aggregated variance* and *aggregated covariance* estimators, respectively. Note, that per definition the two variance estimators are equivalent.

We now illustrate some properties of the Hurst parameter estimators. Consider a finite length LRD process $X(t)$ with $T = 10000$ samples and Hurst parameter $H = 0.7$. Figure 27 depicts the autocovariance (left Y-axis) and the two aggregated variance estimates for this process (right

Y-axis). As expected, all estimates decay approximately with a slope of $2H-2$ on the log-log scale (the expected slope is indicated by the auxiliary dashed lines). It is evident that the autocovariance estimate fluctuates significantly as the lag τ increases, making an accurate estimation of the covariance decay difficult. This is a result of the noise floor discussed in the previous section. On the other hand, both variance based estimates do not noticeably suffer from such a distortion. Moreover, the aggregated covariance estimate is smoother than the aggregated variance estimate. This effect is explained by the smaller number of available samples for calculating the aggregated variance at higher aggregation levels. As the process is divided into non-overlapping segments, for an aggregation level M we obtain $\lfloor T/M \rfloor$ samples. On the other hand, for the same aggregation level M the aggregated covariance method relies on the sample autocovariances $\tilde{c}_X(\tau)$ at lags $\tau \in [1, M-1]$ which are estimated using *at least* $T-M-1$ samples. In Section 5.4.5 we show that for a finite number of samples both estimators exhibit an equal bias.

5.4.4 *Effects of Sampling on Variance-based Estimators*

We now analyze the effects of sampling on the variance based estimators. In Section 5.4.1 we showed that stochastic sampling distorts the autocovariance estimate and that the distortion can be reversed. We now consider the effects of the sampling process on the aggregated variance estimate and demonstrate that a corresponding reconstruction is also possible in this case.

The impact of the sampling process $A(t)$ on the aggregated variance of the observed process $W(t)$ is given by the following lemma:

Lemma 5.3. *Let $A(t)$ and $X(t)$ denote stationary and independent stochastic processes, and further let $W(t) = A(t)X(t)$. The variances of the corresponding processes $A^{(M)}$, $W^{(M)}$ and $X^{(M)}$ aggregated over an interval of length M are related through*

$$\operatorname{Var}\left[W^{(M)}\right] = \mu_X^2 \operatorname{Var}\left[A^{(M)}\right] + \mu_A^2 \operatorname{Var}\left[X^{(M)}\right] + \frac{1}{M}\sigma_X^2\sigma_A^2 + \frac{2}{M^2}\sum_{\tau=1}^{M-1}(M-\tau)c_X(\tau)c_A(\tau)$$

The proof of Lemma 5.3 is given in Section A.3 of the Appendix. The result shows that the aggregated variance $\operatorname{Var}\left[W^{(M)}\right]$ of the observed process $W(t)$ is comprised of the individual, scaled aggregated variances of the traffic process and the sampling process, $X^{(M)}$ and $A^{(M)}$, as well as a scaled summation of the autocovariances of the two processes, $c_X(\tau)$ and $c_A(\tau)$.

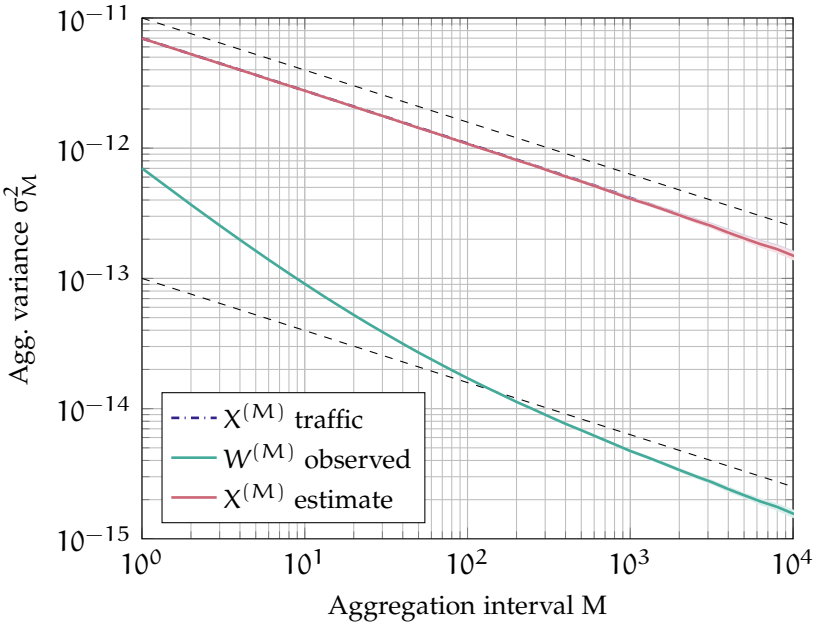


Figure 28: Aggregated variance estimate under geometric sampling, $H = 0.8$ and $\mu_A = 0.1$. The aggregated variance of the process $X(t)$ denoted "$X^{(M)}$ traffic" is exactly covered by the corresponding estimate "$X^{(M)}$ estimate". Shaded regions indicate 95% confidence intervals.[4]Dashed auxiliary lines have a slope corresponding to $H = 0.8$.

We now consider the case of geometric sampling where $c_A(\tau) = 0$ for $\tau > 0$, $\sigma_A^2 = \mu_A(1-\mu_A)$ and $\mathrm{Var}\left[A^{(M)}\right] = \sigma_A^2 M^{-1}$. Substituting these values into Lemma 5.3 and rearranging, we recover $\mathrm{Var}\left[X^{(M)}\right]$ as

$$\mathrm{Var}\left[X^{(M)}\right] = \frac{1}{\mu_A^2}\mathrm{Var}\left[W^{(M)}\right] - \frac{\sigma_A^2}{\mu_A^2 M}\left(\mu_X^2 + \sigma_X^2\right). \tag{31}$$

To estimate $\mathrm{Var}\left[X^{(M)}\right]$ from observations using (31) we additionally require knowledge of the mean μ_X and variance σ_X^2 of the traffic process which are not known a priori. As $\mathrm{E}[W] = \mathrm{E}[XA] = \mathrm{E}[X]\,\mathrm{E}[A]$ we estimate $\mu_X = \mu_W/\mu_A$. Further substituting $\tau = 0$ into Lemma 5.1 yields the variance of $X(t)$ as

$$\sigma_X^2 = \frac{\sigma_W^2 - \sigma_A^2\mu_X}{\sigma_A^2 + \mu_A^2} = \frac{\sigma_W^2 - (1-\mu_A)\mu_W}{\mu_A}. \tag{32}$$

Figure 28 depicts the aggregated variance of the traffic process $X^{(M)}$ (denoted "$X^{(M)}$ traffic") with Hurst parameter $H = 0.8$ as a function of the aggregation interval M. Furthermore, the figure shows the distorted aggregated variance $W^{(M)}$ of the geometrically sampled process $W(t)$ (denoted "$W^{(M)}$ observed"). Finally, the aggregated variance recovered using Eq. (31) is denoted "$X^{(M)}$ estimate". Observe, that the curves of the aggregated variance of the traffic and the corresponding estimate

4 We note, that the displayed CI regions are very narrow.

recovered from the sampled process match exactly. The Hurst parameter H corresponds to the slope of the decay of the aggregated variance $X^{(M)}$ on the log-log scale, i.e., $\sim 2H - 2$ (indicated by the auxiliary lines in Fig. 28).

For arbitrary sampling strategies Lemma 5.3 enables us to assess the distortion of the aggregated variance estimate due to the sampling process. For sampling processes where the covariance structure $c_A(\tau) \neq 0$ a reconstruction of $X^{(M)}$ is not straightforward as Lemma 5.3 contains a summation involving the product of $c_A(\tau)$ and $c_X(\tau)$. However, note that for any sampling processes, the relationship in Lemma 5.3 tends to

$$\operatorname{Var}\left[W^{(M)}\right] \approx \mu_X^2 \operatorname{Var}\left[A^{(M)}\right] + \mu_A^2 \operatorname{Var}\left[X^{(M)}\right]$$

for increasing aggregation intervals $M \to \infty$. As a result, the decay of the observed process $\operatorname{Var}\left[X^{(M)}\right]$ dominates in the observed aggregated variance $\operatorname{Var}\left[W^{(M)}\right]$ for large aggregation intervals for any sampling process which is not LRD.

5.4.5 *Bias of the Aggregated Variance Estimator*

In practical scenarios the interval over which a given process may be observed is finite. Therefore, we now evaluate the behavior of the aggregated variance $\sigma^2_{X^{(M)}} = \operatorname{Var}\left[X^{(M)}\right]$ for measurements with a limited duration. In the following, we quantify the bias of the estimator for a fixed number of samples T.

Consider the sample variance $\tilde{\sigma}^2_{X^{(M)}}$ of the aggregated process $X^{(M)}(k)$ derived by aggregating the process $X(k)$ over intervals of length M as defined in Eq. (27). Further, let $x(k)$ denote a sample path of the process $X(k)$ with T samples, and let $x^{(M)}(k)$ denote the corresponding aggregated process with $N = T/M$ samples. The sample variance of the aggregated process is given as

$$\tilde{\sigma}^2_{X^{(M)}} = \frac{1}{N} \sum_{k=1}^{N} \left(x^{(M)}(k) - \tilde{\mu}_{X^{(M)}} \right)^2 \tag{33}$$

where

$$\tilde{\mu}_{X^{(M)}} = \frac{1}{N} \sum_{k=1}^{N} x^{(M)}(k) \tag{34}$$

is the sample mean.

To evaluate the bias we consider the expected value of Eq. (33)

$$\begin{aligned} \mathrm{E}\left[\tilde{\sigma}^2_{X^{(M)}}\right] &= \mathrm{E}\left[\frac{1}{N} \sum_{k=1}^{N} \left(x^{(M)}(k)^2 - 2x^{(M)}(k)\tilde{\mu}_{X^{(M)}} + \tilde{\mu}^2_{X^{(M)}} \right) \right] \\ &= \mathrm{E}\left[x^{(M)}(k)^2 \right] - \mathrm{E}\left[2x^{(M)}(k)\tilde{\mu}_{X^{(M)}} \right] + \mathrm{E}\left[\tilde{\mu}^2_{X^{(M)}} \right] \end{aligned} \tag{35}$$

We inspect the three terms individually. The first term yields

$$\mathrm{E}\left[x^{(M)}(k)^2\right] = \sigma^2_{X^{(M)}} + \mu^2_{X^{(M)}}$$

where $\sigma^2_{X^{(M)}}$ and $\mu^2_{X^{(M)}}$ are the population parameters. For the second term we substitute Eq. (34) to calculate

$$\mathrm{E}\left[2x^{(M)}(k)\tilde{\mu}_{X^{(M)}}\right] = 2\,\mathrm{E}\left[\tilde{\mu}^2_{X^{(M)}}\right].$$

Finally, noting that for any M the expected value of the sample mean is given by

$$\begin{aligned}\mathrm{E}[\tilde{\mu}_{X^{(M)}}] &= \mathrm{E}\left[\frac{1}{N}\sum_{k=1}^{N} x^{(M)}(k)\right] = \mathrm{E}\left[x^{(M)}(k)\right] \\ &= \mathrm{E}\left[\frac{1}{M}\sum_{t=k}^{k+M-1} x(t)\right] = \mu_X,\end{aligned}$$

we simplify the third term as

$$\mathrm{E}\left[\tilde{\mu}^2_{X^{(M)}}\right] = \mathrm{Var}[\tilde{\mu}_{X^{(M)}}] + \mathrm{E}[\tilde{\mu}_{X^{(M)}}]^2 = \mathrm{Var}[\tilde{\mu}_{X^{(M)}}] + \mu^2_X.$$

Using these results Eq. (35) reduces to

$$\mathrm{E}\left[\tilde{\sigma}^2_{X^{(M)}}\right] = \sigma^2_{X^{(M)}} - \mathrm{Var}[\tilde{\mu}_{X^{(M)}}]. \tag{36}$$

Finally, substituting the sample path of non-overlapping intervals

$$x^{(M)}(k) = \frac{1}{M}\sum_{t=1+(k-1)M}^{kM} x(t)$$

into (34) reveals the following equality

$$\tilde{\mu}_{X^{(M)}} = \frac{1}{T/M}\sum_{k=1}^{T/M}\frac{1}{M}\sum_{t=1+(k-1)M}^{kM} x(t) = \frac{1}{T}\sum_{t=1}^{T} x(t) = \tilde{\mu}_X.$$

Therefore we write Eq. (36) as

$$\mathrm{E}\left[\tilde{\sigma}^2_{X^{(M)}}\right] = \sigma^2_{X^{(M)}} - \mathrm{Var}[\tilde{\mu}_X]. \tag{37}$$

As the variance of the sample mean of any stationary process tends towards zero for $T \to \infty$ the aggregated variance estimate $\tilde{\sigma}^2_{X^{(M)}}$ is asymptotically unbiased.

We obtain a similar result for the aggregated covariance estimator. To calculate its bias we take the expected value of Eq. (29)

$$\mathrm{E}\left[\tilde{\sigma}^2_{X^{(M)}}\right] = \frac{1}{M^2}\sum_{i=1}^{M}\sum_{j=1}^{M}\mathrm{E}[\tilde{c}_x(i-j)],$$

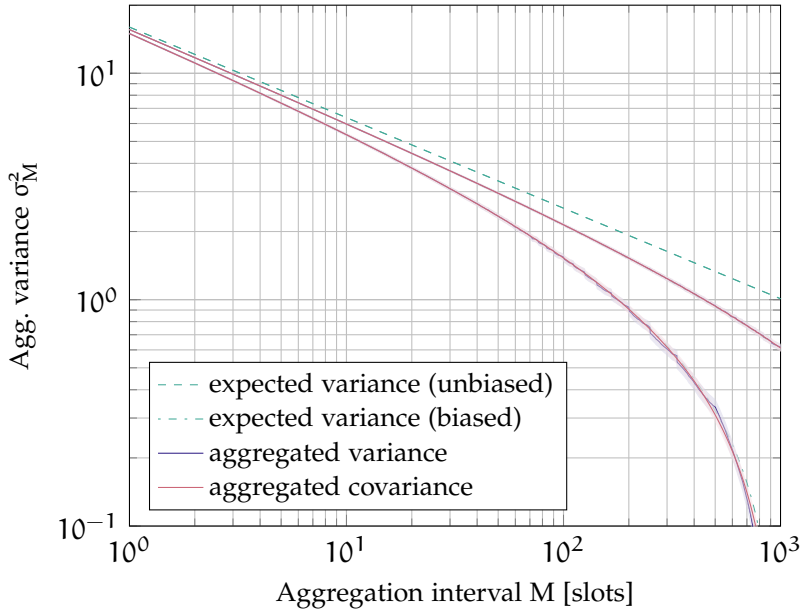


Figure 29: Bias of the aggregated variance estimate of an LRD process with H=0.8, calculated using 1000 and 10 000 samples, respectively. (shaded areas indicate 0.99 confidence intervals)

and substitute the expected value of the sample autocovariance $E[\tilde{c}_X(\tau)] \approx c_X(\tau) - \mathrm{Var}[\tilde{\mu}_X]$ from Eq. (25). Rearranging and substituting Eq. (29) yields

$$\begin{aligned} E\left[\tilde{\sigma}^2_{X^{(M)}}\right] &\approx \frac{1}{M^2}\sum_{i=1}^{M}\sum_{j=1}^{M} c_X(i-j) - \frac{1}{M^2}\sum_{i=1}^{M}\sum_{j=1}^{M}\mathrm{Var}[\tilde{\mu}_X] \\ &= \sigma^2_{X^{(M)}} - \mathrm{Var}[\tilde{\mu}_X]. \end{aligned} \tag{38}$$

We highlight that for both estimators the bias is independent of the considered aggregation interval M. We exploit this fact in the following section.

To illustrate the impact of a finite sampling duration T on the bias of the estimators, we generate synthetic LRD traffic sample paths with $T_1 = 1000$ and $T_2 = 10000$ samples, respectively. We evaluate the aggregated variance and aggregated covariance estimates, repeating each experiments 1000 times. The simulation results are depicted in Fig. 29. Note, that the expected (biased) variance decay calculated using Eq. (37) and represented by dash-dotted curves in Fig. 29 closely matches the simulated results. In the next section we discuss these simulation results in more detail taking the properties of LRD traffic into account.

5.4.5.1 *Aggregated Variance Bias for LRD Processes*

Next, we consider the estimator bias due to limited sample sizes T, for the special case of LRD traffic characterized by a Hurst parameter $H \in (0.5, 1)$. In the following, we omit the process index from the variances for notational brevity. Thus, unless stated otherwise, we use σ^2_M

to denote the variance of a process aggregated over an interval of length M, and write $\sigma^2 = \sigma_1^2$ for the variance of the non-aggregated process.

For an LRD process with Hurst parameter H, the aggregated variance decay is given by $\sigma_M^2 = \sigma^2 M^{2H-2}$, where σ^2 is the variance of the process [136, 10]. A derivation of this relationship using the autocovariance Eq. (3) is provided in Section A.2 of the appendix. Furthermore, for a finite length process with T samples the variance of the sample mean is known to be $\mathrm{Var}[\tilde{\mu}] = \sigma^2 T^{2H-2}$ [10]. Substituting the relationships above into Eq. (36) yields

$$\mathrm{E}\left[\tilde{\sigma}_M^2\right] = \sigma^2 M^{2H-2} - \sigma^2 T^{2H-2}. \tag{39}$$

It is interesting to note, that the bias of the aggregated variance depends only on the number of collected samples (i.e., the sampling duration) an not on the aggregation interval M. Moreover, Eq. (39) shows that the variance of the sample mean decays slowly for large Hurst parameter values. As a result, for highly correlated processes, a larger number of samples must be collected in order to reduce the impact of the bias. Note, that by setting $M = 1$ and $H = 0.5$ in (39) we recover the well known unbiased variance estimator for uncorrelated processes, i.e., $\mathrm{E}\left[\tilde{\sigma}^2\right] = \mathrm{E}\left[\frac{1}{T}\sum_{k=1}^{T}(x(k) - \widetilde{\mu}_X)^2\right] = \sigma^2\frac{T-1}{T}$ and hence $\mathrm{E}\left[\frac{1}{T-1}\sum_{k=1}^{T}(x(k) - \widetilde{\mu}_X)^2\right] = \sigma^2$.

The impact of the sampling duration on the estimate of the aggregated variance is depicted in Fig. 29 for synthetic LRD process with $H = 0.8$. The aggregated variance decay was calculated over a range of aggregation intervals $M \in [1, 10^3]$ using $T_1 = 1000$ and $T_2 = 10000$ samples, respectively, using both estimators. Shaded areas represent 99% confidence intervals. The dashed line indicates the unbiased variance decay $\sigma_M^2 = \sigma^2 M^{2H-2}$. For T_1 it is evident, that the aggregated variance estimate (denoted "expected variance (biased)") deviates significantly from σ_M^2 as predicted by Eq. (39). Note that this is an extreme case where the number of samples corresponds to the largest aggregation interval $M = 10^3$. Increasing the number of samples to $T_2 = 10000$ reduces the bias significantly. Finally, observe that both the *aggregated variance* and *aggregated covariance* methods yield an almost indistinguishable bias, which also closely matches the bias predicted by Eq. (39) (the curves are covered by the estimates).

Next, we demonstrate how the influence of the finite sample size T may be eliminated yielding an unbiased estimate of the Hurst parameter.

Lemma 5.4. *Let $\tilde{\sigma}^2_{X^{(M_k)}}$ denote the sample variances of a stationary LRD process $X(t)$ with Hurst parameter H aggregated over the intervals $M_k = b^{\delta k}$ where k is a sequence of equidistant points with $k = \langle k : k \in \mathbb{N}_0 \rangle$, $b > 1$ and $\delta > 0$. Define the metric $G_k = \tilde{\sigma}^2_{M_k} - \tilde{\sigma}^2_{M_{k+1}}$. An unbiased estimate of the Hurst parameter may be inferred from the exponent $2H-2$ of the expected value of G_k given by*

$$\mathrm{E}[G_k] = \sigma^2(1 - b^{2H-2})M_k^{2H-2}.$$

Proof. Substituting Eq. (39) into $E[G_k] = E\left[\tilde{\sigma}^2_{Y^{(M_k)}}\right] - E\left[\tilde{\sigma}^2_{Y^{(M_{k+1})}}\right]$ eliminates the bias term and thus the impact of the finite sample size T. Using geometrically increasing aggregation levels $M_k = b^{\delta k}$ we calculate the expected value $E[G_k]$

$$\begin{aligned} E[G_k] =& \sigma^2(M_k^{2H-2} - T^{2H-2}) - \sigma^2(M_{k+1}^{2H-2} - T^{2H-2}) \\ =& \sigma^2(b^{\delta k(2H-2)} - b^{\delta(k+1)(2H-2)}) \\ =& \sigma^2(1 - b^{2H-2})b^{\delta k(2H-2)}. \end{aligned}$$

Taking the logarithm on both sides of the expression yields

$$\log_b(E[G_k]) = \log_b(\sigma^2(1 - b^{2H-2})) + (2H-2)\delta k.$$

Thus, on a logarithmic scale an unbiased estimate of the Hurst parameter is obtained from the slope $(2H-2)$ evaluated at δk. □

The differencing approach outlined above has been proposed in [139] as a means for eliminating the impact of shifting means and slowly decaying trends in LRD processes. However the bias due to limited sampling durations is not considered and the authors assume sufficiently large values for T and M.

The relationship given in Lemma 5.4 is depicted in Fig. 30. The figure contains the standard aggregated variance estimate for synthetic LRD traces with Hurst parameter $H = 0.8$ and short sampling duration $T = 10\,000$ samples. Note the deviation of the estimate from the ideal decay of $2H-2$, represented by auxiliary dashed lines. In addition, an unbiased estimate of the variance decay, derived using the metric G_k from Lemma 5.4, was generated from the same data using the intervals $M_k = 2^k$ with $k = \langle 0, 1, 2, 3, \ldots \rangle$ represented by circles and including 0.99 confidence intervals. It is evident that the slope corresponds to $2H-2$.

Assume for a moment that the variances of the considered LRD process may be aggregated over arbitrary, infinitesimally small intervals. In such a scenario, the bias due to a finite sampling size T may be eliminated by taking the derivative of the sample variance

$$\frac{d\tilde{\sigma}^2_M}{dM} = (2H-2)\sigma^2 M^{2H-3}\sigma^2_M \tag{40}$$

allowing for an unbiased estimation of the Hurst parameter from the exponent $2H-3$ of Eq. (40).

However, in practice the aggregation intervals M are discrete. Therefore, we evaluate Eq. (40) for the case of discrete values of M, with discretization interval δ. In this case we approximate the derivative by a finite difference and define the following estimator:

$$\Gamma_M = \frac{\tilde{\sigma}^2_M - \tilde{\sigma}^2_{M+\delta}}{\delta}. \tag{41}$$

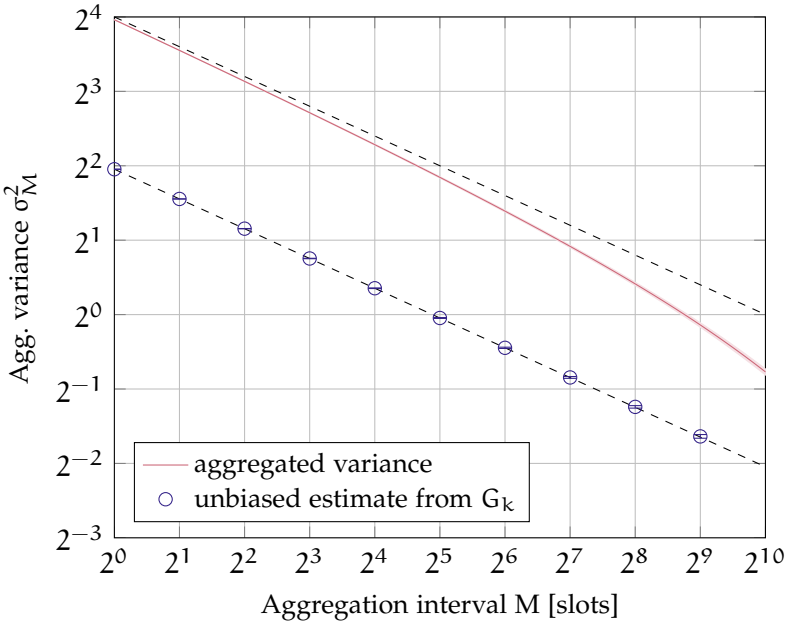


Figure 30: Unbiased Hurst parameter estimation. For short sampling durations ($T = 10^4$) the bias from Eq. (39) is apparent. An unbiased estimate of the exponent $2H-2$ is derived by evaluating the variance estimate G_k from Lemma 5.4 at the aggregation intervals $M_k = 2^k$ with $k = \langle 0, 1, 2, 3, \dots \rangle$. Dashed auxiliary lines have a slope corresponding to $2H-2$ for $H = 0.8$. (0.99 confidence intervals are displayed)

Per definition for $\delta \to 0$ this estimator tends towards the derivative in Eq. (40), i.e., $\lim_{\delta\to 0} \Gamma_M = (2H-2)\sigma^2 M^{2H-3}$. We now quantify the error caused by the finite difference approximation for $\delta \gg 0$. We denote the sample variance by the function $f(M+\delta) = \tilde{\sigma}^2_{M+\delta} = \sigma^2(M+\delta)^{2H-2}$ and use the notation $f'(\cdot)$ to represent the derivative of $f(\cdot)$. We perform a Taylor expansion of order $n = 1$ of the function $f(M+\delta)$ about the point $\delta = 0$ which yields the relationship

$$f(M+\delta) = f(M) + f'(M)\delta + \frac{f''(\vartheta)}{2}\delta^2,$$

where the variable ϑ is in the range $\vartheta \in [M, M+\delta]$. Next, we substitute f and its derivatives into the equation above and rearrange to get an expression for the finite difference

$$\begin{aligned}\frac{\sigma^2(M+\delta)^{2H-2} - \sigma^2(M)^{2H-2}}{\delta} &= \sigma^2(2H-2)M^{2H-3} \\ &\quad + \frac{\sigma^2(2H-3)(2H-2)(\vartheta)^{2H-4}}{2}\delta \\ &= \frac{d\tilde{\sigma}^2_M}{dM} + \eta(\delta).\end{aligned}$$

A comparison of this Taylor representation with the derivative from Eq. (40), reveals that the approximation error due to the finite difference is given by the term $\eta(\delta) = \frac{\sigma^2(2H-3)(2H-2)(\vartheta)^{2H-4}}{2}\delta$. Again, we find that

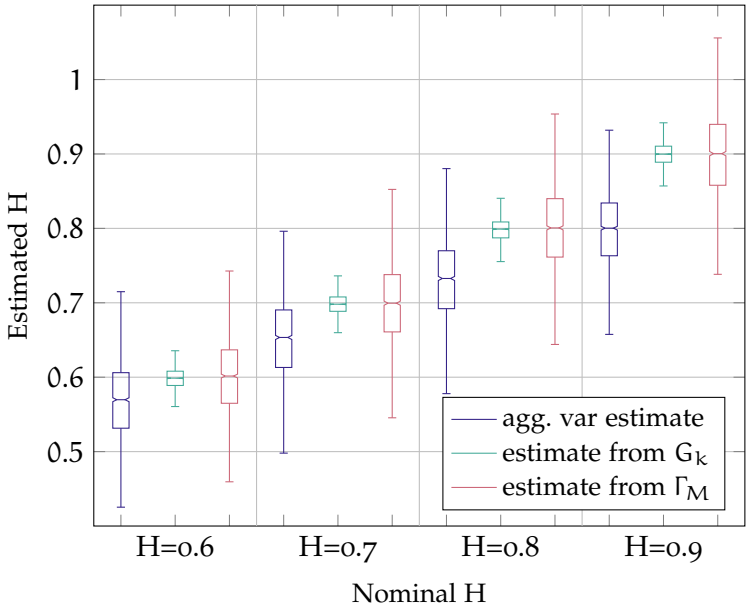


Figure 31: Biased and unbiased Hurst parameter estimates for synthetic traffic traces with 10 000 samples. Estimates were derived from the exponents $2H-2$ in Eq. (39) and $2H-3$ in Eq. (40), respectively. (Box-plot whiskers represent 0.95 confidence intervals)

this error tends towards zero for $\delta \to 0$. To obtain a bound on the approximation error, we observe that the error $\eta(\delta)$ is maximized for $\vartheta = M$. As a result, with $\vartheta = M$ we see that the approximation error is proportional to $\frac{\sigma^2 M^{2H-3}}{M}$, i.e., for increasing values of M the error term decays faster than the derivative in Eq. (40).

Figure 31 depicts boxplots for Hurst parameter estimates derived from the slope of the (biased) aggregated variance estimator Eq. (28), the metric G_k from Lemma 5.4 as well as Γ_M from Eq. (41) with $\delta = 1$. For each experiment we generate 2000 synthetic LRD traffic traces with $H \in \{0.6, 0.7, 0.8, 0.9\}$ and a short sampling duration $T = 10^4$. For the considered sampling duration T the means of the standard aggregated variance estimate lies below the configured Hurst parameter. On the other hand, the averages of the two other estimators closely correspond to the nominal Hurst parameter. For the estimate from Γ_M, it is evident, that the approximation error has a negligible impact on the Hurst parameter estimate. Further, we observe that the estimate from G_k exhibits the smallest variation around the mean.

We conclude by summarizing that the impact of the sampling duration in Eq. (28) diminishes for increasing sampling intervals $T \to \infty$, however, the estimators from Lemma 5.4 and Eq. (41) enable us to obtain accurate estimates of the Hurst parameter even for short T.

5.5 COUNTER SAMPLING

In the previous section we considered the estimation of the traffic covariance structure from packets randomly selected using a predefined sampling strategy. Applications of this *packet sampling* approach include passive packet monitoring, where individual packets are randomly forwarded to a centralized monitoring entity for evaluation, or active probing, where special probing packets are injected into a network link, as demonstrated in our previous work [124].

In contrast, in this section we consider the estimation of the covariance structure of a given traffic flow from random samples obtained from data counters on network devices. Traffic counters aggregate the total amount of data forwarded by a network interface or a specific network flow and are typically queried using mechanisms such as SNMP, IPFIX, or more recently, SDN protocols such as OpenFlow. In this section, we highlight that packet and counter sampling are inherently different and consequently require different techniques for estimating the correlation structure of observed flows (we provide analytical details of these differences in Section 5.5.1). In the sequel, we develop a strategy for estimating the autocovariance matrix of arbitrary flows observed at a specific forwarding device using counter queries with randomly distributed inter-query times. In addition, an analytical evaluation of our proposed algorithm is provided.

A main advantage of counter based sampling is that corresponding monitoring mechanisms are already integrated in modern network devices. Current SDN technologies, such as OpenFlow [99], are of particular interest, as they include byte and packet counters (e.g., flow counters and meters) which may be configured for any forwarding entry installed in the switch flowtable. As a consequence, the techniques described in this section enable the extraction of fine-grained, per-flow traffic correlations.

Specifically, in an SDN environment a controller may, at any time, obtain the total amount of traffic matched by a specific flow table entry (or a group of entries) on any connected forwarding device by submitting an appropriate statistic query message to the switch using the OpenFlow protocol.

As with packet sampling, we wish to extract the time dependency of the flow traffic processes using a sufficiently high sampling rate. In practice, the number of queries which may be generated within a given time interval is limited by the switch control logic, which is responsible for processing all control traffic, as well as the available bandwidth of the control channel. Growing network speeds and the desire to monitor large numbers of flows therefore necessitate the use of statistical sampling strategies in order to ensure scalability of the monitoring architecture.

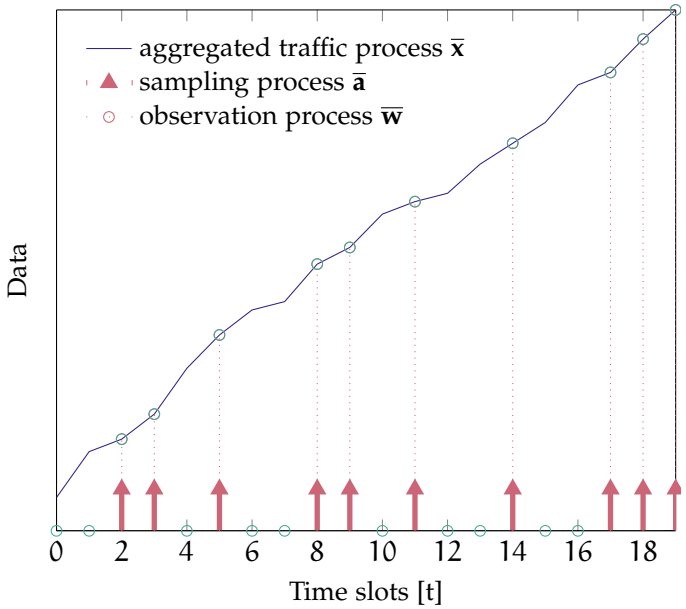


Figure 32: Cumulative arrival process and corresponding observation process obtained from samples with randomly distributed inter-query times.

5.5.1 *Relationship to Packet Sampling*

We begin our analysis by describing the notation used in the this section and outlining the relationship between the covariance structure of stationary increment processes, which we assumed in the previous section, and cumulative processes which are observed by traffic counters.

A switch flow counter aggregates the increments of the associated traffic process over time, i.e., at time t it contains the accumulated amount of data that arrived up to t. In terms of the increment process $\mathbf{x}$ the elements of the counter process $\overline{\mathbf{x}}$ are given as $\overline{x}_t = \sum_{k=1}^{t} x_k$, where x_k denotes the traffic increment at time slot k. In the following, we denote this cumulative counter process using vector notation as $\overline{\mathbf{x}} = \boldsymbol{\Delta}\mathbf{x}$, where $\boldsymbol{\Delta}$ is a lower triangular matrix containing ones for all matrix elements on or below the matrix diagonal:

$$\boldsymbol{\Delta} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 1 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix}.$$

Note, that $\overline{x}_t$ corresponds to the cumulative arrival process $A(t)$ in Eq. (5). Generally, a reversal of the counter aggregation, i.e., $\mathbf{x} = \boldsymbol{\Delta}^{-1}\overline{\mathbf{x}}$, is only possible if the counter values $\overline{x}_t$ are sampled[5] at each discrete time slot t. However, this is not the case in out current scenario where we consider random inter query intervals.

5 In the sequel, we use the terms counter sample, query, and observation interchangeably.

Recall that the covariance matrix $\mathbf{\Sigma}$ of the stationary, non-aggregated traffic increment process has a Toeplitz structure. Thus, each matrix element Σ_{ij} is defined in terms of the process autocovariance as $\Sigma_{ij} = c_x(|i-j|)$, where $c_x(\tau)$ denotes the autocovariance at lag τ. Unless stated otherwise, in the sequel we assume zero mean increment processes, i.e., the expected value $\mu_x = E[x_k]$ has been subtracted from the process. The relationship between the covariance matrix $\mathbf{\Sigma}$ of the traffic increment process $\mathbf{x}$ and the covariance matrix $\overline{\mathbf{\Sigma}}$ of the corresponding cumulative counter process $\overline{\mathbf{x}}$ is given by

$$\overline{\mathbf{\Sigma}} = E\left[\overline{\mathbf{x}}\overline{\mathbf{x}}^\top\right] = \mathbf{\Delta}\, E\left[\mathbf{x}\mathbf{x}^\top\right]\mathbf{\Delta}^\top = \mathbf{\Delta}\mathbf{\Sigma}\mathbf{\Delta}^\top. \tag{42}$$

Next, in analogy to Eq. (15), we define an observation process $\overline{\mathbf{w}}$ which is obtained from the element-wise multiplication of the cumulative traffic process $\overline{\mathbf{x}}$ with a point sampling process represented by the vector $\mathbf{a}$ with random inter query times:

$$\overline{\mathbf{w}} = \mathbf{a} \circ \overline{\mathbf{x}} = \mathbf{A}\overline{\mathbf{x}}, \tag{43}$$

using the operator $(\circ)$ to denote the Hadamard product of two vectors. Equivalently, $\overline{\mathbf{w}}$ may be expressed by a matrix multiplication of the diagonal matrix $\mathbf{A}$ with $\mathrm{diag}(\mathbf{A}) = \mathbf{a}$ with $\overline{\mathbf{x}}$. In the sequel, we assume renewal sampling and that the inter query intervals are drawn from a known probability distribution which yields a sampling intensity $\mu_{\mathbf{a}} = E[\mathbf{a}] \in [0, 1]$. We denote the covariance matrix of the observation process $\overline{\mathbf{w}}$ as $\overline{\mathbf{\Sigma}}_{\mathbf{W}}$. An exemplary observation process obtained from random samples is depicted in Fig. 32.

Next, we highlight that the counter sampling approach presented in this section differs from the sampling procedure outlined in Section 5.4, where we consider the effects of random sampling on non-aggregated traffic increment processes. As a consequence the previously derived methods are not directly applicable. We summarize the difference between the two problem formulations in the following lemma

Lemma 5.5. *The autocovariance matrix* $\overline{\mathbf{\Sigma}}_{\mathbf{W}}$ *of the* aggregated *observation process* $\overline{\mathbf{w}} = \mathbf{A}\overline{\mathbf{x}}$ *cannot be derived from the autocovariance matrix* $\mathbf{\Sigma}_{\mathbf{W}}$ *of the* increment *observation process* $\mathbf{w} = \mathbf{A}\mathbf{x}$ *using Eq.* (42)*. Specifically the following relationship does not hold*

$$\overline{\mathbf{\Sigma}}_{\mathbf{w}} \neq \mathbf{\Delta}\mathbf{\Sigma}_{\mathbf{w}}\mathbf{\Delta}^\top$$

Proof.

$$\begin{aligned}\overline{\mathbf{\Sigma}}_{\mathbf{w}} &= E\left[\mathbf{A}\overline{\mathbf{x}}\overline{\mathbf{x}}^\top\mathbf{A}^\top\right] = E\left[\mathbf{A}\mathbf{\Delta}\mathbf{x}(\mathbf{A}\mathbf{\Delta}\mathbf{x})^\top\right] = E\left[\mathbf{A}\mathbf{\Delta}\mathbf{x}\mathbf{x}^\top\mathbf{\Delta}^\top\mathbf{A}^\top\right] \\ &\neq \mathbf{\Delta}\mathbf{\Sigma}_{\mathbf{w}}\mathbf{\Delta}^\top = \mathbf{\Delta}\, E\left[\mathbf{A}\mathbf{x}\mathbf{x}^\top\mathbf{A}^\top\right]\mathbf{\Delta}^\top = E\left[\mathbf{\Delta}\mathbf{A}\mathbf{x}\mathbf{x}^\top\mathbf{A}^\top\mathbf{\Delta}^\top\right]\end{aligned}$$

□

The relationship above shows that sampling after aggregation is structurally different than aggregation after sampling. Autocovariance matrix reconstruction of sampled traffic aggregates that are shown in this work are more involved than the reconstruction after sampling at the increment level as in [124].

Initially, we consider a baseline scenario wherein queries to the counter associated with some traffic flow are generated at equidistant times, i.e., we assume that the observation process contains only ones and is hence represented by the identity matrix $\mathbf{A} = \mathbf{I}$. We focus on the case where the intervals between counter queries are selected from a random distribution, in Section 5.5.5.

In the following, we outline two approaches for obtaining the covariance structure of an observed network flow.

5.5.2 *Variance Sampling*

First, we formulate an estimation approach which exploits the relationship between the variance decay of a process aggregated over different time scales and its autocovariance, given in Eq. (30).

Let the random variable $\overline{x}_M$ denote an element of the aggregated vector $\overline{\mathbf{x}}$ over some aggregation interval $M \geqslant 1$ as outlined in Section 5.4.3, i.e., $\overline{x}_M = \sum_{t=1}^{M} x_t$. Then the aggregated variance metric from Section 5.4.3 may be written as $\sigma_M^2 = \mathrm{Var}\left[\frac{1}{M}\overline{x}_M\right]$, revealing that the scaled aggregated variance corresponds to the diagonal values of the covariance matrix $\overline{\mathbf{\Sigma}}$, i.e., $M^2\sigma_M^2 = \mathrm{Var}[\overline{x}_M] = \Sigma_{ii}$.

The aggregation interval M may be interpreted as the time between two subsequent controller generated queries to the counters measuring the traffic process $\mathbf{x}$. For any aggregation interval length M, Eq. (30) yields the following relationship between the variance $\mathrm{Var}[\overline{x}_M] = \sigma^2_{\overline{x}_M}$ of the observed process and its covariance. Using matrix notation we write

$$\begin{bmatrix} \sigma^2_{\overline{x}_1} \\ \sigma^2_{\overline{x}_2} \\ \sigma^2_{\overline{x}_3} \\ \vdots \\ \sigma^2_{\overline{x}_M} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 2 & 2 & 0 & \cdots & 0 \\ 3 & 4 & 2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ M & 2M-2 & 2M-4 & \cdots & 2 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{(M-1)} \end{bmatrix}, \tag{44}$$

where c_τ represents the elements of the (Toeplitz) covariance matrix $\Sigma_{ij} = c_{|i-j|}$ of the increment process $\mathbf{x}$. From Eq. (44) it follows that the autocovariance of the monitored process may be extracted from the variances $\sigma^2_{\overline{x}_M}$ of the monitored flow $\mathbf{x}$ over increasing intervals M. To

this end, we invert the $(M \times M)$ matrix in Eq. (44) to obtain the following expression for the autocovariance elements

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{(M-1)} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ -1 & 1/2 & 0 & \cdots & 0 \\ 1/2 & -1 & 1/2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 1/2 & -1 & 1/2 \end{bmatrix} \begin{bmatrix} \sigma^2_{\overline{x}_1} \\ \sigma^2_{\overline{x}_2} \\ \sigma^2_{\overline{x}_3} \\ \vdots \\ \sigma^2_{\overline{x}_M} \end{bmatrix}. \tag{45}$$

In other words, the Eq. (45) enables us to extract the autocovariance structure of an arbitrary flow by monitoring the corresponding flow counters and estimating the variances over different time intervals. The corresponding aggregation intervals, i.e., the times between queries, are chosen by the controller. Unfortunately in practical scenarios, where the number of available traffic samples is limited the above approach is not numerically stable. This limitation is due to the differencing operations in Eq. (45) which amplify the errors of the variance estimates that necessarily arise for small sample sizes. Therefore, in the following section we consider an alternative approach, which estimates the covariance matrix $\overline{\mathbf{\Sigma}}$ directly.

5.5.3 *Covariance Matrix Sampling*

In order to estimate the traffic covariance matrix $\overline{\mathbf{\Sigma}}$ from a finite number of counter queries we observe the counter process $\overline{\mathbf{x}} = (\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_T)^\top$ over a finite monitoring duration of T time slots. We employ a sliding window vector containing the $T_{win} \ll T$ most recent counter readings of the process $\overline{\mathbf{x}}$ at time $t \in [T_{win}, T]$ in order to derive the sample covariance matrix $\widetilde{\overline{\mathbf{\Sigma}}}$ of $\overline{\mathbf{x}}$. We denote the sliding window vector at time t as $\overline{\mathbf{x}}^{[t]} = (\overline{x}_{t-T_{win}+1}, \overline{x}_{t-T_{win}+2}, \ldots, \overline{x}_{t-1}, \overline{x}_t)^\top - \overline{x}_{t-T_{win}}$. At each time slot t, $\overline{x}_{t-T_{win}}$ is subtracted from the collected counter observations to obtain the cumulative arrivals within the sliding widow interval T_{win}. Note, that $E[\overline{x}_{t-T_{win}}] = E\left[\sum_{i=1}^{t-T_{win}} x_i\right] = (t - T_{win})\mu_X$

Now, we obtain an estimate for each element $\widetilde{\overline{\Sigma}}_{ij}$ of the sample covariance matrix $\widetilde{\overline{\mathbf{\Sigma}}}$ as

$$\widetilde{\overline{\Sigma}}_{ij} = \frac{1}{T - T_{win} + 1} \left(\sum_{t=T_{win}}^{T} \widetilde{\overline{x}}_i^{[t]} \widetilde{\overline{x}}_j^{[t]} \right) - ij\widetilde{\mu}_X^2, \tag{46}$$

where, $\widetilde{\overline{x}}_n^{[t]} = \tilde{\overline{x}}_{t-T_{win}+n}$ represents a realization of the random variable at the n^{th} element of the sliding window vector $\overline{\mathbf{x}}^{[t]}$ and $\widetilde{\mu}_X = \tilde{\overline{x}}_{T_{win}}/T_{win}$ is the sample mean of the increment process. The dimensions of the estimated covariance matrix are determined by the window length, i.e., $\widetilde{\overline{\mathbf{\Sigma}}} \in \mathbb{R}^{T_{win} \times T_{win}}$.

The validity of the estimator Eq. (46) follows from the stationarity of the non-aggregated traffic process $\mathbf{x}$. Using $\mathbf{x}^{[t]}$ to denote vector of T_{win} non-aggregated elements from $\mathbf{x}$ starting at time t, it follows from Eq. (42) that $\Delta \, \mathrm{E}\left[\mathbf{x}^{[t]}\mathbf{x}^{[t]\top}\right]\Delta^\top = \mathrm{E}\left[\overline{\mathbf{x}}^{[t]}\overline{\mathbf{x}}^{[t]\top}\right] = \overline{\mathbf{\Sigma}}$ for any t.

5.5.4 *Bias of the Covariance Matrix Estimator*

We seek to find the bias of the elements $\widetilde{\overline{\Sigma}}_{ij}$ of the sample covariance matrix $\widetilde{\overline{\mathbf{\Sigma}}}$ for $i, j \in [1, T_{win}]$. To this end we take the expected value of the estimator Eq. (46). We substitute $k = t - T_{win}$ and represent the sliding window vector as $\overline{\mathbf{x}}^{[t]} = (\overline{x}_{k+1}, \overline{x}_{k+2}, \ldots, \overline{x}_{k-T_{win}-1}, \overline{x}_{k-T_{win}})^\top - \overline{x}_k$. Further, we use the following relationships between the increment and cumulative processes: $\overline{x}_{k+1} - \overline{x}_k = x_{k+1}$ and $\overline{x}_{k+i} - \overline{x}_k = \sum_{l=k+1}^{k+i} x_l$. Realizations of the random variables x_k and $\overline{x}_k$ are denoted as $\widetilde{x}_k$ and $\widetilde{\overline{x}}_k$, respectively. Thus, the expected value of the estimator is given by

$$\begin{aligned}
\mathrm{E}\left[\widetilde{\overline{\Sigma}}_{ij}\right] &= \mathrm{E}\left[\frac{1}{T - T_{win} + 1}\sum_{k=0}^{T-T_{win}} (\widetilde{\overline{x}}_{k+i} - \widetilde{\overline{x}}_k)(\widetilde{\overline{x}}_{k+j} - \widetilde{\overline{x}}_k)\right] - ij\,\mathrm{E}\left[\widetilde{\mu}_X^2\right] \\
&= \frac{1}{T - T_{win} + 1}\,\mathrm{E}\left[\sum_{k=0}^{T-T_{win}} \left(\sum_{l=k+1}^{k+i} \widetilde{x}_l\right)\left(\sum_{m=k+1}^{k+j} \widetilde{x}_m\right)\right] - ij\,\mathrm{E}\left[\widetilde{\mu}_X^2\right] \\
&= \frac{1}{T - T_{win} + 1}\sum_{k=0}^{T-T_{win}} \left(\sum_{l=1}^{i}\sum_{m=1}^{j} \mathrm{E}[\widetilde{x}_{k+l}\widetilde{x}_{k+m}]\right) - ij\,\mathrm{E}\left[\widetilde{\mu}_X^2\right].
\end{aligned}$$

We substitute the relationship $\mathrm{E}[\widetilde{x}_{k+l}\widetilde{x}_{k+m}] = c_X(l - m) + \mu_X^2$ into the equation above to obtain

$$\mathrm{E}\left[\widetilde{\overline{\Sigma}}_{ij}\right] = \sum_{l=1}^{i}\sum_{m=1}^{j} [c_X(l - m)] + ij\mu_X^2 - ij\,\mathrm{E}\left[\widetilde{\mu}_X^2\right]. \tag{47}$$

Further, we express the elements $\overline{\Sigma}_{ij}$ of the autocovariance matrix of the cumulative process $\overline{\mathbf{x}}$ in terms of the autocovariance Σ_{ij} of the increment process as

$$\overline{\Sigma}_{ij} = \sum_{l=1}^{i}\sum_{m=1}^{j} \Sigma_{lm} = \sum_{l=1}^{i}\sum_{m=1}^{j} c_X(l - m). \tag{48}$$

Finally, substituting Eq. (48) and $\mathrm{E}\left[\widetilde{\mu}_X^2\right] = \mathrm{Var}[\widetilde{\mu}_X] + \mu_X^2$ into Eq. (47) yields the following bias for the autocovariance matrix estimator

$$\mathrm{E}\left[\widetilde{\overline{\Sigma}}_{ij}\right] = \overline{\Sigma}_{ij} - ij\,\mathrm{Var}[\widetilde{\mu}_X]. \tag{49}$$

Again, we find that the bias depends on the variance of the sample mean. As this variance tends towards zero for $T \to \infty$ the estimator is asymptotically unbiased[6].

6 Note, that with $M = i = j$ we recover the bias of the aggregated variance estimators where $\mathrm{E}\left[\widetilde{\sigma}_M^2\right] = \mathrm{E}\left[\widetilde{\overline{\Sigma}}_{MM}/M^2\right]$.

5.5.5 *Random Inter Query Times*

Using Eq. (46) we obtain an estimate of the traffic covariance matrix based on flow counter values queried at equidistant times. However, we are interested in minimizing the monitoring load on switches and controllers by reducing the query intensity. At the same time, we wish to avoid sacrificing the ability to capture flow characteristics at small time scales. To this end, the controller chooses the intervals between subsequent counter queries from a known probability distribution. In Section 5.4 we showed that for packet sampling the distortions introduced by the sampling process are reversible. In this section, we seek to find a similar result for the case of counter sampling which we showed to be structurally different in Lemma 5.5.

Consider the case where the cumulative counter process $\overline{\mathbf{x}}$ is observed at random intervals. We model the sampling strategy employed at the controller as a random vector $\mathbf{a}$, with elements $a_k \in \{0, 1\}$, where a value of one indicates that a counter sample was taken at time slot k. Inter query times are independent and identically distributed (i.i.d.) according to some distribution $F_{\mathbf{a}}$. We define the query intensity μ_a as the fraction of time slots at which counter queries were generated by the controller. In the following, we evaluate the effects of the sampling strategy on the estimated covariance matrix. We show that the resulting distortion can be reversed in the case of geometric sampling.

Analogous to the sliding window vector $\overline{\mathbf{x}}^{[t]}$ defined in the previous section, let $\overline{\mathbf{w}}^{[t]}$ denote the sliding window vector containing T_{win} preceding elements of the monitored traffic process at time t observed through the sampling process $\mathbf{a}$. Specifically, the elements of $\overline{\mathbf{w}}^{[t]}$ obtained from counter observations are given by

$$\overline{w}_l^{[t]} = \begin{cases} a_{t-T_{\text{win}}+l}(\overline{x}_{t-T_{\text{win}}+l} - \overline{x}_{t-T_{\text{win}}}) & \text{if} \quad a_{t-T_{\text{win}}} = 1 \\ 0 & \text{if} \quad a_{t-T_{\text{win}}} = 0, \end{cases} \tag{50}$$

with $l \in [1, T_{\text{win}}]$ and $t \in [T_{\text{win}}, T]$. However, note that when the counter query intervals are random, the sliding window $\overline{\mathbf{w}}^{[t]}$ can only be shifted to time points at which a counter reading of the observed flow was collected. Specifically, the sliding window vector is evaluated only when $a_{t-T_{\text{win}}} = 1$, as the corresponding reference value $x_{t-T_{\text{win}}}$ is necessary to obtain the cumulative traffic arrivals within the sliding window interval. The vector elements are set to zero otherwise.

The counter sampling algorithm is represented graphically in Fig. 33. At time $t = 13$ the sliding window vector $\widetilde{\overline{\mathbf{w}}}^{[13]}$ is highlighted in red. Its vector elements contain a zero whenever the cumulative data at the corresponding time slot is not known. The evaluated aggregation intervals for all other positions of the sliding window are illustrated below the time axis.

Clearly, the sampling process introduces a significant number of zero elements into the sliding window vector $\overline{\mathbf{w}}^{[t]}$. Hence, the resulting covariance matrix estimate is distorted and must be corrected to reverse

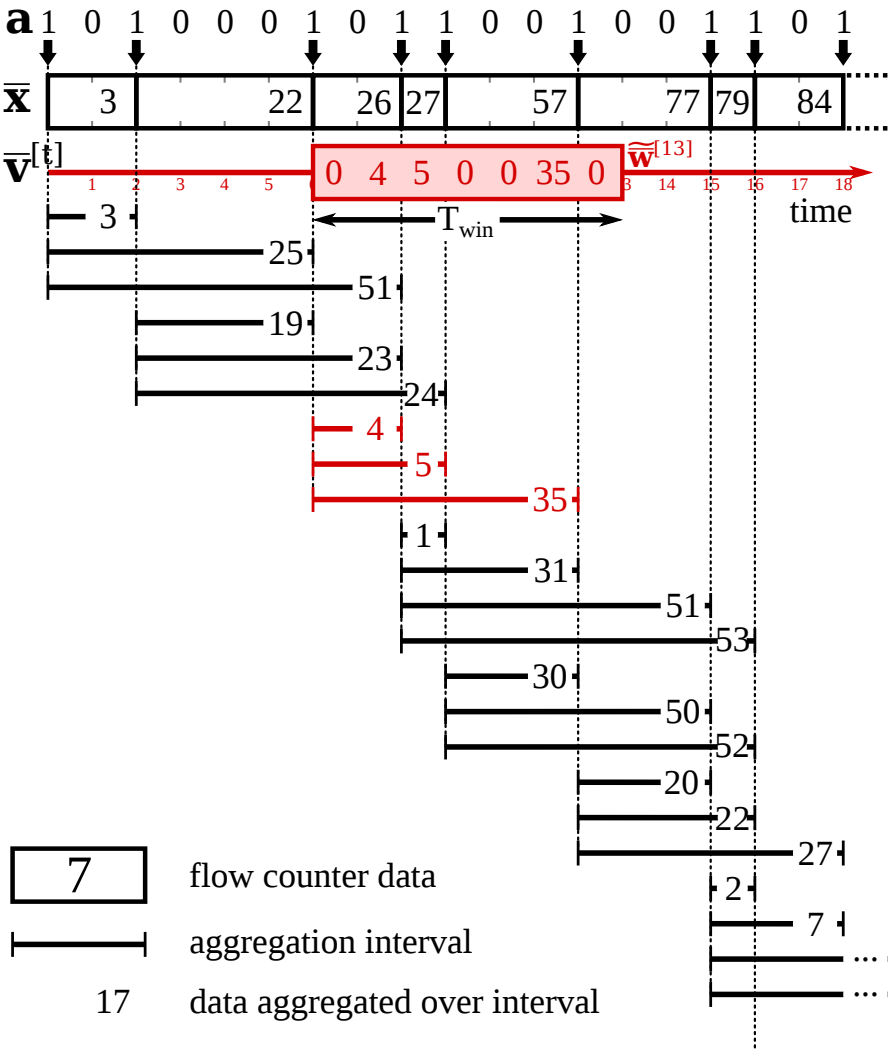


Figure 33: Algorithm for random sampling of flow counters using a sliding window with length $T_{win} = 8$.

the impact of sampling. Intuitively, the distortion is due to the fact that the sampling process generates a different number of samples for each element of the covariance matrix.

The relationship between the covariance of the monitored process $\bar{\mathbf{x}}$ and the covariance of the sampled process is described by the following lemma:

Lemma 5.6. *Let the random vectors $\bar{\mathbf{x}}$ and $\mathbf{a}$ denote stationary and independent traffic and sampling processes, respectively. The relationship between the traffic covariance matrix $\overline{\mathbf{\Sigma}}$ of $\bar{\mathbf{x}}$ and the covariance matrix $\overline{\mathbf{\Sigma}}_{\overline{\mathbf{w}}}$ of the observation process $\overline{\mathbf{w}}$ is given as*

$$\overline{\mathbf{\Sigma}}_{\overline{\mathbf{w}}} = \mathbf{K} \circ \overline{\mathbf{\Sigma}},$$

where the coefficient matrix $\mathbf{K}$ has the elements $K_{ij} = E\left[a_k a_{k+i} a_{k+j}\right]$ and a_k are the elements of the sampling process $\mathbf{a}$ for times $k \in [T_{win}, \infty]$

Proof. Let $k = t - T_{win}$ then we obtain $\overline{x}_l^{[k+T_{win}]} = \overline{x}_{k+l} - \overline{x}_k$ and $\overline{w}_l^{[k+T_{win}]} = a_{k+l}(\overline{x}_{k+l} - \overline{x}_k)$ from Eq. (50).

$$\begin{aligned}
\overline{\Sigma}_{\overline{\mathbf{w}}_{ij}} &= \mathrm{E}\Big[a_k \overline{w}_i^{[k+T_{win}]} \overline{w}_j^{[k+T_{win}]}\Big] \\
&= \mathrm{E}\big[a_k a_{k+i}(\overline{x}_{k+i} - \overline{x}_k) a_{k+j}(\overline{x}_{k+j} - \overline{x}_k)\big] \\
&= \mathrm{E}\big[a_k a_{k+i} a_{k+j}\big]\, \mathrm{E}\big[(\overline{x}_{k+i} - \overline{x}_k)(\overline{x}_{k+j} - \overline{x}_k)\big] \\
&= \mathrm{E}\big[a_k a_{k+i} a_{k+j}\big]\, \mathrm{E}\Big[\overline{x}_i^{[t]} \overline{x}_j^{[t]}\Big] = K_{ij} \overline{\Sigma}_{ij}
\end{aligned}$$

□

Next, we consider a Bernoulli sampling process with geometrically distributed inter sampling intervals with parameter $p = \mu_a \in [0, 1]$. As the process is memoryless we may write $\mathrm{E}\big[a_k a_{k+i} a_{k+j}\big] = \mathrm{E}[a_k]\, \mathrm{E}[a_{k+i}]\, \mathrm{E}\big[a_{k+j}\big]$ and $\mathrm{E}[a_k] = \mathrm{E}[a_{k+i}] = \mathrm{E}\big[a_{k+j}\big] = p$. Consequently, we obtain

$$\overline{\Sigma}_{\overline{\mathbf{w}}_{ij}} = \begin{cases} \mathrm{E}[a_k]\, \mathrm{E}\big[a_{k+i}^2\big] \overline{\Sigma}_{ii} = p^2 \overline{\Sigma}_{ii} & : i = j \\ \mathrm{E}[a_k]\, \mathrm{E}[a_{k+i}]\, \mathrm{E}\big[a_{k+j}\big] \overline{\Sigma}_{ij} = p^3 \overline{\Sigma}_{ij} & : i \neq j. \end{cases} \tag{51}$$

Thus, we adapt Eq. (46) to obtain the elements of the sample autocovariance matrix $\widetilde{\overline{\mathbf{\Sigma}}}$ estimated from a sliding window of randomly sampled observations over a time interval T are given as

$$\widetilde{\overline{\Sigma}}_{ij} = \frac{1}{K_{ij}} \widetilde{\overline{\Sigma}}_{\mathbf{w}ij} = \frac{1}{K_{ij}(T - T_{win} + 1)} \left(\sum_{t=T_{win}}^{T} \widetilde{\overline{w}}_i^{[t]} \widetilde{\overline{w}}_j^{[t]} \right) - ij \widetilde{\mu}_w^2, \tag{52}$$

where $\widetilde{\overline{w}}_n^{[t]}$ denotes a realization of the random variable from Eq. (50), i.e., the n^{th} element of the sliding window vector $\overline{\mathbf{w}}^{[t]}$, and K_{ij} is a element-wise correction factor with $K_{i=j} = p^2$ and $K_{i\neq j} = p^3$ according to Eq. (51). Further, we use $\widetilde{\mu}_w = \widetilde{\overline{x}}_{t^*}/t^*$ to represent the sample mean of the increment process where t^* denotes the time of the most recent observation. As $\mathrm{E}[t^*] = T - \frac{1}{p}$ we approximate $T \approx t^*$ for $T \gg \frac{1}{p}$ and hence $\widetilde{\mu}_w \approx \widetilde{\mu}_x$. Consequently, we may approximate the bias of the estimator with the bias derived for the non-sampled case given in Eq. (49).

Finally, note that from Lemma 5.6 it follows that a reconstruction of the sampled covariance matrix is not restricted to Bernoulli sampling processes. However the analytic evaluation of the term $\mathrm{E}\big[a_k a_{k+i} a_{k+j}\big]$ is significantly simplified for memoryless sampling distributions.

5.5.6 *Impact of Random Sampling*

In the following, we evaluate the impact of the sampling intensity p on the estimate of the covariance matrix, given a finite measurement duration T using simulations. To this end, we consider a synthetic random process[7] with known covariance matrix $\overline{\mathbf{\Sigma}}$. We generate sample paths

7 We generate fBm sample paths with the following parameters: $H = 0.8$, $\sigma = 4$, $\mu = 0$. We consider processes with non-Gaussian increments in Section 5.6.3.

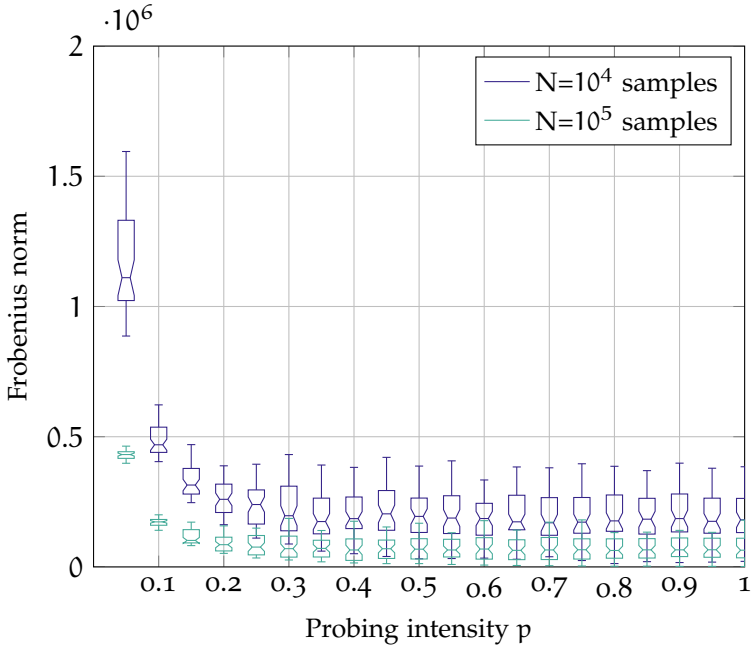


Figure 34: Effects of sampling intensity and sampling duration on the covariance matrix estimate. Deviation between the analytical and empirical covariance matrices for synthetic traffic.

of the process and randomly sample each one using geometrically distributed inter sample intervals with sampling intensities $p \in [0.05, 1]$. Next, we estimate the empirical covariance matrix $\widetilde{\overline{\Sigma}}_p$ for each sampled trace as outlined above. Finally, to quantify the similarity between the analytical covariance matrix and its estimate derived with sampling we calculate the Frobenius norm $||\overline{\Sigma} - \widetilde{\overline{\Sigma}}_p||_F$. For a matrix $\mathbf{U}$ the Frobenius norm $||\mathbf{U}||_F$ is defined as $\sqrt{\sum_i \sum_j |U_{ij}|^2}$. The experiment is repeated 50 times using different realizations of the sampling process. In Fig. 34 we depict box plots of the calculated norms. The first insight we draw from the figure is that for a fixed monitoring duration T random sampling as proposed is highly beneficial. The plateaus indicate that we can save up to 80-90% of the samples without significantly degrading the quality of the estimate. The second effect seen in Fig. 34 is that the quality of the estimate rises, as expected, for longer sampling intervals T.

5.6 SAMPLE PATH GENERATION

The sampling approaches outlined above enable us to reverse the effects of the sampling process and to derive an estimate of the covariance matrix for any monitored flow. We now address the problem of generating independent sample paths which possess the same correlation characteristics as the observed traffic process $\bar{x}$. We assume that the covariance matrix $\overline{\Sigma}$ of the observed flow is known or has been estimated using the techniques presented in the previous sections.

5.6.1 *Cholesky Decomposition*

In order to synthesize an arbitrary number of sample paths $\widetilde{\overline{\mathbf{y}}}$ from the estimated covariance matrix of an observed flow, we make use of an approach based on the Cholesky decomposition [62] which is applicable to any square, symmetric, positive definite matrix. Given an autocovariance matrix $\overline{\boldsymbol{\Sigma}}$, which meets these requirements, the Cholesky decomposition yields a unique lower triangular matrix $\mathbf{L}$ such that $\mathbf{L}$ contains only positive values and

$$\mathbf{L}\mathbf{L}^\top = \overline{\boldsymbol{\Sigma}}.$$

Next, let $\mathbf{z} = (z_1, z_2, \ldots, z_{T_{\text{win}}})^\top$ denote a column vector of uncorrelated random variables z_l with zero mean and unit variance, i.e., $\mathrm{E}[z_l] = 0$, $\mathrm{E}\left[z_l^2\right] = 1$ and consequently $\mathrm{E}\left[\mathbf{z}\mathbf{z}^\top\right] = \mathbf{I}$. Further, assume that the elements z_t are independent and drawn from the normal distribution. We can then generate a new process $\overline{\mathbf{y}}$ using

$$\overline{\mathbf{y}} = \mathbf{L}\mathbf{z}. \tag{53}$$

The covariance matrix of this process is given as

$$\begin{aligned}\mathrm{E}\left[\overline{\mathbf{y}}\overline{\mathbf{y}}^\top\right] &= \mathrm{E}\left[(\mathbf{L}\mathbf{z})(\mathbf{L}\mathbf{z})^\top\right] = \mathrm{E}\left[\mathbf{L}\mathbf{z}\mathbf{z}^\top\mathbf{L}^\top\right] = \mathbf{L}\,\mathrm{E}\left[\mathbf{z}\mathbf{z}^\top\right]\mathbf{L}^\top \\ &= \mathbf{L}\mathbf{I}\mathbf{L}^\top = \overline{\boldsymbol{\Sigma}}.\end{aligned} \tag{54}$$

In other words, the process $\overline{\mathbf{y}}$ possesses the same autocovariance structure as the observed flow $\overline{\mathbf{x}}$. As a result, we may synthesize an arbitrary number of sample paths $\widetilde{\overline{\mathbf{y}}}$ using different realizations $\widetilde{\overline{\mathbf{z}}}$ of the random variables in $\mathbf{z}$.

5.6.2 *Positive Definiteness of the Sample Autocovariance*

A prerequisite for the Cholesky decomposition is that the factorized matrix is positive-definite. A matrix is positive definite (PD)[8] if it is symmetric and all its eigenvalues are positive.

While the (distorted) empirical covariance matrix $\widetilde{\overline{\boldsymbol{\Sigma}}}_{\mathbf{w}}$ is PD by construction, the corrected estimate $\mathbf{Q} \circ \widetilde{\overline{\boldsymbol{\Sigma}}}_{\mathbf{w}}$, with coefficient matrix $\mathbf{Q} = (Q_{ij}) = (1/K_{ij})$, is only guaranteed to be PD when $\mathbf{Q} \succ 0$ (this follows from the Schur product theorem). Unfortunately for Bernoulli sampling processes the coefficient matrix in Eq. (52) is not PD.

A number of approaches for finding the nearest positive definite matrix have been proposed [82, 72]. In this work, we rely on semi-definite programming in order to ensure the positive-definiteness property of the covariance matrix estimate. Specifically we use CVX, a package for solving convex programs [64, 63].

8 We use the notation $(\cdot) \succ 0$ to denote that a matrix $(\cdot)$ is PD.

To obtain a PD autocovariance matrix we formulate the following minimization problem

$$\|\boldsymbol{\Delta}\mathbf{S}\boldsymbol{\Delta}^\top - \widetilde{\widetilde{\boldsymbol{\Sigma}}}\|_F, \quad \text{s.t.} \quad \mathbf{S} \succ 0, \tag{55}$$

where the optimization variable $\mathbf{S}$ is a $T_{win} \times T_{win}$ symmetric Toeplitz matrix. This optimization formulation has a complexity of $O(T_{win}^2)$ where T_{win} corresponds to the largest evaluated lag in the autocovariance function. A less computationally intensive formulation (with $O(T_{win})$ complexity) may be obtained from Eq. (44), which relates the autocovariance of the increment process to the variances of the cumulative process at different aggregation intervals. These variances correspond to the diagonal of the sample autocovariance matrix $\widetilde{\boldsymbol{\sigma}} = \text{diag}(\widetilde{\widetilde{\boldsymbol{\Sigma}}}) = (\widetilde{\widetilde{\Sigma}}_{11}, \widetilde{\widetilde{\Sigma}}_{22}, \ldots, \widetilde{\widetilde{\Sigma}}_{T_{win}T_{win}})^\top$. Thus, instead of minimizing the Frobenius norm in Eq. (55), we may minimize the norm of an T_{win}-dimensional vector to obtain a PD estimate of the autocovariance matrix $\boldsymbol{\Sigma}$. Specifically, we note that if the Toeplitz optimization variable $\mathbf{S}$ defined above is an estimate of the autocovariance matrix $\boldsymbol{\Sigma}$, its first column is an estimate of the autocovariance $(c_X(0), c_X(1), \ldots, c_X(T_{win}))^\top$. Let $\mathbf{s}$ denote this vector, then using $\mathbf{U}$ to denote the $T_{win} \times T_{win}$ matrix from Eq. (44)

$$\mathbf{U} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 2 & 2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ T_{win} & 2T_{win}-2 & \cdots & 2 \end{bmatrix} \quad \text{and} \quad \mathbf{V} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 2^{-2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & T_{win}^{-2} \end{bmatrix},$$

we formulate the following optimization problem:

$$\|\mathbf{VUs} - \mathbf{V}\widetilde{\boldsymbol{\sigma}}\|_2, \quad \text{s.t.} \quad \mathbf{S} \succ 0. \tag{56}$$

In other words, we recover $\mathbf{S}$ from the Toeplitz structure of the autocovariance matrix $\boldsymbol{\Sigma}$. In both cases, in subsequent calculations we replace the sample autocovariance $\widetilde{\widetilde{\boldsymbol{\Sigma}}}$ with the positive-definite estimate $\boldsymbol{\Delta}\mathbf{S}\boldsymbol{\Delta}^\top$ from the solver.

5.6.3 *Reproducing the Traffic Increment Distribution*

So far we focused on the reproduction of the autocovariance structure of the observed traffic flow. Note, that the autocovariance relationship in Eq. (54) holds regardless of the probability distribution of the increments of the traffic process $\bar{\mathbf{y}}$. Thus, in order to ensure that the synthesized sample paths accurately mirror the characteristics of the observed flow, we additionally consider the increment distribution of the observed process over different aggregations intervals. In the sequel, we evaluate conditions under which the factorization approach outlined above yields sample paths which simultaneously capture the autocovariance and the increment distribution of the observed process.

We first consider the case where the elements z_l are normally distributed and independent. From Eq. (53) it follows that the elements y_l of the process $\mathbf{y}$ are comprised of the weighted sum $y_l = \sum_{s=1}^{l} L_{ls} z_l$, where L_{ij} denotes a matrix element of $\mathbf{L}$. Therefore, the distribution of the increments y_l is determined by the distribution of the elements z_l of the generating random vector $\mathbf{z}$. Consequently, if the vector elements z_l are drawn from a standard normal distribution the increments of the sample path vector $\bar{\mathbf{y}}$ will also be normally distributed due to the additive property of Gaussian variables. Specifically, since $\mathrm{Var}[L_{ls} z_l] = L_{ls}^2 \mathrm{Var}[z_l]$ and $\mathrm{Var}[z_l] = 1$ the elements of y_l will be normally distributed with

$$y_l \sim \mathcal{N}\left(0, \sum_{s=1}^{l} L_{ls}^2\right).$$

Thus, in order to generate sample paths with Gaussian increments, such as fBm, it is sufficient to draw normally distributed values in $\mathbf{z}$.

Since the seminal paper [85] numerous works [106, 77, 145, 38] have tested and adopted the normal distribution as a model for Ethernet LAN/WAN traffic. For sufficiently large aggregation intervals the distribution of the increments of any traffic flow $\bar{\mathbf{x}}$ tends towards a normal distribution. This behavior, which arises in its simplest form as a consequence of the central limit theorem, has been examined together with other traffic properties such as LRD under various conditions, e.g., in [137, 110]. Hence, the use of the normal distribution is justified for cases where the minimum time slot between counter queries is sufficiently large or when the queried flow is an aggregate of many fine grained flows. For very fine timescales, empirical increment distributions were shown to deviate considerably from the Gaussian model, e.g., in [77, 67]. Hence, for traffic flows with an increment distribution which deviates significantly from the normal distribution additional steps must be taken to ensure that the synthesized sample paths accurately reflect the characteristics of the considered flow.

Next, we consider processes $\bar{\mathbf{x}}$ with arbitrary, non-Gaussian increments. Recall, that Eq. (53) may be rearranged to obtain a generating vector $\mathbf{z} = \mathbf{L}^{-1}\bar{\mathbf{x}}$ with uncorrelated elements, unit variance and zero mean, as well as an increment distribution which matches the distribution of the process increments. However, in the general case the elements of $\mathbf{z}$ are not guaranteed to be independent. Thus, as the conditional probabilities associated with the elements of $\mathbf{z}$ are typically unknown the generation of independent sample paths is not feasible. Indeed, in the sequel we show that the generation of sample paths with a prescribed covariance structure from *independently* drawn random variables z_t is only possible if these variables are drawn from a normal distribution. We propose a workaround for this restriction in Section 5.6.4.

Consider the random increment process vector $\mathbf{y}$ with covariance matrix $\mathbf{\Sigma} = \mathbf{M}\mathbf{M}^\top$ and $\mathbf{y} = \mathbf{M}\mathbf{z}$. Clearly, the random vector $\mathbf{z}$ may be used

to generate $\mathbf{y}$ as well as the corresponding aggregated traffic process $\overline{\mathbf{y}}$ using

$$\Delta \mathbf{M}\mathbf{z} = \Delta \mathbf{y} = \overline{\mathbf{y}}.$$

Next, note that due to the stationarity of the traffic increment process $\mathbf{y}$ any two vector elements y_r, y_s are identically distributed. Hence the following relationship must hold:

$$\begin{aligned} y_r &\overset{d}{=} y_s \\ \sum_{t=1}^{r} M_{rt} z_t &\overset{d}{=} \sum_{t=1}^{s} M_{st} z_t, \end{aligned} \tag{57}$$

where M_{ij} are the elements of $\mathbf{M}$ and $\overset{d}{=}$ denotes equality in distribution. If we require that the random variables z_t are independent, it follows from Eq. (57) that the equality in Eq. (57) if fulfilled only if the elements z_t follow a stable distribution. The key property of random variables with a stable distribution is that the weighted sum of independent realizations of the variable have the same distribution (up to location and scale) as the variable itself [129, 104]. Clearly, this property is necessary to satisfy the relationship above. Stable distributions are characterized by four parameters: stability (α), skewness, scale and location. We note that the stable distribution with parameter $\alpha = 2$, which is the normal distribution, is the only stable distribution with a finite variance. Hence, it follows that in order to satisfy both Eq. (57) and $\mathbf{M}\,\mathrm{E}\left[\mathbf{z}\mathbf{z}^\top\right]\mathbf{M}^\top = \boldsymbol{\Sigma}$ (analogous to Eq. (54)) the elements z_p must be normally distributed. Consequently the increments of the processes $\mathbf{y}$ and $\overline{\mathbf{y}}$ must also be Gaussian.

5.6.4 *Normalizing Transformations*

Given the normality requirements presented above, in the following we consider a class of transformations which modify the increments of certain non-normal processes to yield a normal distribution. So-called power transformations are a widely used analysis technique. Before delving into the specifics of the transform we outline our proposed methodology.

Consider a function g which transforms the elements x_t of a random vector $\mathbf{x}$ such that the elements $g(x_t) = v_t$ of the transformed process $\mathbf{v}$ are normally distributed. Further assume that g is invertible, i.e., $g^{-1}(g(x_t)) = x_t$. Hence, instead of sampling the traffic process $\mathbf{x}$ directly we apply the sampling approaches outlined in Section 5.5.3 to the transformed process $\mathbf{v}$. Specifically, we use $\mathbf{v}$ to estimate the elements $\Sigma_{v\,ij}$ of the corresponding covariance matrix $\boldsymbol{\Sigma}_\mathbf{v}$ and subsequently calculate the Cholesky decomposition $\mathbf{L}_\mathbf{v}\mathbf{L}_\mathbf{v}^\top = \boldsymbol{\Sigma}_\mathbf{v}$.

As the increments of the transformed process $\mathbf{v} = \mathbf{L}_\mathbf{v}\mathbf{z}$ are normally distributed it follows that the increments of the vector $\mathbf{z}$ are independent

and normally distributed with zero mean and unit variance. Therefore we can draw an arbitrary number of realizations $\widetilde{\mathbf{z}}$ to generate independent sample paths $\widetilde{\mathbf{v}}$ of the correlated, Gaussian increment process $\mathbf{v}$. Finally, we apply the inverse transform to all sample path elements to obtain

$$g^{-1}(\mathbf{v}) = \mathbf{x}. \tag{58}$$

Note, that while the autocovariance matrix $\mathbf{\Sigma_v}$ of the transformed process $\mathbf{v}$ and the autocovariance matrix $\mathbf{\Sigma_x}$ of the original vector $\mathbf{x}$ are structurally different[9] the original covariance $\mathbf{\Sigma_x}$ is restored through the reverse transform of the sample path elements $x_t = g^{-1}(v_t)$. Specifically, we may write the elements of $\mathbf{\Sigma_x}$ as

$$\begin{aligned}\Sigma_{x_{ij}} &= \mathrm{E}\left[g^{-1}(v_i)g^{-1}(v_j)\right] - \mathrm{E}\left[g^{-1}(v_i)\right]\mathrm{E}\left[g^{-1}(v_j)\right] \\ &= \mathrm{E}\left[x_i x_j\right] - \mathrm{E}[x_i]\,\mathrm{E}\left[x_j\right].\end{aligned}$$

BOX-COX TRANSFORM In the sequel we use a well known normalization technique. The so-called Box-Cox transform [14], defined as

$$g(x_t, \lambda) = \begin{cases} \frac{x_t^\lambda - 1}{\lambda} & \text{for} \quad \lambda \neq 0 \\ \ln(x_t) & \text{for} \quad \lambda = 0, \end{cases} \tag{59}$$

aims to normalize the increment distribution of $\mathbf{x}$ using some parameter λ. The parameter λ is estimated from the process data using a maximum likelihood approach. It can be shown using L'Hôpital's rule that $\lim_{\lambda\to 0} \frac{x_t^\lambda - 1}{\lambda} = \ln(x_t)$. In Fig. 35 we depict the increment distributions for an exemplary set of processes $\mathbf{x}_\lambda$, each of which transforms to an identically distributed Gaussian process $g(\mathbf{x}_{\langle\lambda\rangle}, \lambda) = \mathbf{v}$ with $v_t \sim \mathcal{N}(1, 0.6^2)$ for a specific value of λ. Note, that the Box-Cox transform of the normally distributed process $\mathbf{x}_{\langle 1\rangle}$, yields another normally distributed process. Further, $\mathbf{x}_{\langle 0\rangle}$ is a log-normal process. In the following we will show empirically that such distributions can be found in Internet traffic traces.

In our framework we estimate a suitable parameter λ from the collected observations and apply the transform Eq. (59) to each collected sample. Our experiments indicate that the Box-Cox transform is applicable to a wide range of real-world traffic sources. We evaluate the increment distribution of several traffic flows from publicly available Internet backbone traces. For each trace we aggregate the traffic over 10 ms intervals. The probability density function estimates of the flows are depicted in Fig. 36. In addition, the figures contain Quantile-Quantile plots (QQ-plots) which compare the flow increment distribution of each trace to a normal distribution - a line in the QQ plot indicates that the considered distribution is normally distributed. In Fig. 36a we evaluate the total

9 In Section A.4 of the Appendix, we show that for $g(x_t) = \ln(x_t)$ it is possible to derive an analytic relationship between the covariances $\mathbf{\Sigma_v}$ and $\mathbf{\Sigma_x}$. However, such a derivation is not feasible in the general case.

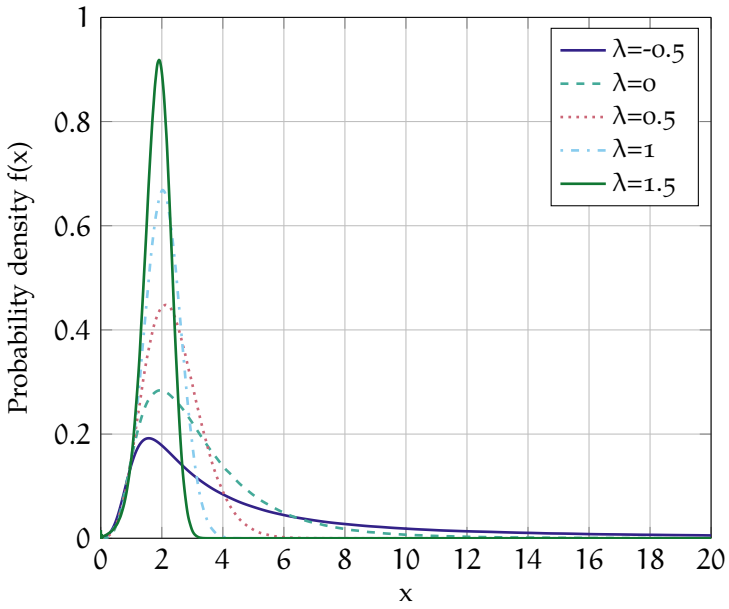


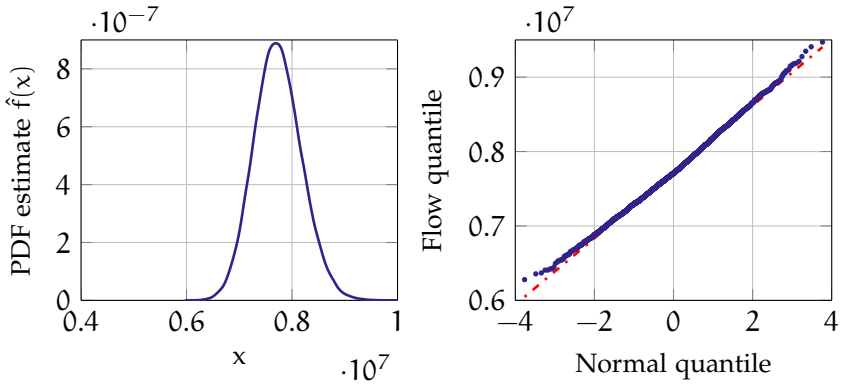
Figure 35: Exemplary probability density functions for increment processes which can be transformed to identically distributed Gaussian processes using the Box-Cox transform.

traffic of a CAIDA trace [131] with a mean rate of 6.1 Gbps. The QQ-plot indicates that the distribution of the increments is close to normal. Next, we select a dominant sub-flow from the same trace which has a mean rate of 307 Mbps. Clearly the distribution of the flow is skewed and the QQ-plot indicates a significant deviation from normality. Finally we inspect the total traffic from a trace from MAWI [98] with a mean rate of 264 Mbps. Again we find that the increments are non-Gaussian.

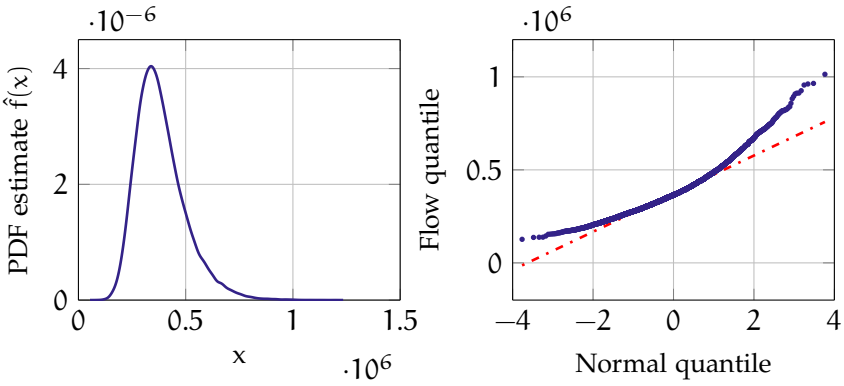
Next, we apply the Box-Cox transform to each of the traces. Using the maximum likelihood we estimate the parameters $\widetilde{\lambda}_1 = 0.12$, $\widetilde{\lambda}_2 = -0.08$, and $\widetilde{\lambda}_3 = 0.27$ corresponding to the CAIDA, CAIDA sub-flow and MAWI traces, respectively. The corresponding QQ-plots are shown in Fig. 37. Clearly the QQ-plots indicate that the distributions of the transformed increments closely resemble a normal distribution for all considered cases.

Finally, we compare the sample autocovariance $\widetilde{\boldsymbol{\Sigma}}_{\mathbf{x}_{C2}}$ of the CAIDA Internet flow (which we denote $\mathbf{x}_{C2}$) to the autocovariances $\widetilde{\boldsymbol{\Sigma}}_{\mathbf{y}_{C2}}$ of independent sample paths $\mathbf{y}_{C2}$ generated using the approach outlined above, as well as the corresponding increment distributions. Specifically, we generate 1×10^5 independent sample paths $\mathbf{y}_{C2}$ using $\mathbf{y}_{C2} = g^{-1}(\mathbf{Lz}, \widetilde{\lambda}_2)$ where $\mathbf{z}$ is a random vector with elements drawn from a standard normal distribution and $\widetilde{\boldsymbol{\Sigma}}_{\mathbf{v}_{C2}} = \mathbf{LL}^\top$ is the sample covariance matrix of the transformed process $\mathbf{v}_{C2} = g(\mathbf{x}_{C2}, \widetilde{\lambda}_2)$. Figure 38 depicts the autocovariance structure[10] of the CAIDA trace as well as the mean of the autocovariances of the generated independent sample paths (the

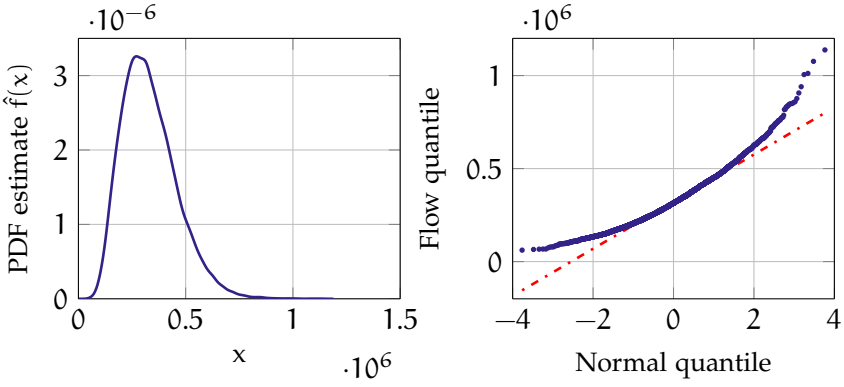
10 For clarity, we plot the autocovariance functions $\widetilde{c}_{(\cdot)}(\tau)$ associated with the autocovariance matrices $\widetilde{\boldsymbol{\Sigma}}_{(\cdot)}$, with elements $\widetilde{\Sigma}_{(\cdot)_{ij}} = \widetilde{c}_{(\cdot)}(i-j)$.



(a) CAIDA Trace

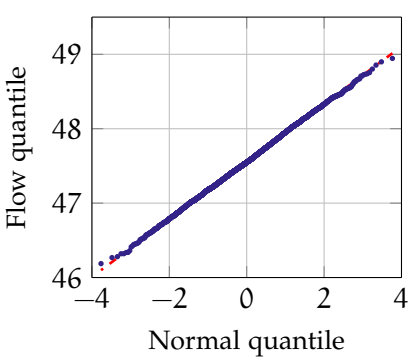


(b) CAIDA Trace (sub-flow)

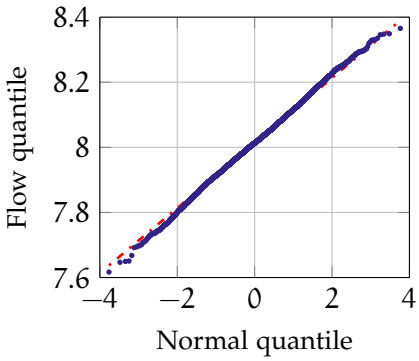


(c) MAWI Trace

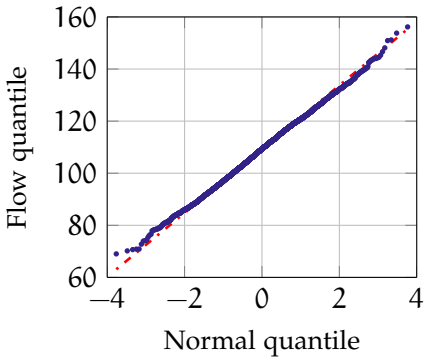
Figure 36: PDFs and corresponding QQ-plots of the increment distribution of Internet backbone traces.



(a) CAIDA Trace ($\lambda_1 = 0.12$)



(b) CAIDA Trace (sub-flow) ($\lambda_2 = -0.08$)



(c) MAWI Trace ($\lambda_3 = 0.27$)

Figure 37: QQ-plots of the transformed Internet traces indicating a Gaussian distribution of the transformed increments.

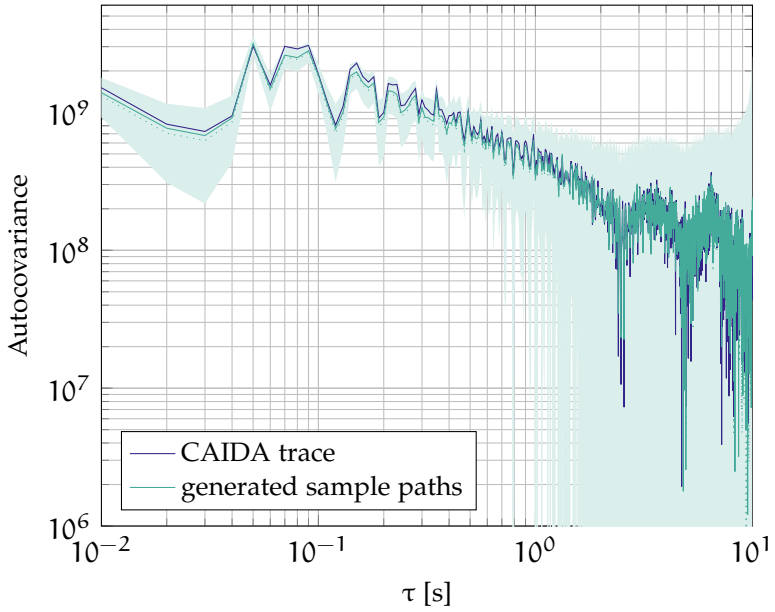


Figure 38: Autocovariance of a CAIDA Internet flow and mean autocovariance of 1×10^5 generated independent sample paths. Dotted line indicates the median. Shaded area indicates the interquartile range.

shaded area represents the 25-75 percentile range, the dotted line indicates the median). We observe that the mean of the sample path autocovariances closely follows the covariance structure of the original trace.

In addition, we evaluate the empirical CDFs G_k of the cumulative increments $\overline{x}_{C2_k}$ of the CAIDA flow trace aggregated over the intervals of $k \in \{1, 10, 100, 1000\}$ time slots with duration $\delta = 10\,\text{ms}$. We compare G_k to the empirical CDFs of the cumulative sample path increments $\overline{y}_{C2_k}$. The results are depicted in Fig. 39 where the CDFs G_k are represented using solid lines and dashed lines represent the CDFs of the cumulative increments of all generated sample paths for the considered integration intervals k. Evidently, the distributions match very well.

To further verify the accuracy of the covariance estimate obtained using a normalizing transform, we repeat the experiment described above for a traffic process with known statistical properties. Specifically, we compare the sample autocovariance $\widetilde{\boldsymbol{\Sigma}}_{\mathbf{x}_{LRD}}$ of a synthetic LRD trace $\mathbf{x}_{LRD}$ with non-Gaussian increments to the autocovariance of the corresponding sample paths $\mathbf{y}_{LRD}$ generated using the Cholesky decomposition. To obtain $\mathbf{x}_{LRD}$ we generated a trace $\mathbf{x}^*$ with an autocovariance structure characterized by a Hurst parameter of $H = 0.8$. We transformed the elements x_t^* of this trace using Eq. (59) with $\widetilde{\lambda}_2$ to get $x_{LRD_t} = (\widetilde{\lambda}_2 x_t^* + 1)^{1/\widetilde{\lambda}_2}$. Furthermore, the variance of $\mathbf{x}_{LRD}$ is configured to match the variance of the CAIDA flow $\mathbf{x}_{C2}$. Finally, we repeated the experiment outlined above to obtain the autocovariances depicted in Fig. 40. Clearly, the mean of the autocovariances of the generated sample paths matches the autocovariance of $\mathbf{x}_{LRD}$ exactly.

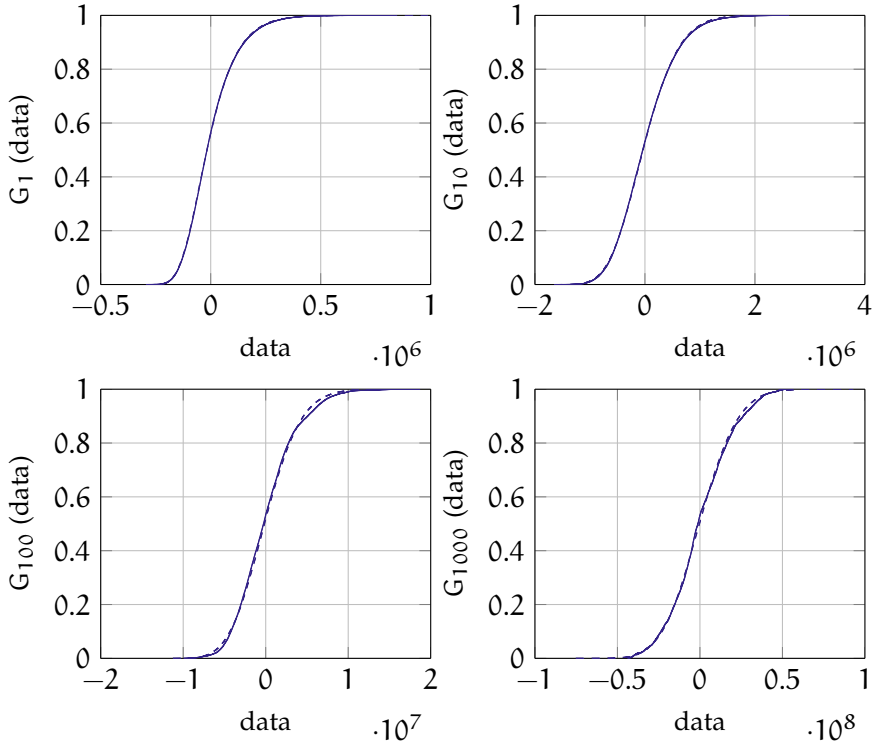


Figure 39: Empirical CDFs of the observed traffic process (solid lines) and the generated sample paths (dashed lines) over different aggregation intervals.

5.6.5 *Simulation Results*

In this section we use independent sample paths generated using the methods above to evaluate the queue length distribution for a CAIDA and MAWI flow assuming that a fixed capacity C is allocated to the corresponding flows. For each experiment we generated $1e6$ sample paths and evaluated the buffer overflow probability using a Monte Carlo approach with Eq. (7). The mean rate of the evaluated CAIDA flow is 468 Mbps. Hence, we simulated the queue length assuming that the flow is allocated the following capacities $C = \{550, 600, 650, 700, 750, 1000\}$ Mbps. In addition, we repeat the experiments using an autocovariance matrix obtained by randomly sampling the CAIDA flow with an probing intensity $\mu_A = 0.25$.

The simulation results for the CAIDA trace are depicted in Fig. 41. Clearly, the buffer overflow probability decreases as more capacity is assigned to the flow. The simulations indicate that assuming that the flow is allocated a capacity of 650 Mbps a queue length of 1 MB will be exceeded with a probability of less than $1e-4$. Moreover, the results obtained from random sampling are very close to the non-sampled simulations.

Next, we repeat the experiment for a MAWI flow, which has a mean rate of 264 Mbps, evaluating the capacities $C = \{300, 400, 500, 600, 700,$

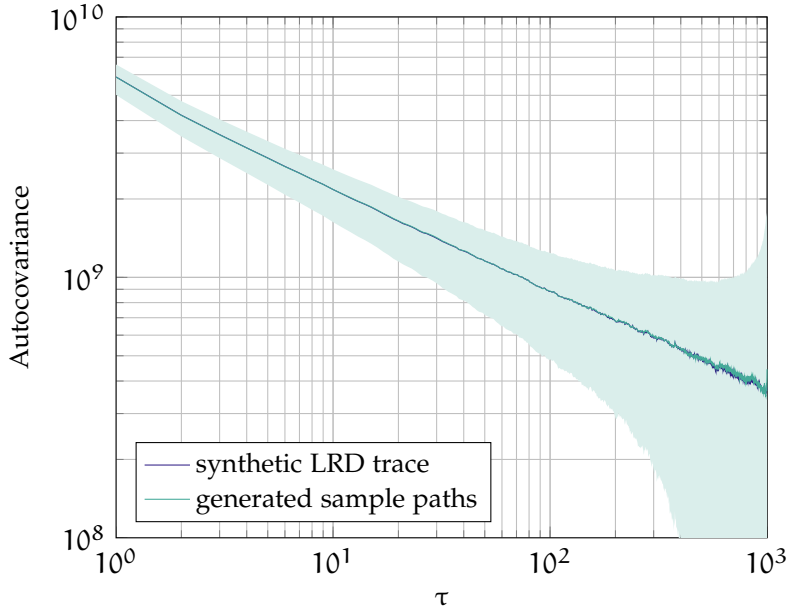


Figure 40: Autocovariance of synthetic LRD traffic flow and mean autocovariance of 1×10^5 generated independent sample paths. Dotted line indicates the median. Shaded area indicates the interquartile range.

1000} Mbps. The results are depicted in Fig. 42. Again, we observe that the results obtained from random sampling are close to the non-sampled simulations. Note, that even though the mean rate of this flow is significantly smaller than the previous scenario, a capacity of over 500 Mbps is required to obtain a similar overflow probability for a 1 MB queue length.

5.7 CENTRALIZED MONITORING OF DISTRIBUTED RESOURCES

The sampling approach outlined in the previous section enables the estimation of the covariance matrix of arbitrary flows using random packet sampling or by querying the associated byte counters on the forwarding device at random intervals. In this section we discuss several aspects of random sampling in centralized monitoring infrastructures. First, we consider the implications of monitoring multiple flows on a single device. Subsequently, we present a controller strategy for distributing the flow queries across multiple devices while maintaining a prescribed probing intensity at specific nodes.

MONITORING MULTIPLE FLOWS We consider the fact, that aggregate network traffic typically contains a small number of so-called elephant flows which contribute the largest share the overall traffic [155]. These dominating flows are particularly interesting candidates for monitoring. Moreover an operator may consider the aggregate traffic associated with a specific tenant tunnel as an individual flow.

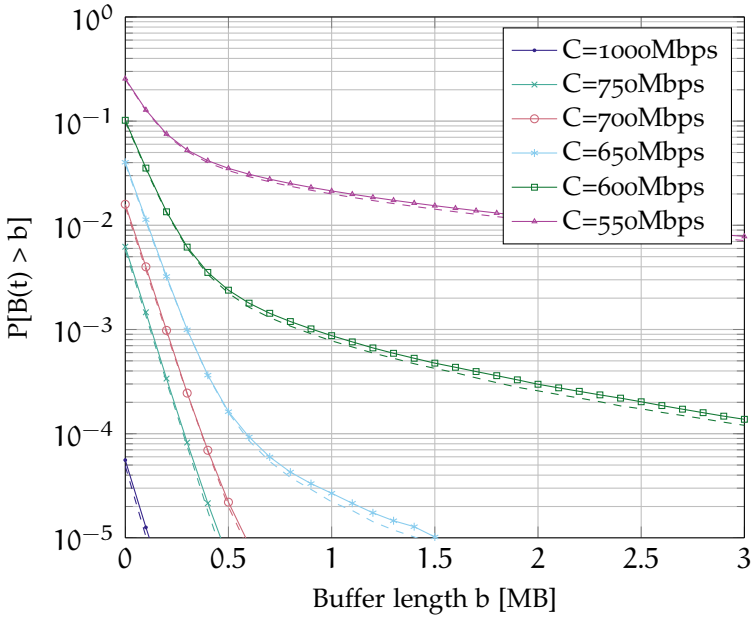


Figure 41: Queue simulations for different allocated capacities (CAIDA). 10×10^6 sample paths. Dashed lines indicate estimates obtained with random sampling.

In order to evaluate the impact of a subset of flows on a given interface, e.g., the removal of a dominant flow or a the migration of virtual link from the switch, the SDN controller should consider the total traffic traversing the examined link.

We model the total traffic traversing a switch interface as a sum of statistically independent flows $\bar{x}_i$, i.e., $\bar{x}_{IF} = \sum_{i \in S} \bar{x}_i$, where the subscript IF denotes the examined interface and S denotes the set of all flows traversing that interface. From the independence of the flows, it follows that the covariance matrix of the total traffic $\mathbf{\Sigma}_{IF}$ is given by the sum of the covariance matrices $\mathbf{\Sigma}_i$ of the individual flows $i \in S$. Therefore, given a set O of *observed* flows, the covariance matrix of the remaining traffic is obtained as $\mathbf{\Sigma}_{IF} - \mathbf{\Sigma}_O$, where $\mathbf{\Sigma}_O$ is the sum of the covariance matrices $\mathbf{\Sigma}_i$ of flows $i \in O$, i.e., the covariance matrix of the set of observed flows. Hence, in order to enable an evaluation of the impact of the monitored flows on the remaining interface traffic, we let the controller query the switch *port* counters in addition to the *flow* counters.

FLOW MONITORING ACROSS MULTIPLE NODES. In order to maintain an accurate view of the substrate state an SDN controller needs to monitor various counters across connected forwarding devices. In the following, we describe a strategy for viable multi-counter monitoring. We consider a controller that uses a counter sampling strategy based on geometrically distributed inter query times.

Consider a controller that generates query messages with an allocated maximum rate r_c which is configured by the network operator. Consequently, $\delta_C = 1/r_c$ denotes the minimum time between two subsequent

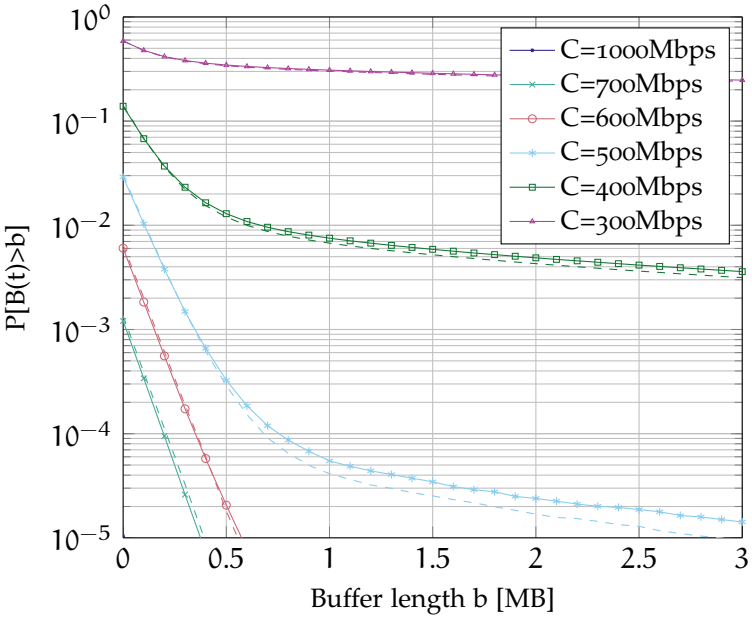


Figure 42: Queue simulations for different allocated capacities (MAWI trace). 10×10^6 sample paths. Dashed lines indicate estimates obtained with random sampling.

query messages issued by the controller (to any device). For a specific switch S let δ_S denote the minimum time interval between two subsequent query messages, which corresponds to a maximum query rate of $r_s = 1/\delta_S$ at that switch. Note that δ_S corresponds to the discrete sample time "slot" defined in the previous sections. Typically, the rate r_S at which the switch control logic can process statistic queries is significantly lower than the rate at which monitoring queries can be generated at the controller which is hosted on high performance server hardware, hence, we assume $\delta_S \gg \delta_C$.

In the following we outline a multi-query strategy using the example depicted in Fig. 43. We assume that query timescales δ_S for switches in the substrate network vary and are adjusted by the controller. Similarly, each switch may be probed with a different query intensity p_S. For ease of exposition, we ignore packet overheads and assume that each flow query is contained within a separate control message. Consider the query times of three flows f_R, f_G, f_B depicted in Fig. 43. The controller is configured to query each of the flows every $\delta_{R,G,B} = 4\delta_C$ time slots, with sampling intensities $p_R = 1/5$, $p_G = 1/10$, $p_B = 2/5$, respectively. The controller allocates a fixed number of tokens every time slot (10 tokens in this example) and assigns labels to the tokens such that the fraction of tokens for one flow corresponds to its intended sampling intensity. If the sum of the sampling intensities is smaller than one, then some tokens remain unlabeled. At every time slot δ_C the controller randomly selects one token, and generates a query message for the corresponding flow counter. The controller remains silent if an unlabeled token is selected. As a result, the token selection can be interpreted as

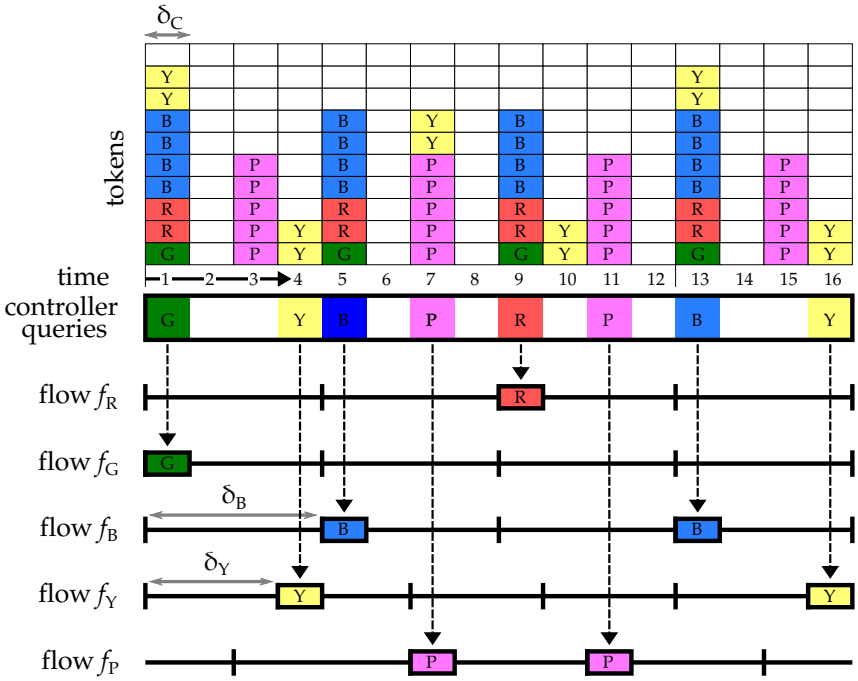


Figure 43: Controller query strategy with geometric inter sample times.

a Bernoulli trial such that the inter-query intervals to the corresponding switch counter are geometrically distributed. Therefore, a reconstruction of the sampling distortion is possible using the framework outlined in the previous section.

Next, the controller adds two new flows to the controller's monitoring list: f_Y with $\delta_Y = 3\delta_C$ and $p_Y = 1/5$ and f_P with $\delta_P = 4\delta_C$ and $p_P = 1/2$. Since not enough free tokens are available at time slot 1 the controller periodically allocates tokens for flow x_p starting at the empty time slot 3[11]. For the considered scenario the random selection scheme is repeated in a round robin fashion every $12\delta_c$ time slots. The controller may add further flows for monitoring until all available tokens are labeled.

5.8 CONCLUSIONS

The work presented in this chapter was motivated by the need for a expressive characterization of network flows which may be used to quantify the requirements in multi-tenant virtual environments. Although the proposed mechanisms are applicable to any scenario where a flow level monitoring of traffic autocovariances is required, in our view techniques which enable the extraction of QoS metric are particularly relevant for SDN environments where applications operate using a global view of the network resources. In this context, the integration of the proposed mechanisms into SDN controller frameworks is consistent with the SDN

11 The allocation of the starting time slot may be randomized.

goal of enabling the reuse of low-level network functions through suitable abstractions.

In this chapter we focused on mechanisms for extracting the autocovariance of individual flows from network measurements. We showed that the autocovariance metric may be used to exploit a number of theoretical results which enable the derivation of QoS metrics such as stochastic delay and backlog bounds. Specifically, we used the empirically obtained covariance structure to generate independent traffic sample paths which replicate the statistical properties of the observed flow. Subsequently, the sample paths are to carry out a Monte Carlo simulation in order to evaluate the queue length distributions which result from forwarding the traffic over specific switch interface configurations.

To minimize the monitoring traffic load and the processing required at the switches without sacrificing resolution we use random sampling. Two approaches were analyzed: the use of randomly sampled packets, e.g., generated by sFlow, and the use of randomly spaced counter queries, e.g., from flow counters in OF switches. We provided an analytical evaluation of random sampling, showing that the effects of the sampling process may be reversed to obtain asymptotically unbiased estimates of the flow autocovariance. In addition, we quantified the effects of finite measurement durations on the quality of the estimate. We highlighted the relationship between the process autocovariance and the aggregated variance metric, demonstrating that a differencing approach may be used to obtain an unbiased estimate of the Hurst parameter, which does not depend on the duration of the observation. Our results highlight the relationship between the packet and counter based autocovariance estimators, but also indicate that the two problems are not equivalent for the case of randomly drawn observations.

Next, we evaluated approaches for synthesizing independent sample paths, which match the increment distribution of the process in addition to its covariance structure. For processes with non-Gaussian increments we used Internet traffic traces to show that a normalizing transformation of the process yields feasible results. Equipped with these results, we used Internet traffic traces to evaluate the queue violation probability given different capacity assignments. Our simulations confirm that random sampling does not have a notable negative impact on the considered QoS metric.

Finally, we demonstrated how the proposed sampling strategy may be integrated into a centralized controller framework enabling the monitoring of a large number of flows across a pool of substrate switches.

6 CONCLUSIONS AND FUTURE WORK

The goal of this thesis was to further the understanding of the abstractions necessary to implement a versatile virtualization platform for SDN and to develop methodologies which augment the network view with useful QoS metrics. In the first part of the thesis we proposed a scalable architecture which enables a full virtualization of the SDN infrastructure, granting tenants unrestricted access to isolated virtual SDN topologies. To achieve this we identified mechanisms for mapping different physical resources to specific virtual contexts. Using the notion of virtual context identifiers we decoupled the resource encoding from the underlying technology and evaluated the feasibility and limitations of the flowspace segmentation in SDN forwarding devices. Context identifiers are concealed from the tenant by the hypervisor to create an illusion of a dedicated SDN substrate.

We demonstrated that the instantiation of arbitrary virtual network topologies, i.e., infrastructure virtualization, requires a 1:1 mapping between the virtual and physical resources, in order to enable the deployment of resilience mechanisms in the vSDN. Moreover, the vSDN resources must be allocated statically by the InP to avoid conflicts with optimization approaches carried out in the virtual domain.

Conversely, we derived a simplified topology abstraction which fully offloads the provision of resilience and load balancing mechanisms to the substrate operator. As a consequence virtual topologies may be specified as a connectivity service between multiple tenant PoPs, with QoS demands at each tenant attachment point. We evaluated the requirements for embedding such a connectivity service, finding that the capacity allocation costs are minimized for rooted tree topologies. An additional reduction of the allocation costs is only possible if the point-to-point traffic requirements between the tenant attachment points are known. However, such demands are inherently difficult to predict. Finally, we detailed a virtual router architecture as an example of a layer 3 connectivity service.

The findings of the first part of our work highlight the importance of a comprehensive view of the network state. Due to the characteristics of long-memory Internet traffic over-provisioning is inevitable for maintaining reliable performance in computer networks. Hence, in order to dimension the physical resources allocated to virtual entities the SDN hypervisor requires techniques which provide a tight characterization of the carried network traffic. Moreover, the derived QoS metrics may be used by SDN applications to optimize their performance. Thus, in the second part of this thesis we focused on the extraction of traffic characteristics and the derivation of associated QoS parameters. We evaluated

approaches for extracting flow-level traffic correlations from network observations obtained from packet samples or switch counter queries. Random sampling was used to reduce the monitoring load while maintaining a high resolution of the autocovariance structure. We analyzed the impact of the sampling distribution on the considered estimates and outlined mechanisms for reversing distortions. Moreover, we carried out an analytical evaluation of the effects of finite sampling durations on the estimators. We showed that the autocovariance estimators are asymptotically unbiased. For finite measurement durations the bias corresponds to the variance of the sample mean. For LRD processes this variance decays slowly and is characterized by the Hurst parameter. We derived an approach for estimating the Hurst parameter, proving that the resulting estimate is unbiased and regardless of the length of the measurement.

The autocovariance estimates were used to perform Monte Carlo simulations with the goal of evaluating the queue length distribution for the observed flow for arbitrary capacity assignments. To this end, we outlined approaches for synthesizing independent sample paths which exhibit the same correlation structure as the observed traffic flow and also match its increment distribution under certain conditions. We demonstrated that the use of random sampling does not have a notable effect on the obtained estimate of the queue length distribution.

The approaches presented in this work are in line with the concepts defined by the SDN paradigm. The proposed connectivity service abstraction hides the complexity of the underlying topology design from the tenant, requiring only the specification of QoS demands and the configuration of the control logic. The integration of the proposed measurement methodologies into a controller framework which provides a global view of the network state enables SDN applications to benefit from the derived QoS metrics. We expect that such derived traffic metrics will become increasingly important as SDN technology matures and SDN application begin to make use of the provided abstractions, minimizing the need for human interaction in deploying network services.

The results of this thesis lead to some open research questions and possible future research directions. The proposed scheme for encoding packet and table identifiers relies on translator lookup tables. While it is possible to implement such lookups using current versions of the OF specification, extensions to the specifications and switch interface may facilitate the instantiation of virtual resources. Moreover, the monitoring traffic load may be further reduced by partially offloading the parsing of counter statistics to the switch control logic and transmitting flow covariance estimates to the controller for aggregation.

In this thesis we assumed a model in which virtual links are allocated a fixed capacity on all interfaces along the corresponding substrate path. In future work, the monitoring framework may be extended to accommodate cases where switch interfaces are modeled as a stochastic service. Finally, the derived QoS parameters may be analyzed using recent results on end-to-end performance evaluation.

Part II

APPENDIX

A

PROOFS AND DERIVATIONS

A.1 COVARIANCE OF THE SAMPLE MEANS

To derive an expression for the bias of the sample autocovariance in Eq. (23) we evaluate the expected value $\mathrm{E}[\tilde{\mu}_{X_0}\tilde{\mu}_{X_\tau}]$ with $\tilde{\mu}_{X_0} = \frac{1}{(T-\tau)}\sum_{t=1}^{T-\tau} x(t)$ and $\tilde{\mu}_{X_\tau} = \frac{1}{(T-\tau)}\sum_{t=\tau+1}^{T} x(t)$ to obtain

$$\begin{aligned}
\mathrm{E}[\tilde{\mu}_{X_0}\tilde{\mu}_{X_\tau}] &= \frac{1}{(T-\tau)^2}\,\mathrm{E}\left[\sum_{t=1}^{T-\tau} x(t)\sum_{t=\tau+1}^{T} x(t)\right] \\
&= \frac{1}{(T-\tau)^2}\,\mathrm{E}[(x_1 + x_2 + \ldots + x_{T-\tau})(x_{\tau+1} + x_{\tau+2} + \ldots + x_T)] \\
&= \frac{1}{(T-\tau)^2}\sum_{i=1}^{T-\tau}\sum_{j=1}^{T-\tau}\mathrm{E}\big[x_i x_{\tau+j}\big].
\end{aligned}$$

Using the relationship $\mathrm{E}\big[x_i x_j\big] = c_x(i-j) + \mathrm{E}[x_i]\,\mathrm{E}\big[x_j\big]$ with $\mathrm{E}[x_i] = \mathrm{E}\big[x_j\big] = \mu_x$ we obtain

$$\begin{aligned}
\mathrm{E}[\tilde{\mu}_{X_0}\tilde{\mu}_{X_\tau}] &= \frac{1}{(T-\tau)^2}\sum_{i=1}^{T-\tau}\sum_{j=1}^{T-\tau}[c_x(i-j+\tau) + \mu_x^2] \\
&= \mu_x^2 + \frac{1}{(T-\tau)^2}\sum_{i=1}^{T-\tau}\sum_{j=1}^{T-\tau} c_x(i-j+\tau).
\end{aligned}$$

Substituting the equation above into Eq. (23) yields

$$\mathrm{E}[\tilde{c}_X(\tau)] = c_X(\tau) - \frac{1}{(T-\tau)^2}\sum_{i=1}^{T-\tau}\sum_{j=1}^{T-\tau} c_x(i-j+\tau)$$

A.2 AGGREGATED VARIANCE OF AN FGN PROCESS

We substitute the autocovariance of an fGn process $c_X(\tau) = \frac{\sigma^2}{2}[(\tau-1)^{2H} - 2\tau^{2H} + (\tau+1)^{2H}]$ given in Eq. (3) into Eq. (30), which yields

$$\begin{aligned}
\sigma^2_{X^{(M)}} =& \frac{\sigma^2}{M} + \frac{2}{M^2}\sum_{\tau=1}^{M-1}(M-\tau)c_x(\tau) \\
=& \frac{\sigma^2}{M} + \frac{\sigma^2}{M^2}\sum_{\tau=1}^{M-1}(M-\tau)[(\tau-1)^{2H} - 2\tau^{2H} + (\tau+1)^{2H}] \\
=& \frac{\sigma^2}{M^2}\Bigg[M + \sum_{\tau=1}^{M-1}(M-\tau)(\tau-1)^{2H} - 2\sum_{\tau=1}^{M-1}(M-\tau)\tau^{2H} \\
&+ \sum_{\tau=1}^{M-1}(M-\tau)(\tau+1)^{2H}\Bigg]
\end{aligned}$$

Substituting $k = \tau - 1$, $l = \tau + 1$ and rearranging the summation bounds appropriately, reveals

$$\begin{aligned}
\sigma^2_{X^{(M)}} =& \frac{\sigma^2}{M^2}\Bigg[M + \sum_{k=0}^{M-2}(M-k-1)k^{2H} - 2\sum_{\tau=1}^{M-1}(M-\tau)\tau^{2H} \\
&+ \sum_{l=2}^{M}(M-l+1)l^{2H}\Bigg] \\
=& \frac{\sigma^2}{M^2}\Bigg[\left(\sum_{k=1}^{M-2}(M-k)k^{2H} - \sum_{k=1}^{M-2}k^{2H}\right) - 2\sum_{\tau=1}^{M-1}(M-\tau)\tau^{2H} \\
&+ \left(\sum_{l=1}^{M-1}(M-l)l^{2H} + \sum_{l=1}^{M}l^{2H}\right)\Bigg] \\
=& \frac{\sigma^2}{M^2}\Bigg[\left(-(M-1)^{2H} + \sum_{k=1}^{M-1}(M-k)k^{2H}\right) - 2\sum_{\tau=1}^{M-1}(M-\tau)\tau^{2H} \\
&+ \sum_{l=1}^{M-1}(M-l)l^{2H} + (M-1)^{2H} + M^{2H}\Bigg] \\
=& \frac{\sigma^2}{M^2}\left[-(M-1)^{2H} + (M-1)^{2H} + M^{2H}\right] \\
=& \sigma^2 M^{2H-2}
\end{aligned}$$

A.3 AGGREGATED VARIANCE OF A SAMPLED PROCESS

Proof of Lemma 5.3. The aggregated version of the process $W(t)$ on the aggregation level M is defined for $k \in \mathbb{N}$ as

$$W^{(M)}(k) = \frac{1}{M}\sum_{t=1+(k-1)M}^{kM} W(t),$$

where M is the block size that is used for averaging. The variance of $W^{(M)}$ is obtained using Eq. (30) as

$$\mathrm{Var}\left[W^{(M)}\right] = \frac{c_W(0)}{M} + \frac{2}{M^2}\sum_{\tau=1}^{M-1}(M-\tau)c_W(\tau). \tag{60}$$

The same expression can be formulated for $\mathrm{Var}\left[A^{(M)}\right]$ and $\mathrm{Var}\left[Y^{(M)}\right]$ by substituting $c_A(\tau)$ resp. $c_Y(\tau)$ for $c_W(\tau)$ in (60). To obtain a relationship between $\mathrm{Var}\left[W^{(M)}\right]$ and $\mathrm{Var}\left[A^{(M)}\right]$, $\mathrm{Var}\left[Y^{(M)}\right]$, $c_A(\tau)$ and $c_Y(\tau)$, we insert $c_W(\tau)$ from Lemma 5.1 into (60). We obtain

$$\begin{aligned}\mathrm{Var}\left[W^{(M)}\right] &= \frac{1}{M}c_Y(0)(c_A(0)+\mu_A^2) + \frac{1}{M}c_A(0)\mu_Y^2 \\ &+\frac{2}{M^2}\sum_{\tau=1}^{M-1}(M-\tau)(c_Y(\tau)(c_A(\tau)+\mu_A^2)+c_A(\tau)\mu_Y^2).\end{aligned}$$

After some reordering we arrive at

$$\begin{aligned}\mathrm{Var}\left[W^{(M)}\right] &= \mu_Y^2\frac{c_A(0)}{M} + \mu_Y^2\frac{2}{M^2}\sum_{\tau=1}^{M-1}(M-\tau)c_A(\tau) \\ &\quad +\mu_A^2\frac{c_Y(0)}{M} + \mu_A^2\frac{2}{M^2}\sum_{\tau=1}^{M-1}(M-\tau)c_Y(\tau) \\ &+\frac{1}{M}c_Y(0)c_A(0) + \frac{2}{M^2}\sum_{\tau=1}^{M-1}(M-\tau)c_Y(\tau)c_A(\tau)\end{aligned}$$

and by application of (60) we obtain

$$\begin{aligned}\mathrm{Var}\left[W^{(M)}\right] &= \mu_Y^2\,\mathrm{Var}\left[A^{(M)}\right] + \mu_A^2\,\mathrm{Var}\left[Y^{(M)}\right] \\ &+\frac{1}{M}c_Y(0)c_A(0) + \frac{2}{M^2}\sum_{\tau=1}^{M-1}(M-\tau)c_Y(\tau)c_A(\tau).\end{aligned}$$

Finally we substitute $c_Y(0) = \sigma_Y^2$ and $c_A(0) = \sigma_A^2$ into the equation above. □

A.4 COVARIANCE OF TRANSFORMED LOGNORMAL PROCESS

We consider a process $\mathbf{x}$ with a lognormal distribution, i.e., the transformed process $\mathbf{v} = \ln(\mathbf{x})$ has normally distributed increments ($\lambda = 0$). In the following we derive a relationship between the covariance matrix $\Sigma_{\mathbf{x}}$ the process of $\mathbf{x}$ and the covariance $\Sigma_{\mathbf{v}}$ of the transformed process $\mathbf{v}$.

Consider the definition of the elements of the covariance matrix:

$$\Sigma_{ij} = \mathrm{Cov}\left[x_i, x_j\right] = \mathrm{E}\left[x_i x_j\right] - \mathrm{E}[x_i]\,\mathrm{E}\left[x_j\right] \tag{61}$$

where x_t are the vector elements of $\mathbf{x}$ and the mean rate is $\mathrm{E}[x_t] = \mu_x$. We write

$$\mathrm{E}[x_t] = \mathrm{E}[e^{v_t}] = e^{\mu_v + \frac{1}{2}\sigma_v^2},$$

and let $w_t = v_i + v_j$

$$\begin{aligned}
\mathrm{E}\left[x_i x_j\right] &= \mathrm{E}[e^{v_i} e^{v_j}] = \mathrm{E}\left[e^{v_i + v_j}\right] \\
&= \mathrm{E}[e^{w_t}] = e^{\mu_w + \frac{1}{2}\sigma_w^2} \\
&= e^{2\mu_v + \frac{1}{2}(\sigma_v^2 + \sigma_v^2 + 2\,\mathrm{Cov}\left[v_i, v_j\right])}
\end{aligned}$$

where the variance $\sigma_w^2 = \sigma_v^2 + \sigma_v^2 + 2\,\mathrm{Cov}\left[v_i, v_j\right]$ and the mean $\mu_w = \mathrm{E}\left[v_i + v_j\right] = 2\mu_v$. Rearranging Eq. (61) we get

$$\begin{aligned}
\frac{\mathrm{Cov}\left[x_i, x_j\right]}{\mathrm{E}[x_i]\,\mathrm{E}\left[x_j\right]} + 1 &= \frac{\mathrm{E}\left[x_i x_j\right]}{\mathrm{E}[x_i]\,\mathrm{E}\left[x_j\right]} \\
&= \frac{e^{2\mu_v + \frac{1}{2}(\sigma_v^2 + \sigma_v^2 + 2\,\mathrm{Cov}\left[v_i, v_j\right])}}{e^{\mu_v + \frac{1}{2}\sigma_v^2} e^{\mu_v + \frac{1}{2}\sigma_v^2}} \\
&= \frac{e^{2\mu_v + \sigma_v^2 + \mathrm{Cov}\left[v_i, v_j\right]}}{e^{2\mu_v + \sigma_v^2}} = e^{\mathrm{Cov}\left[v_i, v_j\right]}
\end{aligned}$$

Thus finally we obtain

$$\begin{aligned}
\mathrm{Cov}\left[x_i, x_j\right] &= \mu_x^2 (e^{\mathrm{Cov}\left[v_i, v_j\right]} - 1) \\
\ln\left(\frac{\mathrm{Cov}\left[x_i, x_j\right]}{\mu_x^2} + 1\right) &= \mathrm{Cov}\left[v_i, v_j\right] \\
\ln\left(\frac{\Sigma_{\mathbf{x}ij}}{\mu_x^2} + 1\right) &= \Sigma_{\mathbf{v}ij}.
\end{aligned}$$

BIBLIOGRAPHY

[1] R. K. Ahuja, T. L. Magnati, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*, pages 320–324. Prentice Hall, 1993.

[2] A. Al-Shabibi, M. De Leenheer, M. Gerola, A. Koshibe, G. Parulkar, E. Salvadori, and B. Snow. Openvirtex: Make your virtual sdns programmable. In *Proc. of HotSDN*, pages 25–30, 2014.

[3] G. Appenzeller, I. Keslassy, and N. McKeown. Sizing router buffers. In *Proc. of ACM SIGCOMM*, pages 281–292, September 2004.

[4] D. Applegate and E. Cohen. Making intra-domain routing robust to changing and uncertain traffic demands: Understanding fundamental tradeoffs. In *Proc. of SIGCOMM*, pages 313–324, 2003.

[5] F. Baccelli, S. Machiraju, D. Veitch, and J. Bolot. The role of PASTA in network measurement. *IEEE/ACM Trans. Networking*, 17(4):1340–1353, 2009.

[6] G. Baier, E. Köhler, and M. Skutella. The k-splittable flow problem. *Algorithmica*, 42(3-4):231–248, 2005. ISSN 0178-4617. http://dx.doi.org/10.1007/s00453-005-1167-9.

[7] T. Ball, N. Bjørner, A. Gember, S. Itzhaky, A. Karbyshev, M. Sagiv, M. Schapira, and A. Valadarsky. Vericon: Towards verifying controller programs in software-defined networks. In *Proc. of ACM SIGPLAN PLDI*, pages 282–293, 2014.

[8] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proc. of ACM Symposium on Operating Systems Principles*, pages 164–177, 2003.

[9] Y. Bejerano and P. V. Koppol. Link-coloring based scheme for multicast and unicast protection. In *Proc. of IEEE HPSR*, pages 21–28, July 2013.

[10] J. Beran. *Statistics for Long-Memory Processes*. Chapman & Hall/CRC, October 1994.

[11] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Services. RFC 2475 (Informational), December 1998. URL `http://www.ietf.org/rfc/rfc2475.txt`. Updated by RFC 3260.

[12] G. Bolch, S. Greiner, H. de Meer, and K. Trivedi. *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*. WileyBlackwell, May 2006.

[13] J.-Y. L. Boudec and P. Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*, volume 2050 of *Lecture Notes in Computer Science*. Springer, 2001.

[14] G. E. Box and D. R. Cox. An analysis of transformations. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 211–252, 1964.

[15] Z. Bozakov. Virtual software routers: A performance and migration study. In *Proc. of DFN-Forum Kommunikationstechnologien*, pages 35–44, 2010.

[16] Z. Bozakov. An open router virtualization framework using a programmable forwarding plane. In *Proc. of ACM SIGCOMM (Poster Session)*, pages 439–440, 2010.

[17] Z. Bozakov. Architecture and algorithms for virtual routers as a service. In *Proc. of International Workshop on Quality of Service, (IWQoS)*, June 2011.

[18] Z. Bozakov and P. Papadimitriou. Autoslice: Automated and scalable slicing for software-defined networks. In *Proc. of ACM CoNEXT Student Workshop*, pages 3–4, 2012.

[19] Z. Bozakov and P. Papadimitriou. OpenVRoute: An open architecture for high-performance programmable virtual routers. In *Proc. of IEEE High Performance Switching and Routing (HPSR)*, pages 191–196, July 2013.

[20] Z. Bozakov and P. Papadimitriou. Towards a scalable software-defined network virtualization platform. In *Proc. of IEEE Network Operations and Management Symposium (NOMS)*, May 2014.

[21] Z. Bozakov and A. Rizk. Taming SDN controllers in heterogeneous hardware environments. In *Proc. of European Workshop on Software Defined Networks (EWSDN)*, pages 50–55, Oct 2013.

[22] R. Braden, D. Clark, and S. Shenker. Integrated Services in the Internet Architecture: an Overview. RFC 1633 (Informational), June 1994. URL `http://www.ietf.org/rfc/rfc1633.txt`.

[23] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification. RFC 2205 (Proposed Standard), September 1997. URL `http://www.ietf.org/rfc/rfc2205.txt`. Updated by RFCs 2750, 3936, 4495, 5946, 6437, 6780.

[24] M. Bredel, Z. Bozakov, and Y. Jiang. Analyzing router performance using network calculus with external measurements. In *Proc. of International Workshop on Quality of Service, (IWQoS)*, June 2010.

[25] L. Breslau, S. Jamin, and S. Shenker. Comments on the performance of measurement-based admission control algorithms. pages 1233–1242, 2000.

[26] L. Breslau, E. W. Knightly, S. Shenker, I. Stoica, and H. Zhang. Endpoint admission control: Architectural issues and performance. In *Proc. of SIGCOMM*, pages 57–69, 2000.

[27] A. Burchard, J. Liebeherr, and S. Patek. A min-plus calculus for end-to-end statistical service guarantees. *IEEE Trans. Inform. Theory*, 52(9):4105–4114, September 2006.

[28] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe. Design and implementation of a routing control platform. In *Proc. NSDI'05*, pages 15–28, 2005.

[29] Z. Cai, A. L. Cox, and T. S. E. Ng. Maestro: A system for scalable openflow control. Technical report, Rice University, 2011.

[30] M. Casado, T. Koponen, R. Ramanathan, and S. Shenker. Virtualizing the network forwarding plane. In *Proc. of PRESTO*, pages 8:1–8:6, 2010.

[31] C. Chang. *Performance Guarantees in Communication Networks*. Springer, 2000.

[32] N. Chowdhury, M. Rahman, and R. Boutaba. Virtual network embedding with coordinated node and link mapping. In *Proc. of INFOCOM*, pages 783 –791, April 2009.

[33] S. Chowdhury, M. Bari, R. Ahmed, and R. Boutaba. Payless: A low cost network monitoring framework for software defined networks. In *Proc. of IEEE NOMS*, pages 1–9, May 2014.

[34] F. Ciucu, A. Burchard, and J. Liebeherr. Scaling properties of statistical end-to-end bounds in the network calculus. *IEEE/ACM Trans. Networking*, 14(SI):2300–2312, June 2006.

[35] Controller Performance. Openflow controller performance comparisons. `www.openflow.org/wk/index.php/Controller_Performance_Comparisons`, 2012.

[36] G. Covington, G. Gibb, J. Lockwood, and N. Mckeown. A packet generator on the netfpga platform. In *Proc. of IEEE Symposium on Field Programmable Custom Computing Machines (FCCM '09)*, pages 235–238, April 2009.

[37] M. Crovella and A. Bestavros. Self-similarity in World Wide Web traffic: evidence and possible causes. *IEEE/ACM Trans. Networking*, 5(6):835–846, December 1997.

[38] R. De Schmidt, R. Sadre, and A. Pras. Gaussian traffic revisited. In *Proc. of IFIP Networking*, pages 1–9, May 2013.

[39] D. Dietrich, A. Rizk, and P. Papadimitriou. Multi-domain virtual network embedding with limited information disclosure. In *Proc. of IFIP Networking*, 2013.

[40] X. Dimitropoulos, P. Hurley, and A. Kind. Probabilistic lossy counting: An efficient algorithm for finding heavy hitters. *SIGCOMM Computer Communication Review*, 38(1):5–5, January 2008. ISSN 0146-4833.

[41] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella. Towards an elastic distributed sdn controller. In *Proc. of HotSDN*, pages 7–12, 2013.

[42] M. Dobrescu, N. Egi, K. Argyraki, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy. Routebricks: Exploiting parallelism to scale software routers. In *Proc. of ACM SIGOPS*, pages 15–28, 2009.

[43] A. Doria, J. H. Salim, R. Haas, H. Khosravi, W. Wang, L. Dong, R. Gopal, and J. Halpern. Forwarding and Control Element Separation (ForCES) Protocol Specification. RFC 5810 (Proposed Standard), March 2010. URL `http://www.ietf.org/rfc/rfc5810.txt`. Updated by RFCs 7121, 7391.

[44] R. Doriguzzi Corin, M. Gerola, R. Riggio, F. De Pellegrini, and E. Salvadori. Vertigo: Network virtualization and beyond. In *Proc. of EWSDN*, pages 24–29, Oct 2012.

[45] DPDK. DPDK: Data plane development kit. `http://dpdk.org/`.

[46] D. Drutskoy, E. Keller, and J. Rexford. Scalable network virtualization in software-defined networks. *Internet Computing, IEEE*, 17(2): 20–27, March 2013. ISSN 1089-7801.

[47] N. Duffield et. al. Resource management with hoses: point-to-cloud services for virtual private networks. *Networking, IEEE/ACM Transactions on*, 10(5):679 – 692, October 2002. ISSN 1063-6692.

[48] N. Egi, A. Greenhalgh, M. Handley, M. Hoerdt, F. Huici, and L. Mathy. Towards high performance virtual routers on commodity hardware. In *Proc. of ACM CoNEXT*, pages 20:1–20:12, 2008.

[49] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman. Network Configuration Protocol (NETCONF). RFC 6241 (Proposed Standard), June 2011. URL `http://www.ietf.org/rfc/rfc6241.txt`.

[50] D. Erickson. The beacon openflow controller. In *Proc. of HotSDN*, pages 13–18, 2013.

[51] A. Erramilli, O. Narayan, and W. Willinger. Experimental queueing analysis with long-range dependent packet traffic. *IEEE/ACM Trans. Networking*, 4(2):209–223, 1996.

[52] S. K. Fayazbakhsh, V. Sekar, M. Yu, and J. C. Mogul. Flowtags: Enforcing network-wide policies in the presence of dynamic middlebox actions. In *Proc. of HotSDN*, pages 19–24, 2013.

[53] A. Feldmann, A. C. Gilbert, P. Huang, and W. Willinger. Dynamics of IP traffic: A study of the role of variability and the impact of control. In *Proc. of ACM SIGCOMM*, pages 301–313, Aug. 1999.

[54] M. Fidler. A survey of deterministic and stochastic service curve models in the network calculus. *IEEE Communications Surveys and Tutorials*, 12(1), 2010.

[55] M. Fidler, Z. Bozakov, and D. Dietrich. Projekt G-Lab - phase 2, teilvorhaben: G-Lab_Virturama (leitungs- und routervirtualisierung): Schlussbericht. Technische Informationsbibliothek u. Universitätsbibliothek, 2011. URL `http://www.pt-it.pt-dlr.de/de/2037.php`. Berichtszeitraum: 01.09.2009 - 31.08.2011.

[56] B. Fortz and M. Thorup. Internet traffic engineering by optimizing ospf weights. In *Proc. of IEEE INFOCOM*, volume 2, pages 519–528 vol.2, 2000.

[57] B. Fortz and M. Thorup. Optimizing ospf/is-is weights in a changing world. *IEEE J.Sel. A. Commun.*, 20(4):756–767, September 2006. ISSN 0733-8716.

[58] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker. Frenetic: A network programming language. In *Proc. of ACM SIGPLAN ICFP*, pages 279–291, 2011.

[59] I. Ganichev, B. Dai, P. B. Godfrey, and S. Shenker. Yamr: Yet another multipath routing protocol. *SIGCOMM Computer Communication Review*, 40(5):13–19, October 2010. ISSN 0146-4833.

[60] A. Genz. Numerical computation of multivariate normal probabilities. *Journal of Computational and Graphical Statistics*, 1(2):pp. 141–149, 1992. ISSN 10618600.

[61] G. Giorgi and C. Narduzzi. Rate-interval curves - a tool for the analysis and monitoring of network traffic. *Perform. Eval.*, 65(6-7): 441–462, June 2008. ISSN 0166-5316.

[62] G. H. Golub and C. F. Van Loan. *Matrix Computations*, chapter 4, page 143. JHU Press, London, third edition edition, 1996. ISBN 978-0-801-85414-9.

[63] M. Grant and S. Boyd. Graph implementations for nonsmooth convex programs. In *Recent Advances in Learning and Control*, Lecture Notes in Control and Information Sciences, pages 95–110. Springer, 2008.

[64] M. Grant and S. Boyd. CVX: Matlab software for disciplined convex programming, version 2.1. `http://cvxr.com/cvx`, March 2014.

[65] M. Grossglauser and D. N. C. Tse. A framework for robust measurement-based admission control. *IEEE/ACM Trans. Netw.*, 7(3):293–309, June 1999. ISSN 1063-6692.

[66] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. Nox: Towards an operating system for networks. *SIGCOMM Computer Communication Review*, 38(3):105–110, July 2008. ISSN 0146-4833.

[67] H. Gupta, A. Mahanti, and V. Ribeiro. Revisiting coexistence of poissonity and self-similarity in Internet traffic. In *Proc. of MASCOTS*, pages 1–10, September 2009.

[68] S. Han, K. Jang, K. Park, and S. Moon. Packetshader: A gpu-accelerated software router. In *Proc. of ACM SIGCOMM*, pages 195–206, 2010.

[69] M. Handley, O. Hodson, and E. Kohler. Xorp: an open platform for network research. *SIGCOMM Computer Communication Review*, 33(1):53–57, 2003. ISSN 0146-4833.

[70] S. Hassas Yeganeh and Y. Ganjali. Kandoo: A framework for efficient and scalable offloading of control applications. In *Proc. of HotSDN*, pages 19–24, 2012.

[71] J. He and J. Rexford. Toward internet-wide multipath routing. *Network, IEEE*, 22(2):16–21, March 2008. ISSN 0890-8044.

[72] N. J. Higham. Computing the nearest correlation matrix—a problem from finance. *IMA Journal of Numerical Analysis*, 22(3):329–343, 2002.

[73] I. Houidi, W. Louati, D. Zeghlache, P. Papadimitriou, and L. Mathy. Adaptive virtual network provisioning. In *Proc. of ACM SIGCOMM VISA '10 workshop*, pages 41–48, 2010.

[74] Y. Jiang, P. J. Emstad, V. Nicola, and A. Nevin. Measurement-based admission control: A revisit. In *Seventeenth Nordic Teletraffic Seminar, Fornebu*, pages 25–27, 2004.

[75] E. Keller and J. Rexford. The "platform as a service" model for networking. In *Proc. of INM/WREN '10*, April 2010.

[76] F. P. Kelly. Notes on effective bandwidths. Number 4 in Royal Statistical Society Lecture Notes, pages 141–168. Oxford University, 1996.

[77] J. Kilpi and I. Norros. Testing the gaussian approximation of aggregate traffic. In *IMW '02*, pages 49–61, 2002.

[78] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. kvm: the linux virtual machine monitor. In *Proc. of the Linux Symposium*, June 27th–30th 2007.

[79] J. M. Kleinberg. Single-source unsplittable flow. In *Proc. of the 37th Annual Symposium on Foundations of Computer Science*, pages 68–77, 1996.

[80] E. W. Knightly. Second moment resource allocation in multi-service networks. In *Proc. of ACM SIGMETRICS*, pages 181–191, 1997.

[81] E. W. Knightly. Resource allocation for multimedia traffic flows using rate variance envelopes. *Multimedia Systems*, 7(6):477–485, 1999. ISSN 0942-4962.

[82] D. Knol and J. ten Berge. Least-squares approximation of an improper correlation matrix by a proper one. *Psychometrika*, 54(1): 53–61, 1989. ISSN 0033-3123.

[83] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The click modular router. *ACM Transactions on Computer Systems (TOCS)*, 18(3):263–297, 2000.

[84] T. Koponen, M. Casado, N. Gude, et al. Onix: A distributed control platform for large-scale production networks. In *Proc. of OSDI*, 2010.

[85] W. Leland, M. Taqqu, W. Willinger, and D. Wilson. On the self-similar nature of Ethernet traffic. *IEEE/ACM Trans. Networking*, 2 (1):1–15, February 1994.

[86] D. Levin et al. Logically centralized?: state distribution trade-offs in software defined networks. In *Proc. of HotSDN*, pages 1–6, 2012.

[87] C. Li, A. Burchard, and J. Liebeherr. A network calculus with effective bandwidth. *Networking, IEEE/ACM Transactions on*, 15(6): 1442 –1453, dec 2007. ISSN 1063-6692.

[88] L. Li, M. Buddhikot, C. Chekuri, and K. Guo. Routing bandwidth guaranteed paths with local restoration in label switched networks. *Selected Areas in Communications, IEEE Journal on*, 23(2): 437–449, Feb 2005. ISSN 0733-8716.

[89] J. Liebeherr, A. Burchard, and F. Ciucu. Delay bounds in communication networks with heavy-tailed and self-similar traffic. *IEEE Trans. Inform. Theory*, 58(2):1010–1024, 2012.

[90] Linux Foundation. Opendaylight. http://www.opendaylight.org, 2014.

[91] J. Lischka and H. Karl. A virtual network mapping algorithm based on subgraph isomorphism detection. In *Proc. of ACM VISA*, pages 81–88, 2009.

[92] Y. Liu, S. O. Amin, and L. Wang. Efficient fib caching using minimal non-overlapping prefixes. *SIGCOMM Computer Communication Review*, 43(1):14–21, January 2013. ISSN 0146-4833.

[93] P. Loiseau, P. Goncalves, G. Dewaele, P. Borgnat, P. Abry, and P. Primet. Investigating self-similarity and heavy-tailed distributions on a large-scale experimental facility. *IEEE/ACM Trans. Networking*, 18(4):1261–1274, Aug. 2010.

[94] J. Lu and J. Turner. Efficient mapping of virtual networks onto a shared substrate. Technical report, Washington University in St. Louis, 2006.

[95] R. Lübben, M. Fidler, and J. Liebeherr. A Foundation for Stochastic Bandwidth Estimation of Networks with Random Service. In *Proc. of IEEE INFOCOM*, pages 1817–1825, April 2011.

[96] LXC. LXC. `https://linuxcontainers.org/`.

[97] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright. Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks. RFC 7348 (Informational), August 2014. URL `http://www.ietf.org/rfc/rfc7348.txt`.

[98] MAWI. MAWI working group traffic archive. samplepoint-F - 30.03.2012 00:00, 2012. `http://mawi.wide.ad.jp/mawi/ditl/ditl2012/`.

[99] N. McKeown et al. Openflow: Enabling innovation in campus networks. *SIGCOMM Computer Communication Review*, 38(2):69–74, March 2008. ISSN 0146-4833.

[100] B. Melamed and W. Whitt. On arrivals that see time averages. *Oper. Res.*, 38(1):156–172, Feb. 1990.

[101] J. C. Mogul et al. Devoflow: cost-effective flow management for high performance enterprise networks. In *Proc. of ACM Hotnets-IX*, pages 1:1–1:6, 2010.

[102] M. Moshref, M. Yu, and R. Govindan. Resource/accuracy tradeoffs in software-defined measurement. In *Proc. of HotSDN*, pages 73–78, 2013.

[103] S. Nadas. Virtual Router Redundancy Protocol (VRRP) Version 3 for IPv4 and IPv6. RFC 5798 (Proposed Standard), March 2010. URL `http://www.ietf.org/rfc/rfc5798.txt`.

[104] J. P. Nolan. Numerical calculation of stable densities and distribution functions. *Communications in statistics. Stochastic models*, 13(4): 759–774, 1997.

[105] I. Norros. A storage model with self-similar input. *Queueing Systems*, 16(3):387–396, September 1994.

[106] I. Norros. On the use of fractional Brownian motion in the theory of connectionless networks. *IEEE J. Select. Areas Commun.*, 13(6): 953–962, Aug. 1995.

[107] NTT. Ryu controller. http://osrg.github.io/ryu, 2014.

[108] OpenVZ. OpenVZ. `https://openvz.org/`.

[109] P. Pan, G. Swallow, and A. Atlas. Fast Reroute Extensions to RSVP-TE for LSP Tunnels. RFC 4090 (Proposed Standard), May 2005. URL `http://www.ietf.org/rfc/rfc4090.txt`.

[110] K. Park and W. Willinger. *Self-Similar Network Traffic and Performance Evaluation*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 2000.

[111] V. Paxson and S. Floyd. Wide-area traffic: The failure of Poisson modeling. *IEEE/ACM Trans. Networking*, 3(3):226–244, 1995.

[112] V. Paxson, G. Almes, J. Mahdavi, and M. Mathis. Framework for IP Performance Metrics. RFC 2330 (Informational), May 1998. URL `http://www.ietf.org/rfc/rfc2330.txt`. Updated by RFC 7312.

[113] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal. Fastpass: A centralized "zero-queue" datacenter network. In *Proc. of ACM SIGCOMM*, pages 307–318, 2014.

[114] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, and S. Shenker. Extending networking into the virtualization layer. In *Proc. of ACM HotNets-VIII*, October 2009.

[115] A. Philippe and M. Viano. Random sampling of long-memory stationary processes. *Journal of Statistical Planning and Inference*, 140(5):1110–1124, 2010.

[116] R. Presuhn. Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP). RFC 3416 (INTERNET STANDARD), December 2002. URL `http://www.ietf.org/rfc/rfc3416.txt`.

[117] Quagga. Quagga routing suite. `http://www.nongnu.org/quagga/`.

[118] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker. Abstractions for network update. In *Proc. of ACM SIGCOMM*, pages 323–334, 2012.

[119] G. Rétvári, Z. Csernátony, A. Körösi, J. Tapolcai, A. Császár, G. Enyedi, and G. Pongrácz. Compressing ip forwarding tables for fun and profit. In *Proc. of ACM HotNets-XI*, 2012.

[120] V. Ribeiro. *Multiscale queuing, sampling theory, and network probing*. PhD thesis, Rice University, May 2005.

[121] A. Rizk and M. Fidler. Non-asymptotic End-to-end Performance Bounds for Networks with Long Range Dependent FBM Cross Traffic. *Computer Networks*, 56(1):127–141, 2012.

[122] A. Rizk and M. Fidler. Statistical End-to-end Performance Bounds for Networks under Long Memory FBM Cross Traffic. In *Proc. of IWQoS*, 2010.

[123] A. Rizk, Z. Bozakov, and M. Fidler. H-Probe: Estimating traffic correlations from sampling and active network probing. *CoRR*, abs/1208.2870, July 2012.

[124] A. Rizk, Z. Bozakov, and M. Fidler. Estimating traffic correlations from sampling and active network probing. In *Proc. of IFIP Networking Conference*, May 2013.

[125] L. Rizzo, M. Carbone, and G. Catalli. Transparent acceleration of software packet forwarding using netmap. In *Proc. of IEEE INFOCOM*, pages 2471–2479, March 2012.

[126] L. Rizzo. netmap: A novel framework for fast packet i/o. In *Proc. of USENIX Security*, pages 101–112, August 2012.

[127] C. E. Rothenberg, M. R. Nascimento, M. R. Salvador, C. N. A. Corrêa, S. Cunha de Lucena, and R. Raszuk. Revisiting routing control platforms with the eyes and muscles of software-defined networking. In *Proc. of HotSDN*, pages 13–18, 2012.

[128] M. Roughan, M. Thorup, and Y. Zhang. Traffic engineering with estimated traffic matrices. In *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement*, pages 248–258, 2003.

[129] G. Samoradnitsky and S. Taqqu. *Stable Non-Gaussian Random Processes: Stochastic Models with Infinite Variance*. Stochastic Modeling Series. Taylor & Francis, 1994. ISBN 9780412051715.

[130] N. Sarrar, S. Uhlig, A. Feldmann, R. Sherwood, and X. Huang. Leveraging zipf's law for traffic offloading. *SIGCOMM Computer Communication Review*, 42(1):16–22, January 2012. ISSN 0146-4833.

[131] C. Shannon, E. Aben, kc claffy, and D. E. Andersen. The CAIDA UCSD anonymized internet traces - 16.09.2010, 2010. http://www.caida.org/data/passive/passive_2010_dataset.xml.

[132] R. Sherwood, G. Gibb, K. Yap, G. Appenzeller, N. McKeown, and G. Parulkar. Flowvisor: A network virtualization layer. Technical report, Stanford University, 2009.

[133] M. Skutella. Approximating the single source unsplittable min-cost flow problem. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 136–145, 2000.

[134] J. Strauss, D. Katabi, and F. Kaashoek. A measurement study of available bandwidth estimation tools. In *Proc. of IMC*, pages 39–44, 2003.

[135] W. Szeto, Y. Iraqi, and R. Boutaba. A multi-commodity flow based approach to virtual network resource allocation. In *Global Telecommunications Conference, 2003. GLOBECOM '03. IEEE*, volume 6, pages 3004 – 3008 vol.6, December 2003.

[136] M. Taqqu, V. Teverovsky, and W. Willinger. Estimators for long-range dependence: An empirical study. *Fractals*, 3(4):785–798, 1995.

[137] M. Taqqu, W. Willinger, and R. Sherman. Proof of a fundamental result in self-similar traffic modeling. *SIGCOMM Computer Communication Review*, 27(2):5–23, April 1997.

[138] R. Teixeira, N. Duffield, J. Rexford, and M. Roughan. Traffic matrix reloaded: Impact of routing changes. In *Passive and Active Network Measurement*, volume 3431 of *Lecture Notes in Computer Science*, pages 251–264. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-25520-8.

[139] V. Teverovsky and M. Taqqu. Testing for long-range dependence in the presence of shifting means or a slowly declining trend, using a variance-type estimator. *Journal of Time Series Analysis*, 18(3):279–304, 1997. ISSN 1467-9892.

[140] D. Thaler and C. Hopps. Multipath Issues in Unicast and Multicast Next-Hop Selection. RFC 2991 (Informational), November 2000. URL http://www.ietf.org/rfc/rfc2991.txt.

[141] A. Tootoonchian, M. Ghobadi, and Y. Ganjali. Opentm: Traffic matrix estimator for openflow networks. In *Proc. of PAM*, pages 201–210, 2010.

[142] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood. On controller performance in software-defined networks. In *Proc. of USENIX Hot-ICE*, pages 10–10, 2012.

[143] J. Vasseur and J. L. Roux. Path Computation Element (PCE) Communication Protocol (PCEP). RFC 5440 (Proposed Standard), March 2009. URL http://www.ietf.org/rfc/rfc5440.txt.

[144] D. Veitch and P. Abry. A wavelet-based joint estimator of the parameters of long-range dependence. *IEEE Trans. Inform. Theory*, 45 (2):878–897, April 1999.

[145] D. Veitch, N. Hohn, and P. Abry. Multifractality in TCP/IP traffic: the case against. *Computer Networks*, 48:293–313, 2005.

[146] A. Vidacs and J. Virtamo. Parameter estimation of geometrically sampled fractional brownian traffic. In *Proc. of INFOCOM*, volume 3, pages 1791–1796 vol.3, Mar 2000.

[147] H. Wang, H. Xie, L. Qiu, Y. R. Yang, Y. Zhang, and A. Greenberg. Cope: Traffic engineering in dynamic networks. In *Proc. of SIGCOMM*, pages 99–110, 2006.

[148] R. Wolff. Poisson arrivals see time averages. *Operations Research*, 30(2):223–231, 1981.

[149] X. Xiao, A. Hannan, B. Bailey, and L. Ni. Traffic engineering with mpls in the internet. *Network, IEEE*, 14(2):28–33, Mar 2000. ISSN 0890-8044.

[150] B. Yazici and R. Kashyap. Signal modeling and parameter estimation for 1/f processes using scale stationary models. In *Proc. of ICASSP*, volume 5, pages 2841–2844 vol. 5, May 1996.

[151] B. Yazici and R. Kashyap. A class of second-order stationary self-similar processes for 1/f phenomena. *IEEE Trans. on Signal Processing*, 45(2):396–410, Feb 1997. ISSN 1053-587X.

[152] M. Yu, Y. Yi, J. Rexford, and M. Chiang. Rethinking virtual network embedding: substrate support for path splitting and migration. *SIGCOMM Computer Communication Review*, 38:17–29, March 2008. ISSN 0146-4833.

[153] M. Yu, L. Jose, and R. Miao. Software defined traffic measurement with opensketch. In *Proc. of USENIX NSDI*, pages 29–42, 2013.

[154] M. Yu et al. Scalable flow-based networking with difane. In *Proc. of ACM SIGCOMM*, pages 351–362, 2010.

[155] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker. On the characteristics and origins of internet flow rates. In *Proc. of ACM SIGCOMM*, pages 309–322, 2002.

[156] Y. Zhu and M. Ammar. Algorithms for assigning substrate network resources to virtual network components. In *Proc. of INFOCOM*, pages 1 –12, April 2006.

OWN PUBLICATIONS

Z. Bozakov. Virtual software routers: A performance and migration study. In *Proc. of DFN-Forum Kommunikationstechnologien*, pages 35–44, 2010a.

Z. Bozakov. An open router virtualization framework using a programmable forwarding plane. In *Proc. of ACM SIGCOMM (Poster Session)*, pages 439–440, 2010b.

Z. Bozakov. Architecture and algorithms for virtual routers as a service. In *Proc. of International Workshop on Quality of Service, (IWQoS)*, June 2011.

Z. Bozakov and M. Bredel. Online estimation of available bandwidth and fair share using Kalman filtering. In *Proc. of IFIP Networking Conference*, pages 548–561, 2009.

Z. Bozakov and P. Papadimitriou. Autoslice: Automated and scalable slicing for software-defined networks. In *Proc. of ACM CoNEXT Student Workshop*, pages 3–4, 2012.

Z. Bozakov and P. Papadimitriou. OpenVRoute: An open architecture for high-performance programmable virtual routers. In *Proc. of IEEE High Performance Switching and Routing (HPSR)*, pages 191–196, July 2013.

Z. Bozakov and P. Papadimitriou. Towards a scalable software-defined network virtualization platform. In *Proc. of IEEE Network Operations and Management Symposium (NOMS)*, May 2014.

Z. Bozakov and A. Rizk. Taming SDN controllers in heterogeneous hardware environments. In *Proc. of European Workshop on Software Defined Networks (EWSDN)*, pages 50–55, Oct 2013.

Z. Bozakov and V. Sander. Openflow: A perspective for building versatile networks. In A. Clemm and R. Wolter, editors, *Network-Embedded Management and Applications*, pages 217–245. Springer New York, 2013. ISBN 978-1-4419-6768-8.

M. Bredel, Z. Bozakov, and Y. Jiang. Analyzing router performance using network calculus with external measurements. In *Proc. of International Workshop on Quality of Service, (IWQoS)*, June 2010.

M. Bredel, Z. Bozakov, A. Barczyk, and H. Newman. Flow-based load balancing in multipathed layer-2 networks using openflow and multipath-tcp. In *Proc. of Hot Topics in Software Defined Networking (Poster)*, pages 213–214, August 2014.

A. Rizk, Z. Bozakov, and M. Fidler. H-Probe: Estimating traffic correlations from sampling and active network probing. *CoRR*, abs/1208.2870, July 2012.

A. Rizk, Z. Bozakov, and M. Fidler. Estimating traffic correlations from sampling and active network probing. In *Proc. of IFIP Networking Conference*, May 2013.

CURRICULUM VITAE

PERSONAL INFORMATION

Name	**Zdravko Bozakov**
Address	Schaufelder Str. 9 D-30167 Hanover Germany
Phone	+49 178 9714 205
E-Mail	zdravko@bozakov.de
Nationality	German

WORK EXPERIENCE

03/2009 – present	**Institute of Communications Technology** **Leibniz Universität Hannover** *Research assistant / Ph.D. candidate* - network performance evaluation, traffic analysis, virtualization of network resources, software-defined networking
02/2008 – 02/2009	**KOM - Multimedia Communications Lab** **Technische Universität Darmstadt** *Research assistant / Ph.D. candidate* - performance evaluation of wireless networks, active probing methods
10/2007 – 1/2008	**KOM - Multimedia Communications Lab** **Technische Universität Darmstadt** *Student assistant* - setup and automation of network experiments
07/2002 – 12/2005	**Department of Mechanics** **Technische Universität Darmstadt** *Student assistant* - system administration of faculty network
09/1999 – 02/2007	**Lahmeyer International GmbH** *Working student* - software development and data analysis

INTERNSHIPS

11/2005 – 01/2006	**Infineon Technologies**, Munich. VHDL design and integration of a SoC component
04/2001 – 05/2001	**ABB Group**, Hanau. Assembly and testing of gas-insulated switchgear.

EDUCATION

Ph.D. Dissertation *submitted 04/2015*	*"Architectures for Virtualization and Performance Evaluation in Software Defined Networks. "*, **Leibniz Universität Hannover**, Faculty of Electrical Engineering and Computer Science
09/1999 – 08/2007	**Diplom-Ingenieur Elektrotechnik**, Technische Universität Darmstadt, Electrical engineering and information technology Major field: *Computer Systems and Networks*.
Diploma thesis	*"Unsupervised Component Extraction for Design Optimization using Feature Analysis Methods"*, **Honda Research Institute Europe** Designed a voxel-based image processing algorithm using C/C++, OpenGL and MATLAB.
Student thesis	*"Analysis and VHDL Design of a Cellular Automata Architecture with Hardware Support for a Genetic Algorithm"*, **Inst. of Microelectronic Systems** Programmed and optimized FPGA for offloading CPU calculations.
1999	**Abitur**, Einhardschule, Seligenstadt, Germany.
1996	**GCSE**, MCS, Murree, Pakistan.

Hanover, April 6, 2015

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

`http://code.google.com/p/classicthesis/`

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

`http://postcards.miede.de/`

Final Version as of July 25, 2016 (`classicthesis` Version 1.0).