SSICLOPS

Initial Design of Cloud Infrastructure and Federation Mechanisms

SSICLOPS Deliverable D1.1

Alexandru Agache[8], Gianni Antichi[9], Felix Eberhardt[2], Lars Eggert[5], Peer Hasselmeyer[4], Oliver Hohlfeld[6], Michio Honda[5], Anton Ivanov[4], Maël Kimmerlin[1], Stefan Klauck[2], Jukka Kommeri[3], Giuseppe Lettieri[7], Simon Oechsner[4], Max Plauth[2], Costin Raiciu[8], Andreas Ripke[4], Pasi Sarolahti[1], Lennart Schulte[1], Evisa Tsolakou[1], and Marcin Wójcik[9]

[1]*Aalto University*
[2]*Hasso Plattner Institute*
[3]*Helsinki Institute of Physics*
[4]*NEC Laboratories Europe*
[5]*NetApp*
[6]*RWTH Aachen*
[7]*Università di Pisa*
[8]*Universitatea Politehnica Din Bucuresti*
[9]*University of Cambridge*

February 5, 2016

This paper has received funding from the European Union's Horizon 2020 research and innovation program 2014–2018 under grant agreement No. 644866 ("SSICLOPS"). It reflects only the authors' views and the European Commission is not responsible for any use that may be made of the information it contains.

Contents

Executive Summary

SSICLOPS is aiming at empowering enterprises to create and operate high-performance private cloud infrastructures that allow flexible scaling through federation with other private clouds without compromising their service level and security requirements. To that end, work packages 1 and 3 of the SSICLOPS project are tackling challenges related to improving data transport in a single datacenter (WP1) and workload distribution across multiple datacenters (WP3). This document is joint work between these two work packages and describes the results achieved in the SSICLOPS project during its first year. It constitutes deliverable D1.1, which includes the results related to private and federated cloud infrastructures.

In its first year of operation, the SSICLOPS project has developed and investigated a number of ideas addressing challenges in multiple areas covering various parts of the end-to-end service delivery infrastructure. More specifically, SSICLOPS worked on (and continues to work on) the networking stack on endpoints, the protocols spoken between the endpoints, and the distribution of workload across the infrastructure.

SSICLOPS has been developing improvements for all three categories and thus shows broad coverage of the end-to-end cloud infrastructure. In these domains, SSICLOPS has published six papers so far, with more being in the pipeline. One piece of software, namely mSwitch, has been released to the community as open source. Other pieces are expected to follow suit. Moreover, partners of the project consortium have contributed to five IETF drafts. Accordingly, the project has made good progress towards its objectives of providing improvements of private cloud infrastructure.

Among the various achievements of the project, one highlight is the development of mSwitch, a high-performance software switch, which outperforms FreeBSD's bridge by up to eight times. Another highlight is TCP sendbuffer advertising, which has been shown to improve the average flow throughput by 5% to 15%. In the workload distribution domain, Hyrise-R adds scale-out capabilities to the in-memory data base Hyrise and makes the amount of processed read requests linearly scalable with the number of replicas.

This deliverable reports on both published and (not yet published) ongoing work. For published work, the deliverable contains only a summary of the publication and the full publication is part of the appendices of this deliverable. Ongoing work is briefly summarized in separate "work-in-progress" sections and will be reported in more detail once it has been finalized and published.

1. Introduction

SSICLOPS is aiming at empowering enterprises to create and operate high-performance private cloud infrastructures that allow flexible scaling through federation with other private clouds without compromising their service level and security requirements. To reach this goal, the project is divided into several work streams, including one which is looking at improving data transport in a single datacenter (WP1), and one which is looking at workload distribution across multiple datacenters (WP3). This document is a joint effort between these two work streams and describes the results achieved in the SSICLOPS project during its first year. It constitutes deliverable D1.1, which includes the results related to private and federated cloud infrastructures.

The deliverable reports on both published and ongoing work that has not been submitted for external publication or is under review. For published work, the deliverable contains only a summary of the publication and the full publication is part of the appendices of this deliverable. Ongoing work is briefly summarized in separate "work-in-progress" sections and will be reported in more detail once it has been published.

Besides academic papers, SSICLOPS has been working on Internet drafts and open source software. As the Internet drafts have not been released as RFCs yet, they are contained in the "work-in-progress" sections as well. SSICLOPS is aiming at releasing many pieces of its software developments as open source. So far, only one piece has been released, namely mSwitch. Other pieces are currently under preparation for future release. The software artifacts are not mentioned in their own section in this document. Rather, as they are associated with academic papers, they are described as part of the section on the respective paper.

The work during the first year of the project was related to three main topics, which constitute the main chapters of this deliverable. Each chapter starts with an introduction on how the individual pieces contribute to the goals of the overall SSICLOPS project.

The high-level cloud landscape as considered by SSICLOPS is shown in Figure 1. SSICLOPS is looking at the end-to-end connections between communicating endpoints. Such endpoints can be located in different places, including within one datacenter ("intra-cloud"), within separate datacenters connected by a (wide-area) network ("federated cloud"), and at other locations ("client"). The endpoints are connected to each other via a communication infrastructure. Inside a single datacenter, this is the local datacenter fabric, whereas in the other cases, communication is assumed to happen over the Internet.

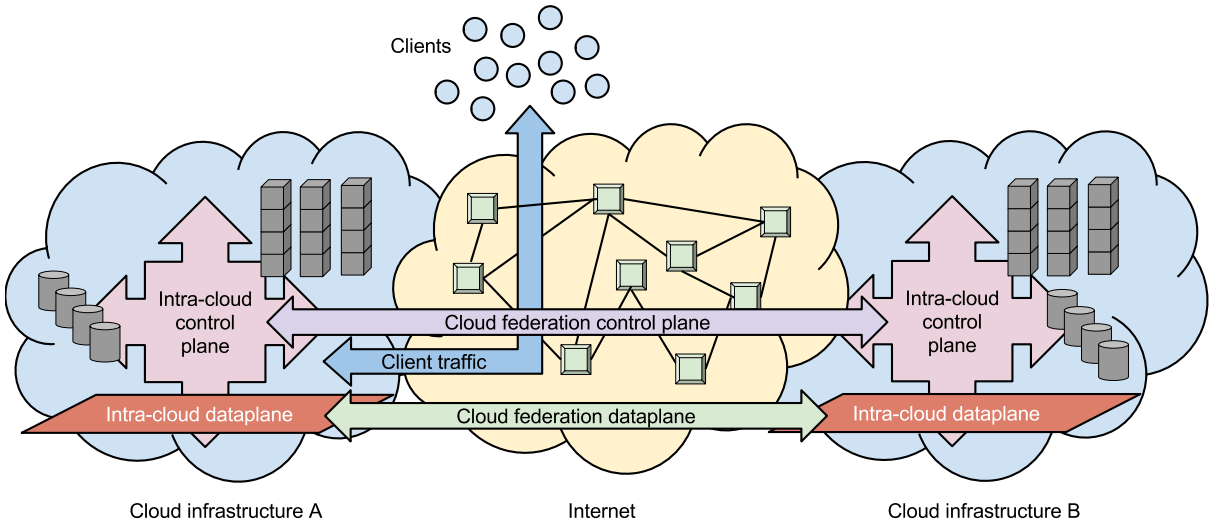

Figure 1: SSICLOPS architecture overview.

The network within a datacenter typically exhibits different properties than the (wide-area) network interconnecting several of them. The networking infrastructure within a datacenter is under the control of a single entity and can be adapted to fit the needs of the applications which are running on that infrastructure. Furthermore, the equipment in a single datacenter is physically close, making it easier to ensure short delays and high throughput between communicating endpoints than for communication over wide-area networks.

In contrast to the controlled environment of a datacenter, the Internet that is used for transporting data between datacenters and clients provides a much more diverse communication substrate. It is provided by a multitude of operators, which are following different aims and policies. The cloud federation data plane is therefore much more unreliable and performance can vary to a much larger extent than inside a single datacenter. As datacenter sites are usually far away from each other (at least when compared to equipment within a single site), characteristics of the communication paths are different from the ones inside a single datacenter. In particular, latency is larger and variability of capacity is higher.

The SSICLOPS project is aiming at improving the performance of cloud-based applications from end-to-end. Included are applications within a single datacenter and communication between datacenters. As communication properties within datacenters are different from such properties in the Internet, different problems are encountered and it is useful to consider methods that explicitly address these different conditions. Therefore, SSICLOPS is looking at both the intra-cloud data plane and the inter-cloud (or “cloud federation”) data plane.

Improving communication performance can happen in multiple ways and on multiple levels. Looking at the end-to-end communication chain, data sent to another endpoint has to go through the local network stack, the network interface card (NIC), and local and wide area networks. In the case of virtualized servers, there might be multiple software network stacks that data has

to pass through — at least one inside the virtual machine (VM) and one inside the hypervisor, possibly at each endpoint. SSICLOPS includes activities to improve performance of all parts of the communication path.

Broadly speaking, the developments within the SSICLOPS project can be split into the three categories network stack improvements, protocol improvements, and workload scheduling improvements. The content of this deliverable has been structured according to these categories.

Network stack improvements look at the communications software and hardware inside a single server. The two main areas of interest in the network stack are protocol handling (TCP/IP, Ethernet) and data passing (between applications and the operating system, within the operating system, and from the operating system to the network interface card). Included are virtualization technologies, which can increase the overhead in both areas, e.g., by encapsulation in the protocol handling domain and by virtual switches in the data passing domain.

Protocol improvements are looking at how to change the behavior of protocols (TCP/IP in particular) in order to improve the performance of end-to-end connections. The performance of networking protocols depends on various static characteristics and dynamic conditions of the path between the communicating endpoints and how the protocols adapt to both. They include latency, capacity, available bandwidth, queueing disciplines and packet loss rate, among others.

Workload scheduling is used in a very broad sense within SSICLOPS. It includes the assignment of data and processing tasks to servers within a single cloud infrastructure and in multiple, federated cloud infrastructures. On the networking side, it refers to assigning network traffic to specific paths inside a communications network (commonly called "traffic engineering").

During its first year, SSICLOPS has been addressing performance issues in the mentioned areas and the project has been looking at different approaches. On the end-hosts, data processing inefficiencies in the network stack have been identified and approaches for their mitigation have been (and continue to be) explored. Two main reason for such inefficiencies are context switches (between user and kernel space) and data copying. SSICLOPS has been looking at removing such inefficiencies using various approaches.

One approach evaluated in *Santa* (see Section 3.2) is delegating the creation of responses to requests from user space to kernel space, thereby removing the overhead of context switches between kernel and user space for such requests. In a DNS experiment, *Santa* provides a throughtput increase of up to 340% while generally reducing the median response time by up to 47%. Another approach, *FrankenStack* (see Section 3.4.1), moves the complete packet processing chain to user space, circumventing a number of problems associated with the implementation of packet processing in the operating system. On request-response workloads, FrankenStack improves throughput by 8.1% to 139.6% and reduces latency by 7.7% to 58.8%. Although the benefits of using user space stacks have been shown before, *FrankenStack* makes the transition to this approach much easier, as it re-uses the existing TCP/IP stack implementation in the kernel, avoiding the need to re-implement the stack in every application.

For software switches, SSICLOPS has developed *mSwitch*, i.e., mechanisms for improving packet distribution in software switches (see Section 3.1). *mSwitch* outperforms FreeBSD's bridge by up to 8 times. An accelerated Open vSwitch module boosting performance by 2.6 to 3 times has also been implemented. Exploiting the multi-core processing capabilities of current processors is the aim of work presented in Section 3.4.4. At the expense of packets being processed out-of-order, linear scalability in processing time per packet with the number of cores has been achieved for packet switching.

An issue appearing in virtualized systems is the synchronization between packet producers and consumers. Depending on the (relative) processing speeds of both parties, performance can be impacted in non-obvious ways. This phenomenon and some approaches for mitigating its impact are studied in the work reported in Section 3.4.3.

The investigation of the interface between the network driver and the hardware network interface card is subject of the research done on a DMA engine for FPGA network cards, as reported in Section 3.3. The implemented software allows data transfer rates well above 40 Gbps for FPGA to virtual machine transmissions. The FPGA network interface card can be used to enhance the speed of flow processing. Delegating parts of such processing to the FPGA card has demonstrated a 30-fold performance gain in comparison to software-only processing as part of the work on HyPaFilter (see Section 3.4.2).

Improvements of the performance of the network stack on endpoints are relevant to both the intra- and inter-cloud data plane, because all data transfers have to undergo processing in the network stack on all the endpoints involved in the communication, no matter whether communication is within a single cloud or between clouds. Such improvements therefore directly contribute to the projects goal of efficient data transport within a cloud and across federated clouds.

Proceeding beyond the network stack of individual hosts and looking at the end-to-end transport, the operational performance of the protocol used for data exchange is critically important to throughput and latency, both for the intra-cloud data plane and the cloud federation data plane. SSICLOPS has been looking at methods for improving this end-to-end performance in both dimensions — throughput as well as latency. High throughput and low latency are often at odds with each other and achieving both at the same time is a difficult task.

For example, *Multipath TCP* (MPTCP) aims to improve throughput for large data transfers, but is not changing the behavior of the network with regards to short flows, which usually benefit from low latency. *Datacenter TCP* (DCTCP) specifically aims at improving performance in datacenters by reducing buffer occupancy in the data plane. Small buffers reduce queueing latency between endpoints in the datacenter, which has a positive impact on applications that exchange short transactional messages, like database queries and or key/value lookups.

The combination of Multipath TCP and Datacenter TCP is currently explored by SSICLOPS (see Section 4.2.5), with the aim of providing short latencies while at the same time efficiently using all the available capacity by spreading traffic across multiple paths. Preliminary simulation

results show that the combination of protocols can reduce buffer occupancy by 60% without impacting throughput.

TCP-aNCR, introduced in Section 4.2.4, is reducing the impact of packets received out-of-order. It adapts the TCP retransmission delay automatically according to the current sending rate thereby reducing the number of retransmissions caused by reordered packets. The algorithm reduces delay spikes in varying environments by 20% compared to Linux and 40% compared to TCP-NCR while preventing throughput degradation.

Hoover (see Section 4.2.7) explores the idea of keeping all packet headers, but removing the data payload in case of network congestion. Receivers then know that there is congestion on the path from the sender and they can even infer the amount of the congestion by measuring the percentage of truncated packets. This information can then be used to react appropriately to the present congestion. Preliminary results from simulating this idea show close to optimal behavior for both short and long flows. Another approach for providing better information about the source and amount of congestion is to disseminate information about backlogged applications to the network. This method has been explored as *sendbuffer advertising* in Section 4.1. Such detailed information about the state of applications and the network can be used by management systems to reason about congestion causes and to make better decisions about how to improve performance, e.g., by migrating work (in terms of traffic and processing) to different regions in a datacenter. Simulations using the information for flow scheduling show an improvement of average flow throughput from 5% to 15%.

Client-facing traffic can benefit from using some of these different protocol improvements as well. More specifically targeted at client-facing connectivity is work on an *MPTCP proxy* (see Section 4.2.6). This proxy allows client and server applications that are not multipath-enabled to still benefit from multiple available network paths by splitting up and joining regular (single path) TCP connections into multiple MPTCP subflows.

SSICLOPS is also looking at improvements for processing the workload put on applications. Two cooperating approaches can be broadly distinguished here: *placement* of data and code, and *dispatching* requests to these locations. Content Delivery Networks (CDNs) are one way of improving access times to data. The influence of on-demand caches and their placement is being evaluated in large-scale simulations (see Section 5.3.3). Initial results of these simulations show a start time of 3 seconds for most of the viewers, whereas a traditional CDN would cause users to wait 5 to 8 seconds.

A framework for separating the task of placing data and processing from implementing the actual algorithms has been designed as part of the *Claud* system (see Section 5.2). This separation allows programmers to focus on implementing the actual function of an application while the system autonomously ensures good performance by placing data and processing close to each other.

In order to serve large workloads, the in-memory database *Hyrise* has been extended to a multi-node, clustered setup consisting of a variable number of replicas. The replicas are using

lazy replication for ensuring consistency among them (see Section 5.1). This extension has been shown to exhibit linear scalability in the number of read requests with the number of replicas. Performance can be further improved by using an optimized query dispatcher (see Section 5.3.1).

Improving the request throughput in a more general fashion is the subject of *CloudTalk* (see Section 5.3.4). CloudTalk makes network information available to clients, which can then make informed decisions about which server to contact for getting their tasks done, e.g., which replica to choose from a set of servers hosting the same piece of data. Experimental results show that *CloudTalk* can deliver up to 30% better file read times for HDFS and 80% better write performance.

The workload distribution methods developed and being worked on in the SSICLOPS project contribute to the optimization of placing workloads in single and federated clouds. The methods improve response times for different use cases and cover a broad range of applications, including access to data, use of processing power, and a combination of these two.

Another direction for optimization is improving energy efficiency of computations. SSICLOPS has been looking at reducing energy consumption by consolidating idle virtual machines on specialized, energy-efficient "park" servers, while assigning active VMs to different servers. Experimental results show a clear improvement of 9% to 48% in the total energy efficiency.

The work reported on in this deliverable demonstrates good progress towards the main objective of the SSICLOPS project, namely the creation of high-performance private cloud infrastructure. Infrastructure improvements are being developed on all levels, including the network stack, the network protocols, and the placement of workloads. These topics are contained in Sections 3, 4, and 5, respectively. Section 2 introduces the state of the art to detail which technologies SSICLOPS can base its work on. Section 6 wraps up the deliverable. Copies of the published papers can be found in the appendices.

2. State of the Art

This chapter details the state of the art in networking and workload scheduling — the topic areas of highest importance to this deliverable. The focus is on briefly introducing those relevant pieces of work that are influential in the field. This section is by no means comprehensive. It should rather give an overview of technologies that are most relevant to the work in the SSICLOPS project.

2.1. Networking

This section looks at networking protocols that are optimized for intra-data center or inter-data center communication. As SSICLOPS is aiming at improving performance end-to-end, all parts of the network and related protocols are relevant to the project. The descriptions contained in this section are not provided to detail every aspects of the mentioned work, but rather to provide an overview of the various directions of the protocol design space which have been explored in previous work. This section helps with the identification of impairments of existing schemes and increases understanding of where there is room for improvement. The protocols discussed in this section are therefore considered as baseline technologies on which the SSICLOPS project can build.

When it comes to protocols that exhibit good performance in data centers, a number of them are tailored for this specific environment. Many of these protocols rely on changes made to previously existing ones that are widely deployed, mainly, of course, changes to TCP itself. TCP being the most widely used protocol, this does not come as a surprise. Some others are protocols designed from scratch, sometimes based on already existing techniques and put together to perform better in data centers. There are a number of algorithmic schemes as well, again designed to improve performance in the data center.

Based on the observation that the average round-trip time (RTT) in a data center is on the order of microseconds, the TCP variant described in [89] improves TCP retransmission timeout (RTO) handling by enabling timeouts with microsecond granularity. The goal is to improve the overall throughput, reduce the impact of packet loss, and to effectively avoid incast collapse. Packet loss is not avoided with this approach and for such RTO values, high resolution timers are necessary. An addition to this approach is to either reduce the delayed ACKs' timeout period to a few microseconds or to disable the timeouts completely, in order to get rid of spurious retransmissions with the fine-grained timers.

Datacenter TCP (DCTCP), as described in [3, 4, 15], is a TCP variant specifically designed for data centers, so it can achieve high burst tolerance, low latency and high throughput. It makes use of a threshold K to determine if the network is congested and with the mechanism of Explicit Congestion Notification (ECN) marking provides congestion feedback to the end hosts. The DCTCP source reacts to this marking scheme by reducing the window by a factor that depends on the number of marked packets. DCTCP only works if both ends, sender and receiver, are DCTCP-capable.

Double-Threshold DCTCP (DT-DCTCP) [27] improves on the original DCTCP by using two parameters K1 and K2 instead of the threshold K. K1 is to start ECN marking in advance, informing that there might be congestion in the network and if the queue length decreases to a value lower than threshold K2, congestion message is released.

DCTCP with Weighted Random Early Detection (WRED) [57] ensures that DCTCP flows do not starve when coexisting with flows from other, more aggressive TCP variants. There are two changes proposed to the classical DCTCP. First, the fraction of marked packets to be updated with every acknowledgment packet received and second, decrease of the congestion window whenever an ECN-Echo is received.

High-bandwidth Ultra-Low Latency architecture (HULL) [5] is an architecture that simultaneously balances ultra-low latency and high bandwidth utilization. The HULL architecture consists of three main components: DCTCP congestion control, phantom queues (PQ), and packet pacing. The PQ mechanism is practically a counter that sets ECN marks based on link utilization and not on queue occupancy, achieving low fabric latency because of the signaling before queueing occurs. HULL reacts to these ECN marks using DCTCP's mechanism. In the HULL architecture, pacing takes place in the hardware after the last source of bursty transmission and only the packets that belong to large flows are paced.

Deadline-Aware DCTCP (D^2TCP) [88] is a TCP-based data center network protocol that aims at achieving high bandwidth for background flows while meeting the Online Data Intensive (OLDI) applications' deadlines. At the same time, it requires no changes to the switches. D2TCP changes the congestion window size through a gamma-correction function based on deadline information and encountered congestion. If the flows have no deadlines, then D2TCP functions as DCTCP.

TCP Westwood (TCPW) [42, 63] is a modified version of the TCP congestion window algorithm on the sender side. The basis of the mechanism is the continuous monitoring of the bandwidth used via the rate of returning ACKs and not relying on packet loss only. The protocol selects a slow start threshold and a congestion window based on the bandwidth used at the time of congestion.

Incast Congestion Control TCP (ICTCP) [92] is an incast congestion control scheme for TCP on the receiver side that regulates the receiving window. It is an approach for many-to-one, low-latency, high-bandwidth networks and achieves almost zero timeout and high goodput. ICTCP

tries to avoid packet loss before incast congestion happens by adjusting the receive window size of each connection so the aggregate traffic size does not lead to congestion.

Multipath TCP (MPTCP) [75] is an extension of the TCP protocol which uses multiple paths by spreading data from a single TCP connection to multiple subflows that can take different paths in the network. Each subflow has its own congestion window and subflows with larger windows increase them faster than the ones with smaller windows. MPTCP achieves increased network utilization since underutilized and idle links can be explored. It also provides fairer allocation of capacities to flows and robustness by using multiple paths and by avoiding to send traffic to congested ones.

Data Center UDP (DCUDP) [94] is a UDP-based protocol which uses ECN to provide congestion control and achieves very high throughput for normal and short flows. DCUDP inherits the architecture from UDT, where UDP socket communication is used and control mechanisms for reliable transmission are deployed. To support ECN, Congestion Window Reduced and ECN-Echo bits are added and DCUDP uses these bits in the same way as TCP-ECN.

Rate Control Protocol (RCP) [35] is a protocol that is mainly created to keep flow completion times low. It achieves this goal by the routers "stamping" a rate $R(t)$ of every link to the packets that pass through. The sender receives these values and gets informed of the slowest rate or bottleneck and adjusts the rate that it should be using, avoiding this way slow start and managing to complete flows quickly and in fairness.

D^3 ("Deadline-Driven Delivery") [91] is a control protocol specifically designed for data centers. It is aware of the flows' deadlines and uses explicit rate control to give them bandwidth according to these deadlines. End hosts, by getting the information about flow deadline and size, which is supposed to be available at flow initiation time, ask the network for the required rate of transmission. The goals of the protocol are to maximize the application throughput, to accommodate flow bursts and high network utilization.

Preemptive Distributed Quick (PDQ) [47] is a protocol designed for quick flow completion, taking into account and meeting flow deadlines. PDQ performs dynamic decentralized scheduling by allowing switches gather information about flow workloads that then propagates through packet headers. PDQ takes allocation decisions with the use of a range of scheduling disciplines and informs about the rate that packets should be sent with.

pFabric [7] is a data center transport design that is based on the idea of decoupling flow scheduling from rate control. For flow scheduling, end hosts put a priority number on every packet and the switches decide which packets to accept and which to drop based on this number. As for rate control, all flows start at line rate and decrease their sending rate only if they see high and persistent loss.

PASE [66] is a transport framework that combines existing transport strategies. It adopts the self-adjusting endpoints from TCP strategies, where senders make their rate decisions based on the observed network conditions. From the queueing algorithms it adopts the arbitration, where

a part of the network allocates rates to each flow and it uses in-network prioritization like in pFabric, where switches schedule and drop packets based on some priority number.

Hedera [40] is a flow scheduling system that schedules a multi-stage switching fabric to efficiently utilize aggregate network resources. The scheduling technique that is deployed is rerouting of traffic according to the computation of non-congested paths from the switches. The goal of the scheme is to maximize aggregate network utilization without impact on active flows.

DeTail [95] uses cross-layer mechanisms to reduce the long tail of flow completion times. The goals of the stack are to reduce packet losses and retransmissions, prioritize latency-sensitive flows and balance traffic across multiple paths. The design detects congestion at lower network layers and finds different paths with less congestion to destination, leading in this way the routing of the flows. The lossless fabric deployed ensures that losses occur only because of hardware errors or failures.

FastPass [72] is a data center network architecture that makes use of centralized arbitration to determine the time at which packets should be transmitted and which paths should be used. With FastPass there is no congestion at the switches and the endpoints can transmit at wire-speed since the architecture uses a timeslot allocation and a path assignment algorithm. The goal is to send the packets in a way that will avoid congestion and make use of full paths dealing with problems that might arise in the network a priori.

SWAN [48] is a system that boosts the utilization of inter-data center networks by centrally controlling when and how much traffic each service sends and by frequently reconfiguring the network's data plane to match current traffic demand. SWAN employs three priority levels: interactive, elastic, and background. All traffic types, except interactive, must inform the controller of their inter-data center demands. The controller computes a global allocation and then updates the forwarding state of the switches via software-defined networking (SDN) mechanisms. The main interesting points of SWAN are the global allocation algorithm, the congestion-free transition plan between forwarding states, and a dynamic tunnel allocation method that is used to cope with a limited number of OpenFlow rules.

B4 [52] is a private WAN that connects Google's data centers. It exhibits some unique characteristics given the moderate number of sites, Google's total control of network and applications, together with the bulk of the traffic being bandwidth-intensive. B4 consists of three layers. Each data center has a switch hardware layer that mainly deals with forwarding traffic using custom-built switches. Next, a site controller enables distributed routing and central traffic engineering. Finally, the global layer is responsible for centralized control of the entire network using SDN. B4's traffic engineering is deployed as an overlay over the routing service.

2.2. Workload Scheduling

Clouds are sets of computing and networking infrastructure, put together to handle the work given to it by its customers. The workload needs to be distributed appropriately within clouds

as well as across multiple clouds in the federated cloud case. As SSICLOPS is looking at the complete end-to-end infrastructure, the placement of work happens on different hierarchy levels: federated clouds, data center, rack, computer system, NUMA-node, core. This section looks at the state of the art in workload scheduling at the intra-system level, the data center level, and the federated cloud level.

A workload inside a single system ("intra-system") is typically executed by threads performing a task. The threads use system resources such as CPUs, memory and devices. The utilization of these resources as well as the utilization level can change over time. Scheduling of workload inside a system tries to avoid fundamental problems such as over-utilized resources by placing the threads and data accordingly.

The idealized machine model often consists of multiple processors connected via a shared bus to one single block of memory. The memory access of those systems is uniform in the sense that every core has the same latency and bandwidth to every memory address in the system [76]. However, modern server systems are non-uniform memory access (NUMA) systems with multiple processors having their own share of the whole memory lcoally connected. Several nodes are coupled with an interconnection network [24, 55, 58]. Memory accesses on such systems have different costs depending on the distance between the accessing thread and the memory location. Bad placement of threads and data can have severe impact on the overall performance of the system due to congestion of the underlying interconnection network. The NUMA architecture solves the core count scaling problem of UMA architectures. However, in contrast to UMA-based systems, not only thread but also data placement has to be considered. Both have major influence on scalability and performance of such systems. Delays due to remote memory access are not the main issues [32]. The major problems are contention for memory controllers and interconnections because they lead to congestion and an imbalanced system.

Several approaches exist for NUMA-aware scheduling to handle these issues, for example the critical path-based thread placement strategy introduced in [84]. [20] and [21] present bubble-based thread placement strategies (*Affinity* and *ForestGOMP*). These strategies use thread cooperation affinities to schedule the threads based on the framework stated in [87]. [8] shows and explores thread placement strategies on different NUMA Architectures. *DINO* [17] extends the approach of the *Distributed Intensity (DI)* technique (see [97] for more details) to work on NUMA systems.

Data placement strategies have similar goals as the thread placement strategies: memory pages should be distributed among the nodes in the NUMA system to avoid out-of-memory nodes; resource contention should be avoided; data should be placed close to the accessing threads. There are four major actions dealing with memory pages: initial placement; migration: memory pages get migrated from one NUMA node to another; interleaving: memory pages get equally distributed among the NUMA nodes; replication: memory pages get replicated among all or a subset of the NUMA nodes.

The strategies implementing those actions are static or dynamic. Current operating systems use static data placements such as *first touch*: The data is placed on the node where the first reading or writing thread is executed [8]. Other static policies are *round-robin*, *full*, *block*, *skew-mapping* and *prime-mapping* (see for example [51] for a detailed explanation).

Dynamic strategies adapt the placement of data over time. Therefore, data access patterns have to be measured to place memory pages appropriately. For example, when a memory page is mostly accessed by threads from one specific node, it is reasonable to use the migration strategy. Interleaving a memory page is recommended if that page is read and written by threads from several nodes. If a memory page is accessed read-only by threads from several nodes, replication of that page across the nodes is advisable.

Examples of data placement strategies can be found in [19], [61] and [68]. In order to evaluate memory migration strategies, [17] calculates a metric called *Saved Remote Accesses*. The metric is defined as saved remote memory accesses due to migration in relation to not migrating the memory pages. However, this approach does not consider that memory pages can be accessed from different threads simultaneously. [32] presents a mechanism to dynamically migrate, interleave or replicate memory pages according to certain metrics gathered during runtime of the NUMA system. The authors propose an algorithm called *Carrefour* which especially tries to avoid congestion in the memory hierarchy. If a page was accessed from more than one node, the page is either replicated or interleaved. The page is replicated if all accesses to the page are read-only. If the access is mixed read and write, the page gets interleaved. While replication improves performance, it also increases the memory footprint dramatically.

In a NUMA system, the scheduler and the memory management subsystem have, to some degree, conflicting goals. The scheduler tries to distribute the computing load evenly across the system. This considers reduction of contention of threads for last level caches thus placing memory-intensive threads together with compute intensive threads on the same node. However, the scheduler does not consider memory locality and will move threads off overloaded nodes if necessary. Rescheduled threads supposed to reduce last-level cache contention can now lead to increased interconnection utilization if the memory resides on a remote node. The memory management on the other hand tries to allocate the data local to the threads. [32] is running several benchmarks using as many threads as there are cores in their test system with the first touch memory allocation and interleaved memory allocation. The results were mixed. Several benchmarks performed better with first touch, others with interleaved memory pages and on some benchmarks neither strategy has significant advantages. Neither policy is suited for mixed workloads. Memory placement policies therefore have to adapt to the actual workloads. For example, focusing on locality can lead to unbalanced systems with respect to data and thread placement if one thread initializes data for other threads. The data is placed on the node of the initializing thread and the other threads get placed on other nodes by the scheduler. Now there is a trade-off between evenly distributed threads and locality of the memory in the system. The conclusion is that the scheduler and the memory management subsystem have to cooperate in order to achieve the common goal in a NUMA system: improve thread and data placement.

Going beyond scheduling inside single machines, workloads in clouds need scheduling. Workloads in infrastructure-as-a-service clouds typically come with the granularity of virtual machines (VMs). Scheduling in clouds therefore refers to the placement of virtual machine instances on hosts in a cloud data center. Considering that containers are very similar to virtual machines, in particular when looking at their placement, scheduling as discussed here includes VMs as well as containers and, potentially, other virtualization techniques.

Scheduling is currently mainly focused on the goal of avoiding Service Level Agreement (SLA) violations, besides fulfilling specific requirements such as the type of virtualization. However, it is possible to consider this problem with an energy-efficiency focus for example [83], or to ensure high-availability. Scheduling is a bin packing problem and has been proven to be NP-hard. There are two main approaches to scheduling VM instances. The pro-active approach consists of selecting the optimum host when creating the instance. When placing an instance, the scheduler considers all the hosts and selects the "best" one (according to some metric), e.g. the one that has the largest amount of resources available. The second approach, complementary to the first one, is reactive and consists of moving the instance between the hosts to optimize some objective function, for example based on energy or costs.

As a practical example of pro-active scheduling, *Nova*, OpenStack's scheduler, performs scheduling at different levels. If the cloud is using several cells, a first scheduler instance determines in which cell the instance should be placed [69]. In each cell, a Nova scheduler determines the optimal node to put the virtual machine on. This placement happens in two steps. The scheduler first applies filters to determine which nodes are suitable to place the new instance. Filters can include criteria such as CPU cores, disk space, and hardware features. Once the scheduler has determined the subset of hosts able to host the virtual machine, they are weighted using different metrics, including memory utilization and network bandwidth utilization. The host with the lowest weight will be selected as the host for the new instance [70]. However, OpenStack does not provide built-in host overload detection; reactive scheduling mechanisms are therefore not used.

Many policies for scheduling virtual machines in a data center have been devised. The first and simplest one is round-robin [85]. Priority-based scheduling assigns a priority value to each instance. This idea has been applied by Xiao et al. [93]. Some recent work has been focusing on using the load variation patterns to place virtual machines in order to avoid wasting resources [18]. This is a first step in the direction of using a load prediction to place virtual machines. Similarly, some scheduling takes into account the expected workload of the VM to place. Different algorithms are then used to predict the workload of each virtual machine.

Cloud consolidation places and migrates virtual machines across hosts in order to reduce the number of hosts needed to run all the virtual machines (e.g., for cost and/or energy-efficiency reasons) while maintaining SLA guarantees [18, 22, 31]. Cloud consolidation requires the virtualization system to be aware of the current workload of every virtual and physical machine. A first approach is to monitor the current state of the host and to react to an over-load or under-load situation [13]. Another approach is to predict the future load of the VM using either

deterministic or non-deterministic algorithms [14]. Some systems were designed to take into account specific hardware, such as low-energy hosts [56] or I/O capabilities [2].

However, because perfect scheduling is not possible due to the variability of the workload, other techniques are necessary to limit the impact of SLA violations and to improve the efficiency of a data center. Hirofuchi et al. demonstrated, for example, that using a post-copy technique for memory during live migration of VMs permits to react faster to a load burst and can enable a slightly lower performance degradation during SLA violations than the pre-copy technique [45].

A variant of placement algorithms particularly relevant to SSICLOPS is traffic- or network-aware placement. Such algorithms take into account inter-VM connection requirements, mainly in the form of necessary bandwidth, and try to minimize the traffic load on the network. The algorithms affect both network and computational workload at the same time as placement of VMs affects resource usage of both domains.

[65] describes a VM placement strategy that takes into account traffic between virtual machines in order to optimize data center traffic flows. To that end, the algorithm does not place individual VMs, but rather groups of VMs. The basic strategy is to separately group VMs (aiming for high intra-group traffic) and hosts (aiming for low-cost connection clustering), followed by assigning each group of VMs to a group of hosts.

The traffic-aware optimization problem of [65] is extended in [16] by defining a *Min-Cut Ratio-aware VM Placement (MCRVMP)* problem, which is NP-hard. The extended problem tries to take into account traffic variations by minimizing the maximum network cut load, thus aiming for a stable placement even under varying traffic load. The evaluation shows long problem solving times even for the proposed heuristics, from less than one minute to half an hour for a DC topology with 64 nodes and placing up to 1280 VMs.

A placement algorithm that takes an application description (complete with availability and connection constraints) and a data center description and translates these into tree models is presented in [54]. The algorithm places VMs on physical machines by first grouping VMs and placing groups on server racks, then checking if the individual VM demands (such as amount of memory or number of CPU cores) are met. The algorithm types used for the optimization are graph cuts, graph matching and bin-packing, all of which are NP-hard or NP-complete.

Erickson et al. consider network-aware VM placement in the context of a larger process [38]. Here, the optimization algorithms are only the second step in an envisioned cycle of Measure – Optimize – Migrate: first, information from the data center is collected, which is then used to calculate a new, better placement. Finally, this placement is implemented by migrating VMs. The optimization goal is to minimize the maximum resource utilization (CPU or network link), similar to [16] which considers the maximum network cut load. The evaluation shows a general trend for performance improvements (in terms of requests/sec for web, and runtime for MapReduce/Hadoop) with increased network knowledge of the algorithm. Greedy Fill performs best, an algorithm that starts with a clean slate and uses CPU and network utilization, topology,

link capacity, and routing information. The runtime of the proposed heuristics is in the tens of seconds.

Silo [53] is a Microsoft approach for traffic management and QoS guarantees in data centers. VMs and thus the tenants owning them get guarantees in terms of bandwidth, maximum packet sizes, and burst sizes, which, taken together, result in a maximum message latency. Silo is evaluated in a small-scale testbed, a medium-sized packet level simulation and a large-scale flow level simulation. The results show that Silo can guarantee message latency, but can lead to lower utilization in the network compared to Oktopus [10] (albeit still being higher than greedy locality). However, for a high data center occupancy, it can accept more tenant requests because tenants can complete their jobs on time and leave the system more quickly.

3. Networking Stack Improvements

Cloud consumers nowadays face a difficult tradeoff: make use of extremely high-performance, scalable, and reliable public clouds, while suffering their substantial negative impact on security and privacy, or develop their own private cloud infrastructure that will lack many of the economies of scale experienced by hyperscale cloud providers.

In this context, the SSICLOPS project yearn for enabling private cloud operators to run high-performance, scalable infrastructures at a fraction of the cost needed to support all the computations locally with existing public cloud technologies. In particular, the ambition is to empower enterprises to create and operate high-performance private cloud infrastructure that allows flexible scaling through federation with other private clouds without compromising their service level and security requirements.

Enabling a high-performance and scalable infrastructure requires an optimal dataplane design. It is critically important that the hardware making up the network fabric and the network stack software used by the individual components of the cloud infrastructure for end-to-end communication are optimized for one another, because the performance of the communication dataplane can be a limiting factor to the overall utilization and performance of a cloud infrastructure. The dataplane of a cloud infrastructure indeed, provides the communication substrate for both the control plane management traffic for workload scheduling, etc. and the workloads themselves. An inefficient dataplane slows down control plane functions such as workload monitoring, scheduling and optimization, as well as decreasing the potential performance of the workloads themselves. Logically, the dataplane consists of network fabric hardware, i.e., the switches and network interface cards of the various computer, storage and other appliances in a datacenter, as well as the software stacks that are involved in providing end-to-end connectivity between devices that are part of the fabric.

In this scenario SSICLOPS tackles the performance problems from both the network fabric hardware and the software stack prespective. As for the latter we propose a PCIe DMA engine to support the virtualization of 40 Gbps FPGA-accelerated network appliances (Section 3.3) and a high-scalable, modular software switch (Section 3.1), while for the former we developed a faster packet delivery mechanism (Section 3.2).

Direct memory access (DMA) is a feature of computer systems that allows certain hardware subsystems to access main system memory independently of the central processing unit. The DMA represents the first performance bottleneck when data needs to be moved from the hardware to the software. The flexibility offered by programmable hardware such as an FPGA-based

network appliance has created an opportunity for impact where one previously did not exist. We now have the ability for the host, OS and application to utilize heterogeneous implementation environments to provide a richer diversity of service at higher performance. In the past, issues of differences in implementation, difficulty in updates, and general complexity surrounding ongoing maintenance and the inflexibility of in-silicon implementations have impeded the wide adoption of per-application specialization with any useful generality or at any significant scale. This flexibility allows developing and adapting scheduling, queue management and congestion/load notification schemes that are currently fixed in hardware. In this context we propose a PCIe DMA engine (see Section 3.3) that allows boosting the performance of virtual network appliances by using FPGA accelerators. Two key technologies are demonstrated, SR-IOV and PCI Passthrough. Using these two technologies, a single FPGA board can accelerate several virtual software appliances. The advantage of this approach is that FPGAs can very efficiently implement many networking tasks, thus boosting the performance of virtual networking appliances. By taking advantage of SR-IOV and PCI Passthrough technologies, the DMA engine provides transfers rate well above 40 Gb/s for data transmissions from the FPGA to a virtual machine. We have also identified the bottlenecks in the use of virtualized FPGA accelerators caused by reductions in the maximum read request size and maximum payload PCIe parameters.

Cloud network stacks see also greater vertical integration (e.g., merging of historically kernel-level protocol implementations with userspace), and more functionality is "offloaded" to NICs/ switches to move processing closer to data. In particular, software packet switching has been part of operating systems for almost 40 years, but mostly used as a prototyping tool or as a low-cost alternative to hardware-based devices. A number of recent events have brought renewed importance to software switches: the increased importance of virtualization technologies which rely on a back-end software switch to multiplex packets between physical interfaces and containers or VMs; the growing prominence of middleboxes [81, 82] in today's networks along with their virtualization (NFV) [30, 62]; and the widespread use of Software-Defined Networking, for which software switches provide significant flexibility. In the SDN field in particular, a recent trend named SDNv2 and targeting operator networks calls for a model comprising a simple, hardware-based core network providing limited functionality, with complex functionality being pushed into software switches at the edge [64]; a similar call has been made in the context of data centers [6]. Another recent and important trend is the growing popularity of containers, which will require higher port densities in software switches, beyond what current solutions offer. To this aim, we have been working on a modular, scalable and high-performance software switch (i.e. mSwitch, see Section 3.1). mSwitch enables SDN research and experimentation on commodity servers by allowing for easy customization of switching fabrics while yielding production-quality packet rates. This requires simultaneous support for (1) a programmable data plane (beyond what Openflow supports); (2) high throughput; (3) high packet rates; (4) efficient CPU usage; and (5) high port density. These features are difficult to harmonize, and there are, to the best of our knowledge, no available solutions, either as products or research prototypes, that simultaneously provide all of them.

Increasing the hardware/software thoughput and enabling a better communication between virtual machines using a highly scalable software switch represents the first step towards the deployment of high performance and flexible networking appliances. Once the data is available at the CPU it is also important to study problems that arise doing high speed networking on Virtual Machines (see Section 3.4.3).

On the other hand, enabling flexibility at networking stacks in end systems is also a key factor to evaluate and adopt innovative transport protocols that are highly optimized for performances. One recent approach has been to rethink the networking software stack to improve performance while enabling a high level of flexibility. In this context, we proposed Santa (see Section 3.2), an application-agnostic mechanism allowing user-level server applications to offload requests to common replies to a kernel-level cache. By allowing user-space applications to offload frequent requests to the kernel-space, Santa offers drastic performance improvements and unlocks the speed of kernel-space networking for legacy server software without requiring extensive changes. In addition, SSICLOPS is also investigating on other mechanisms to address a number of problems within the operating systems network stack that include the packet I/O subsystem, synchronization and APIs (see Section 3.4.1).

Finally, as the security is also a key concern for intra-cloud dataplanes, and with network traffic rates continuously growing, security systems like firewalls are facing increasing challenges to keep up with line speed without sacrificing protection. In this regard we are currently investigating a hybrid classification system based on tailored circuitry on an FPGA as an accelerator for a Linux netfilter firewall (see Section 3.4.2).

3.1. mSwitch: A Highly-Scalable, Modular Software Switch

The use of software switches has become prominent in a wide variety of systems and scenarios, from standard ones (back-end packet mux/demux for virtualization technologies, replacements for hardware switches) to novel ones such as providing switching planes for consolidated middlebox platforms and software defined networks. This wide range of use cases would greatly benefit from the availability of an extensible software switch that can provide modularity and high performance. Unfortunately, no existing software switches match both requirements. In this paper we present mSwitch, a modular software switch with a strong emphasis on performance, scalability, and extensibility. mSwitch is based on the netmap API and extends the VALE software switch in various ways. Scalability comes by restructuring VALE's internals to support much larger number of ports and increase parallelism among senders. Extensibility is achieved by decoupling the high speed switching fabric from the implementation of the switching logic. The latter can be dynamically reconfigured by loading a kernel module, without having to deal with the complexity of an efficient dataplane implementation. These changes allow mSwitch to scale to hundreds of ports (as opposed to VALE's 64), and dramatically outperform VALE's throughput. In order to show the flexibility of our approach, we use mSwitch to build three distinct modules: a learning bridge consisting of 45 lines of code that outperforms FreeBSD's

bridge by up to 8 times; an accelerated Open vSwitch module requiring small changes to the code and boosting performance by 2.6 to 3 times; a protocol demultiplexer for user-space protocol stacks; and a middlebox multiplexer that direct packets to virtualized middleboxes.

We are actively maintaining mSwitch software, and continually adding new features. Recent ones include polling support for NICs and a new interface that allows switch modules to receive a batch of packets in order for more flexibility and optimization.

Published at the ACM SIGCOMM Symposium on SDN Research 2015 (SOSR'15) [46]; reproduced in appendix A.

3.2. Santa: Faster Packet Delivery for Commonly Wished Replies

Increasing network speeds challenge the packet processing performance of networked systems. This can mainly be attributed to processing overhead caused by the split between the kernel-space network stack and user-space applications. To mitigate this overhead, we propose Santa, an application agnostic kernel-level cache of frequent requests. By allowing user-space applications to offload frequent requests to the kernel-space, Santa offers drastic performance improvements and unlocks the speed of kernel-space networking for legacy server software without requiring extensive changes.

Published at the ACM SIGCOMM 2015 poster session [79]; reproduced in appendix B.

3.3. A PCIe DMA Engine to Support the Virtualization of 40 Gbps FPGA-Accelerated Network Appliances

Network Function Virtualization (NFV) allows creating specialized network appliances out of general-purpose computing equipment (servers, storage, and switches). In this paper we present a PCIe DMA engine that allows boosting the performance of virtual network appliances by using FPGA accelerators. Two key technologies are demonstrated, SR-IOV and PCI Passthrough. Using these two technologies, a single FPGA board can accelerate several virtual software appliances. The final goal is, in an NFV scenario, to substitute conventional Ethernet NICs by networking FPGA boards (such as NetFPGA SUME). The advantage of this approach is that FPGAs can very efficiently implement many networking tasks, thus boosting the performance of virtual networking appliances.

The SR-IOV capable PCIe DMA engine presented in this work, as well as its associated driver, are key elements in achieving this goal of using FPGA networking boards instead of conventional NICs. Both DMA engine and driver will be open source, and target the Xilinx 7-Series and UltraScale PCIe Gen3 endpoint. The design has been tested on a NetFPGA SUME board,

offering transfer rates reaching 50 Gb/s for bulk transmissions. By taking advantage of SR-IOV and PCI Passthrough technologies, our DMA engine provides transfers rate well above 40 Gb/s for data transmissions from the FPGA to a virtual machine. We have also identified the bottlenecks in the use of virtualized FPGA accelerators caused by reductions in the maximum read request size and maximum payload PCIe parameters. Finally, the DMA engine presented in this paper is a very compact design, using just 2% of a Xilinx Virtex-7 XC7VX690T device.

Published at the 2015 International Conference on Reconfigurable Computing and FPGAs (ReConFig 2015) [96]; reproduced in appendix C.

3.4. Work in Progress

This section briefly introduces work which is currently on-going within the SSICLOPS project, but which has not yet been published. This will be reported on in more detail in a future deliverable.

3.4.1. FrankenStack: Redesigning Operating System's Network Stack for Low Latency

A network stack implemented in an operating system (OS) has provided applications with networking services while ensuring protection, isolation and efficient network utilization. However, proliferation of transaction workloads which exchange small packets and frequently setup and terminate TCP connections, has introduced performance problems with it.

Both researchers and practitioners have started blaming a network stack in an OS. They have thus developed network stacks optimized for transaction workloads in user-space almost from scratch, including TCP/IP protocol suite. However, none of these network stacks implements a modern TCP which has been heavily extended since the original "RFC-compliant TCP". This could cause problems with handling various network conditions and attacks.

To improve performance on transaction workloads, we take another path, basing a solution on OS TCP which is a modern, actively maintained and production quality TCP implementation. A key insight is that the OS TCP implementation is only one of the components of the entire network stack, and we observe it takes only 1 μs to process a single receiving packet.

We propose FrankenStack that addresses a number of inefficiencies in the other components of the OS network stack that include packet I/O subsystem, synchronization and socket APIs, while preserving modern, stable Linux TCP. By dedicating a NIC to an application, in the same way as with high-performance user-space network stacks, we bring a number of techniques into a Linux network stack, such as direct packet buffer access, active NIC polling and application-driven network stack execution, to name a few.

On transaction workloads, FrankenStack improves throughput by 8.1% to 139.6% and reduces latency by 7.7% to 58.8% for 64 to 1024 byte serving message sizes and 1 to 1024 concurrent TCP connections.

3.4.2. HyPaFilter – A Versatile Hybrid FPGA Packet Filter

With network traffic rates continuously growing, security systems like firewalls are facing increasing challenges to process incoming packets at line speed without sacrificing protection. Accordingly, specialized hardware firewalls are increasingly used in high-speed environments. Hardware solutions, though, are inherently limited in terms of the complexity of the policies they can implement, often forcing users to choose between throughput and comprehensive analysis. On the contrary, complex rules typically constitute only a small fraction of the rule set. This motivates the combination of massively parallel, yet complexity-limited specialized circuitry with a slower, but semantically powerful software firewall.

The key challenge in such a design arises from the dependencies between classification rules due to their relative priorities within the rule set: complex rules requiring software-based processing may be interleaved at arbitrary positions between those where hardware processing is feasible. We therefore discuss approaches for partitioning and transforming rule sets for hybrid packet processing, and propose HyPaFilter, a hybrid classification system based on tailored circuitry on an FPGA as an accelerator for a Linux netfilter firewall. Our evaluation demonstrates 30-fold performance gains in comparison to software-only processing.

3.4.3. A Study of Speed Mismatches Between Communicating Virtual Machines

Computer systems have many components that need to exchange data and synchronize with each other (i.e. determine when new data can be sent or received). Synchronization can be implicit, e.g. when a piece of hardware has a guaranteed response time, or explicit, requiring asynchronous *notifications* (e.g. interrupts) and/or *polling*, i.e. repeatedly reading memory or I/O registers to figure out when to proceed.

The cost of synchronization can be highly variable, and sometimes even much larger than the data processing costs. For virtual machines, which are the main target of our investigation, the notification cost is of the order of microseconds. Several network I/O frameworks (e.g., DPDK and DNA) rely on polling (or busy-waiting) to remove the cost of asynchronous notifications, at the expense of a very high resource usage.

Modern "paravirtualized" VM devices implement a middle ground between asynchronous notifications and pure polling. In these solutions, the system uses polling under high load conditions, but reverts to asynchronous notifications after some unsuccessful poll cycles. The key problem in these solutions is that strategies to switch from one to another mechanism are

normally not adaptive, and very susceptible to fall into pathological situations where small variations in the speed of one party cause significant throughput changes. In our tests, we have frequently seen systems moving from 100-200 Kpps to 1 Mpps with minuscule changes in operating conditions.

We provide a model of the system explaining how different operating regimes may arise and what kind of impact on performance comes by speed differences, delays and queues. We use this model to derive criteria to select reasonable operating regimes basing on operating parameters, and also understand the impact of erroneous decisions due to incorrect estimates of the parameters.

3.4.4. Packet Reordering and Multi-Core Processing in a Software Switch

Software switches have played an important role in cloud networks and virtualization backends. To minimize packet reordering, they serialize packets in the same flow on the same CPU core or queue, using NIC feature such as receive side scaling (RSS). However, it is not possible to forward packets at rates of 40 and 100 Gbps using a single CPU core depending on packet processing in a software switch. This means, rates of individual flows are limited by the processing speed of a single CPU core. Unfortunately, clock speed of a single CPU core does not get faster anymore. Last but not least, achieving high throughput with a single CPU core would need large queue length for batching, which could increase latency and its variation.

In this work we propose ROAST, an architecture that lets networks reorder packets to some extent in order for multi-core in-flow packet processing, and copes with reordered packets at end systems. ROAST consists of two components: a software switch that processes packets on the most idle queue or CPU core, and a TCP extension to tolerate packet reordering. In addition to high throughput and low latency, ROAST would make latency predictable, because the latency is not affected by flow hash values or load on a specific core/queue, but just by overall load. The ROAST architecture also efficiently utilizes multiple cores for traffic that cannot be balanced based on flows like encrypted traffic.

We extended mSwitch to support active NIC polling and to perform keyword search for every packet, similarly to Deep Packet Inspection. We sent a flow to this mSwitch. To emulate proposing software switch behaviour, we direct packets to different CPU cores or NIC rings by using different UDP source ports on each packet in this flow. When this switch forwards ten 1514 byte packets arriving at 10 Gbps line rate, it took approximately 150 μs using a single CPU core; but using four CPU cores, it took approximately only 30 μs with two packets being reordered. We are currently investigating the impact on TCP performance to develop an algorithm to cope with packet reordering caused by a software switch.

4. Protocol Improvements

Cloud computing typically involves large amounts of communication between the nodes involved in a computation task. Therefore it is crucial that the communication is efficient and does not cause bottlenecks in time-sensitive tasks.

We assume that the Internet Protocol (either IPv4 or IPv6) will remain the core protocol used in cloud systems, as also current deployments show. However, the traditional flavours of TCP have turned out to have challenges in cloud environments due to the specific nature of traffic patterns that are not so common in the general Internet. Particularly, traffic patterns can be bursty and synchronised, as shown by the TCP in-cast problem discussed in earlier work [28]. Improving TCP, or the transport layer in general is therefore a key focus point in making cloud communication more efficient.

Current protocol enhancements for cloud computing are mostly focused on **intra-cloud scenarios**, i.e., solving communication problems within single data centers. Improving protocols for the general Internet has the challenge that protocol improvements need to interoperate with the legacy protocol deployments. This significantly limits the design space of the new improvements. However, because systems within a single datacenter are under common control, deploying new solutions is easier, as all systems can be updated at the same time, and the communication remains inside the datacenter. Therefore solving the problems in intra-datacenter scenarios has more freedom for innovative solutions, as the burden from legacy systems is smaller. The SSICLOPS project has participated in developing the state of the art in data center transport protocols further, partially based on existing solutions.

Datacenter TCP (DCTCP) is a proposal from Microsoft that seems promising for general use in cloud systems. We have therefore chosen DCTCP as one of the starting points for our work in further improving the communication performance in data centres and cloud systems in general. As further development of DCTCP is important also for the SSICLOPS project and our goals, we have participated and contributed to the DCTCP standardisation work in the IETF TCPM (TCP Maintenance and Minor Extensions) working group.

One of the ongoing work items in improving DCTCP is to adapt it for multi-path communication, as multiple disjoint paths are common in cloud systems allowing heavily replicated connections between the nodes (see Section 4.2.5). We except to gain significant improvements in resource utilisation – and consequently communication performance – by doing this.

SSICLOPS also participates in documenting other state-of-the-art TCP implementations in the IETF, such as CUBIC, to increase the awareness of the currently deployed implementation space (see Section 4.2.1).

In addition to improving TCP, intra-cloud communication would allow for more revolutionary, non-backwards compatible protocol improvements. Even though this involves heavier re-design and re-implementation work, it can achieve more significant improvements than are possible by just modifying TCP. Our initial Hoover results are a good indication of this (see Section 4.2.7).

Designing communication for **inter-cloud** scenarios is more challenging because in this case a single administrative entity cannot control the whole communication path, as in the case of intra-cloud communication. The revolutionary non-backwards compatible solutions are therefore more difficult to be taken into use. DCTCP was designed to be applied in intra-cloud scenarios, but it could be possible to adapt similar ideas to be compatible with the Internet and its congestion control principles. One option is to convey more information about the data patterns, to allow the network make smarter decisions in routing the packets inside the cloud, and between the clouds. Reporting the current TCP send buffer size is one potentially effective way of doing this, as we have shown in our recent work (see Section 4.1).

4.1. Oh Flow, Are Thou Happy? TCP Sendbuffer Advertising for Make Benefit of Clouds and Tenants

Datacenter networks have become multipath and this makes their management a difficult task. Tasks such as traffic engineering, finding hotspots and per-flow performance debugging are unnecessarily difficult because the network lacks sufficient information about the traffic it routes. This results in inefficient traffic allocations and wasted expert time for network debugging.

In this paper we propose a simple change to TCP to address these issues: all senders should encode in their packets the amount of data they have buffered but not sent on the wire yet. We call this technique “sendbuffer advertising” and propose an implementation that has zero bandwidth overhead. We have explored a varied range of use cases, finding that sendbuffer advertising unlocks optimizations that are not possible without it, and that it benefits the network and traffic receivers in multiple scenarios.

Sendbuffer advertising is particularly beneficial for cloud environments were the cloud provider has currently no means to understand what its tenant apps need. While changing tenant apps to provide such information is infeasible, we believe incentivising tenants to report sendbuffer usage is possible.

Published at the 7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud ’15) [1]; reproduced in appendix D.

4.2. Work in Progress

This section briefly introduces work which is currently on-going within the SSICLOPS project, but which has not yet been published. This will be reported on in more detail in a future deliverable.

4.2.1. CUBIC for Fast Long-Distance Networks

The default congestion control scheme in Linux — which provides the operating system basis for most open-source-based cloud infrastructures — years ago changed from the IETF standard NewReno algorithm to CUBIC. CUBIC uses a cubic function (hence the name) instead of the linear window increase of the current IETF TCP standards to improve scalability and stability under fast and long distance networks.

CUBIC was originally described in a 2008 academic paper by researchers from NCSU [43], but Linux quickly diverged from this basis, as improvements and bug fixes were committed to the code base.

The result is that the highly optimized CUBIC congestion control mechanism, which is responsible for controlling the transmission of a large fraction of traffic on the Internet and in datacenters, is not openly specified for implementation. Implementing a modern variant of CUBIC requires studying GPL code in the Linux kernel, which makes it difficult for developers who have done so to re-implement the scheme in non-GPL code. This means that cloud infrastructures that are not based on Linux have difficulties benefiting from the advantages that CUBIC can bring.

In order to address this situation, the IETF has begun to document the current Linux implementation of CUBIC in an Internet Draft [77], from which other inter-operable implementations can be freely derived, without the need to study GPL code, which can taint other code bases.

4.2.2. Using the TCP Echo Option for Spurious Retransmission Detection

A spurious retransmission in TCP happens when a sender incorrectly determines that a previously transmitted segment has been lost. This can happen for a number of reasons, such as a delay spike or packet reordering along the path, among others. After determining that packet loss occurred, the sender reacts by reducing the sending rate and retransmitting the apparently lost segment. If the loss event was incorrectly determined, performance of the TCP connection suffers, both because the sending rate was reduced without real need, and because the packet retransmission used capacity that would otherwise have been used for new data. Avoiding, or at least quickly detecting and reacting to spurious retransmissions, has been an ongoing topic in the development of TCP at least as far back as TCP Eifel [60] in 2003.

SSICLOPS recently made a new proposal to the IETF for detecting spurious retransmissions [98], based on the TCP Echo Option [99], which is another new proposal introducing a generic echo option mechanism to the TCP protocol.

4.2.3. UDP Usage Guidelines

The User Datagram Protocol (UDP) [73] provides a minimal message-passing transport that has no inherent congestion control mechanisms. In 2008, the IETF published RFC 5405 [36], a "Best Current Practice" (BCP) document, in order to provide guidelines to designers of unicast applications and upper-layer protocols using UDP. Congestion control guidelines were a primary focus, but the document also provided guidance on other topics, including message sizes, reliability, checksums, and middlebox traversal.

Since the publication of RFC 5405, UDP has become much more widely used as a tunneling mechanism, both in the Internet as well as within datacenters, as well as being used as the transport for Google's proposal for a successor to HTTP, called QUIC [50].

Therefore, the IETF Transport Area Working Group has identified additional areas where guidance is needed and began work on a successor to RFC 5405, which is being developed in an Internet Draft [37]. This new document provides guidelines on the use of UDP for the designers of applications, tunnels and other protocols. Congestion control guidelines are still one focus, but the document also provides guidance on other topics, including UDP multicast, message sizes, reliability, checksums, middlebox traversal, the use of Explicit Congestion Notification (ECN), Differentiated Services Code Points (DSCPs), and ports. Some guidance is also applicable to the design of other protocols (e.g., protocols layered directly on IP or via IP-based tunnels), especially when these protocols do not themselves provide congestion control.

4.2.4. TCP-aNCR: Facing Reordering with Agility

Packet reordering in the network causes some packets to arrive late compared to normal transmission. TCP assumes packet loss due to congestion and then responds by unnecessarily reducing the sending rate. Therefore, reordering in the network can cause major performance issues and throughput degradation to TCP.

The most common method of preventing the negative impact is to delay the retransmission and the accompanied congestion response until it can be decided if the packet was late due to reordering or really lost. This avoids the main source of performance degradation in terms of throughput. However, delaying retransmissions has a major drawback for applications that do not tolerate delay spikes, such as interactive streams or storage traffic: TCP hands data to the application only in-order. If a segment is missing TCP waits for its reception before giving more data to the application. Delaying retransmissions hence causes an equal delay in data reception for the receiving application. Therefore, it is important to keep the delay low while avoiding throughput drawbacks.

From measurements in live mobile networks we conclude, that the reordering is heavily dependent on the sending rate. Hence, when facing reordering in variable environments a reordering algorithm should react to changing conditions. Only then can it prevent negative impact and spurious retransmissions while still keeping the delay low. Adaptive non-congestion robustness for

TCP (TCP-aNCR) exploits the sending rate dependent reordering extent to set the retransmission delay in a way that automatically adapts according to the current sending rate. This achieves robustness to packet reordering while still resulting in low delay under varying environments.

4.2.5. Multipath Data Center TCP

Both multipath transport and delay-reducing transport are current trends in the development of data centre protocols, as shown also by the amount of related work referenced in this deliverable. MPTCP has been shown to increase connection throughput in data centers while DCTCP and similar protocols based on congestion notification and Active Queue Management tackle the problem of queueing delay caused by the large standing queues resulting from the use of TCP.

Since MPTCP uses TCP as a protocol for its individual subflows, the changes DCTCP applies to standard single-path TCP should in principle be applicable in a similar fashion to the individual subflows of MPTCP, thus creating a combination of the two approaches with potentially the benefits of both.

While both protocols by themselves are still being investigated and improved upon, particularly DCTCP, an investigation of such a combination cannot be based on a stable suite of protocols. However, it has been shown that the basic idea is sound and thus further investigation is promising [23]. In the context of SSICLOPS, a combination of DCTCP and MPTCP has been initially evaluated, showing that already with minor changes to the protocol base queue lengths can be reduced significantly for multipath flows. However, the approach remains to be optimized (in particular w.r.t. fairness between subflows) and compared to related approaches.

More importantly, a relevant research question to be explored based on this initial protocol design is how feasible the deployment of such a combination is in practice. In particular, deployability of such adapted protocols in data centres is typically assumed to be given due to the level of control over the end hosts. However, since the inter-data centre as well as the client-facing transport are important use-cases within SSICLOPS, the resulting necessity to accommodate standard protocols such as (MP)TCP in parallel to specialised ones needs to be addressed in more detail.

Thus, the effect of the co-existence of standard transport protocols and adapted ones in data centres as well as the mechanisms necessary to enable it will be investigated further. Here, a proxy approach such as described in the following section could be exploited as well, i.e., a proxy enabling the use of adapted protocols without having to deploy changes in the client/remote network stack.

4.2.6. MPTCP Proxy

MultiPath TCP (MPTCP) [41] is a protocol extension to exploit available multiple paths between two end-points by multiplexing data via different regular TCP connections (subflows) with some

new TCP options to mark the multiplexed data. This accumulation of different paths provides not only an increased bandwidth capacity but also increased robustness. From an application's point of view, there is only one standard TCP connection to its communication peer and there is no need to know about MPTCP. The MPTCP-aware network stacks at the end-points handle the multipath complexity on the transport layer and provide a single, albeit logical, connection to the upper layer. The respective endpoints' policies determine how a path manager initiates and controls the different subflows (e.g., full mesh, based on different addresses or ports) as well as the algorithm used by the scheduler to carry out data multiplexing (e.g., round-robin, lowest RTT first). Further, an adapted congestion control algorithm [74] should reflect the resource pooling property by coupling the congestion control loops for the different subflows.

Not every client or server runs an MPTCP-aware network stack and hence, in order to benefit from MPTCP, a proxy can be placed in front of an endpoint converting between TCP and MPTCP. While a transparent http proxy using MPTCP is easily put in place (e.g., polipo) it does not serve https well (man-in-the-middle attack). SSICLOPS is currently implementing an MPTCP proxy which does not terminate the transport connection but converts transparently from TCP to MPTCP and vice versa. This kind of non-terminating transparent proxy for protocol conversion is not new and was described in [44] for fast MPTCP handover scenarios in wireless networks and in [33] which also translates TCP to MPTCP but requires an additional TCP extension for traffic redirection similar to SOCKS. A more general splitter/combiner proxy architecture for regular TCP was reported in [9].

The MPTCP proxy is being implemented as a tranparent and light-weight proxy running in user-space intercepting packets directly from the network interface and avoiding the kernel network stack. Regular TCP sequence numbers are converted to MPTCP sequence numbers while the TCP subflows use independent sequence numbers. Buffers and state information are kept to as little as possible. Deploying the MPTCP proxy instead of using an MPTCP-aware network stack means that the above mentioned MPTCP path manager and scheduler are moved away from the end-point while the congestion control remains at the end-point (which might have an impact). As the MPTCP proxy could be placed in front of both the client and the server there are different requirements regarding scaling and policy configuration for the path manager and scheduler. On the server side, the cooperation of MPTCP with load balancers as already outlined in [71] can be further explored. Also, the planned introduction of a proxy flag as drafted in [90] can be tried with the MPTCP proxy. On the client side, the proxy has to serve only a comparatively small number of connections but how to install the proxy here is not decided yet. We plan to carry out measurements for different scenarios. Interoperability is tested against the MPTCP reference Linux implementation.

4.2.7. Hoover

Modern datacenter networks provide very high capacity and low switch latency, but transport protocols rarely manage to deliver performance matching the underlying hardware. Hoover is a novel datacenter transport architecture that achieves both near-optimal file completion times and

throughput in a wide range of scenarios including incast. Hoover achieves this through a novel and synergistic combination of techniques including packet spraying, extremely small switch buffers, a randomized variant of packet trimming when queues fill, a priority queuing mechanism and a highly tuned receiver-driven ACK clocking mechanism that can trigger retransmissions so rapidly that lost packets have very little impact on performance.

The rise of large datacenters has been accompanied by a growing realization of the limitations of transport protocols designed for the Internet in such environments. Protocols such as DCTCP optimize existing approaches to mitigate TCP's slow timeout and congestion control dynamics and Multipath TCP [75] enables more efficient use of parallel paths. More radically, CP [29] trims packets back to just their headers at congested links to provide more information about the congestion state of the network. These approaches have their merits but none manages to pull all of the pieces required together so as to achieve both extremely low delay and high throughput. Hoover meets these goals with a three-pronged approach: we need to keep queues short while not losing time with retransmissions, we need a way to use all the capacity the underlying topology makes available in the "core", and we need a way to deal with incast at access links.

To fully utilize available capacity so that hot spots are not created, we would like to perform per-packet ECMP, also known as *packet spraying* [34]. In a Clos topology such as FatTree [39], packet spraying ensures traffic cannot concentrate on any subset of the paths, except as traffic fans in to the receiver. This allows high throughput, but the challenge is also to cope with asymmetric paths that may appear due to failures or congestion hotspots, as well as to minimize transfer times for the small flows that dominate in datacenters.

The *incast* problem appears when many flows converge on one receiver. Others have worked on probing-based approaches to avoid incast; we take a much more radical approach: the sender should always send the first RTT of data at line rate, to ensure short flows fully utilize the capacity. Instead of avoiding it, we design mechanisms to detect it in minimal time and retransmit as fast as possible. The key observation is that retransmissions do not cost bottleneck bandwidth; if we can minimize their latency cost, then it's better to optimize retransmissions instead of trying too hard to avoid them. Control messages can traverse the network in less than one data packet serialization time, so long as they don't encounter queues. With careful protocol design we can take advantage of this low delay so that retransmissions don't significantly hurt flow completion time.

The innovation required is packet trimming (aka Cut Packet [29]): when a switch queue fills, the switch does not drop the packet, but rather trims off the payload leaving only the header. For typical 9 KByte datacenter Ethernet packets, trimming to just a 60 Byte header provides 150:1 compression, allowing 150 headers to be forwarded in the time that a single data packet would take. In our experiments we operate switches with eight packet buffers and can still achieve greater than 95% of full bisection bandwidth.

The final and most crucial element of Hoover is the data clocking algorithm run between a receiver and the senders sending to it. A Hoover sender will only burst at line-rate for one RTT

– typically ten packets in a Fat Tree topology. After this it waits for the receiver to tell it what to do next, because only the receiver knows the instantaneous demand. By giving up on sender congestion control, we are also robust to asymmetric topologies. We believe that Hoover is the first fully distributed datacenter protocol that achieves near-perfect utilization and near-optimal flow completion times in a wide range of scenarios.

5. Workload Scheduling Improvements

Scheduling lies at the heart of both public and private clouds. Datacenters contain tremendous amounts of compute resources, which come at a very high cost. They rely on the principle of resource pooling to utilize the infrastructure in an efficient manner and to get the best possible performance. Seeing the cloud as an elastic resource that can easily respond and adjust to user demand is already an established paradigm, but there still is room for improvement.

While getting better performance is probably what first comes to mind when thinking about the goals of cloud computing (and computing in general), there is no recipe that can be tailored to every scenario, nor a way around all the trade-offs that invariably appear. Moreover, for public clouds, tenant desires can be at odds with provider policies. In this setting, any mechanism that fosters reconciliation and cooperation is welcome, as otherwise everyone has something to lose. The need for workload scheduling is widespread among the various parts of the SSICLOPS project, as it appears on multiple levels and in a significant number of scenarios.

At the lowest level that is of interest to us, we find workload scheduling within a single distributed application. This is important in order to achieve the goal of flexible scaling; proper scheduling is needed to get the most out of the hardware infrastructure. There are cases where an application that was not traditionally used in a distributed manner can have a lot to benefit from a scale-out approach. The transition from running on a single machine to a cluster or even more is difficult, and highly dependent on the particular application; being able to efficiently partition and parallelize the workload ensures that horizontal scaling is actually an attractive option.

An example of parallelizing and then efficiently using scheduling to significantly improve performance and reliability is the work on Hyrise-R, a replication system for the in-memory Hyrise database (see Section 5.1). Hyrise is used for enterprise applications, which require the ability to run both analytical and transactional queries. Traditionally, databases like Hyrise were scaled vertically when higher demands became apparent, but this cannot really go beyond a certain point. Instead, using a cluster of servers is comparatively cheaper, more resilient and also provides a great deal of flexibility. Hyrise-R builds on the fact that an overwhelming amount of queries for both analytical and transactional processing are reads, which are easily parallelizable. This is done by employing an arbitrary number of replicas beside a master node. Since reads do not endanger the consistency of the system, no synchronization is required for most of the queries, and they can easily be processed by any node. Synchronization is required for writes and it's handled by using lazy replication, which happens asynchronously. The delay caused by the Hyrise-R implementation of lazy replication is not an issue for most OLAP applications.

The clustering aspect is transparent to the user; a central dispatcher is responsile for scheduling queries among the available nodes. Hyries-R also implements a failover mechanism which increases availability based on the newfound redundancy caused by replication.

Improvements are already being done to the Hyrise-R dispatcher, and this is a work in progress mentioned in Section 5.3.1. The original behaviour was to schedule the query workload in a round-robin fashion, but this is not optimal for asymmetric clusters, and cannot provide throughput guarantees when they are required. The new dispatcher implementation gives priority to write queries (and thus can guarantee a minimum write throughput), and also allows different nodes to accept and respond to queries at their own pace.

When building a system designed for efficient scaling, more information may be required as input from the running environment, but this doesn't necessarily have to be explicitly handled by applications. It is well known that as computers start to drift apart from the theoretical model of an abstract machine, it becomes more important to find and utilize the best instruments available in a particular scenario. For example, when running over a NUMA system, we have to take into account fundamental parameters such as locality, the nature of the interconnect and access latencies. The same mindset carries over to even more heterogenous environments such as clusters, and (federated) cloud architectures. The design of an application can take all this into account, but it leads to a long development process which may not even be feasible in many scenarios. One solution is to provide a programming model that abstracts away most of the details and provides an interface both easy to utilize and powerful enough to cover a wide range of use cases. The work on Claud reported on in Section 5.2 represents a solution for scheduling workloads that can be scattered at data centre scale or even across different datacenters. It enables developers to benefit from flexible placement strategies which can be applied to a wide range of operating environments. Claud offers a coherent programming model based on an extensible set of distribution modules, which together define the coordination of data and tasks, and also shape the interface which is ultimately exposed to the user. For example, one possible way of interacting with Claud is to see a shared-memory abstraction and to implicitly create tasks using parallel loops or recursive calls. The inner workings of Claud are further illustrated by the example of a Barnes-Hut N-Body algorithm, where tasks and data handling are described in detail.

An issue that invariably arises after the transition from a local hardware infrastructure to a public cloud is the loss of information, especially related to the network which connects virtual servers. If resources such as memory, CPU or storage are described to the cloud customer in clear terms, the network is generally presented in rather opaque descriptions. For example, a virtual server has a specific amount of RAM and storage capacity, but the network connection is described in vague terms like "moderate" or "high". A tenant thus deploys applications in an unfamiliar environment, where important assumptions may no longer hold. On the other side of the cloud, most of the optimizations that providers employ to increase efficiency are based on heuristics and perceived usage patterns. This is to be expected as the intent of each customer is unknown to the provider. This virtual stand-off can be avoided, and we attempt to do this with

the work on Cloudtalk, briefly presented in Section 5.3.4. Cloudtalk is a middleware that stands between tenants and the cloud provider. It provides a language used to describe the workload of a particular tenant, and it can serve multiple purposes. First of all, the tenant can receive assistance from the provider in placing tasks on servers that are best suited for the job. Secondly, the provider can make better informed decisions based on the intent of each tenant. Finally, in federated clouds, the provider can influence the placement of customer tasks in a way which attempts to keep undesirable events such as inter-DC transfers to a minimum.

Cloud tenants generally care about application performance and platform reliability, whereas providers worry about efficiency and other aspects that influence operational costs behind the scenes. A big concern in this regard is energy efficiency: providers attempt to consolidate tenant workloads on physical servers in a way that minimizes consumption. This cannot be done indiscriminately because different tenants have different SLAs, so what generally happens is that virtual machines are shuffled around the datacenter via migration. An interesting statistical effect is the fact that, at any given time, a significant number of VMs are actually idle. The composition of this set of machines is continuously changing, but as long as they remain virtually idle, a large number of them can be placed on a relatively small number of servers. The work on energy efficiency of dynamic management of a virtual cluster with heterogeneous hardware described in Section 5.3.2 takes this a step further and relies on specialized, energy-efficient servers to store idle virtual machines. Initial testing shows potential savings of up to 48% and improvements in terms of energy efficiency for large installations of up to 40% (based on simulations). This kind of scheduling for workloads which become temporarily idle may prove to significantly increase VM density without compromising the overall level of service.

Finally, an interesting case of workload scheduling at the federated cloud level, and possibly beyond, is presented in Section 5.3.3. The workload consists of an ad-hoc network of ephemeral virtual CDN servers. A simulation framework is built to estimate the effects of a large-scale ephemeral CDN deployment, and data from a large video hosting service is used as a point of reference for the experiments.

5.1. Hyrise-R: Scale-out and Hot-Standby through Lazy Master Replication for Enterprise Applications

In-memory database systems are well-suited for enterprise workloads, consisting of transactional and analytical queries. A growing number of users and an increasing demand for enterprise applications can saturate or even overload single-node database systems at peak times. Better performance can be achieved by improving a single machine's hardware but it is often cheaper and more practicable to follow a scale-out approach and replicate data by using additional machines.

In this paper we present Hyrise-R, a lazy master replication system for the in-memory database Hyrise. By setting up a snapshot-based Hyrise cluster, we increase both performance by

distributing queries over multiple instances and availability by utilizing the redundancy of the cluster structure. This paper describes the architecture of Hyrise-R and details of the implemented replication mechanisms. We set up Hyrise-R on instances of Amazon's Elastic Compute Cloud and present a detailed performance evaluation of our system, including a linear query throughput increase for enterprise workloads.

Published at the 3rd VLDB Workshop on In-Memory Data Mangement and Analytics (IMDM 2015) [80]; reproduced in appendix E.

5.2. Claud: Coordination, Locality And Universal Distribution

Due to the increasing heterogeneity of parallel and distributed systems, coordination of data (placement) and tasks (scheduling) becomes increasingly complex. Many traditional solutions are not taking into account the details of modern system topologies and consequently experience unacceptable performance penalties with modern hierarchical interconnect technologies and memory architectures. Others offload the coordination to the programmer by requiring explicit information about thread and data creation and placement. While allowing full control of the system, explicit coordination severely decreases programming productivity and disallows implementing best practices in a reusable layer.

In this paper we introduce Claud, a locality-preserving latency-aware hierarchical object space. Claud is based on the understanding that productivity-oriented programmers prefer simple programming constructs for data access (like key-value stores) and task coordination (like parallel loops). Instead of providing explicit facilities for coordination, our approach places and moves data and tasks implicitly based on a detailed topology model of the system relying on best performance practices like hierarchical task queues, concurrent data structures, and similarity-based placement.

Published at the International Conference on Parallel Computing 2015 (ParCo 2015) [11]; reproduced in appendix F.

5.3. Work in Progress

This section briefly introduces work which is currently on-going within the SSICLOPS project, but which has not yet been published. This will be reported on in more detail in a future deliverable.

5.3.1. A Scalable Query Dispatcher for Hyrise-R

Section 5.1 introduced Hyrise-R, a scale-out and hot-standby extension of the in-memory database Hyrise. The Hyrise-R cluster consists of one master database instance and an arbitrary number of replica instances. Users send their requests to the dispatcher. Write requests are

forwarded to the master node. Read requests are distributed among all cluster instances in a round-robin manner. The master node sends log entries to the replicas to keep them up-to-date. Therefore, we added an interface for the log exchange to the Hyrise core. We implemented a heartbeat protocol to detect node failures.

The communication between the Hyrise-R cluster instances was the focus of work so far. The dispatcher distributes queries in a round-robin manner with no prioritization of writes. During write intensive workloads, reads are still sent to the master node, which results in a heavier load compared to the replicas. To overcome this imbalance, we work on an extensible query dispatcher. Besides prioritizing writes on the master node, advanced query distribution algorithms can exploit knowledge about the cluster instances, e.g., existing indices to forward queries to the most suitable node.

5.3.2. Energy Efficiency of Dynamic Management of Virtual Cluster with Heterogeneous Hardware

The variation among cloud computing tasks, both in their resource requirements and time of processing, makes it possible to optimize the usage of physical hardware by applying migration technologies. For this purpose, we developed a methodology and prototype system for load based management of virtual machines in an OpenStack cluster. The method is based on an idea of "packing" idle virtual machines into special park servers optimized for this purpose. The method was evaluated by running real high-energy physics analysis workload in an OpenStack cluster and additionally by simulating the same principle using the Cloudsim simulator software. The results show a clear improvement, 9% to 48%, in the total energy efficiency when applying the method together with resource overbooking and heterogeneous hardware.

Existing cluster management systems are usually based on heuristics using data gathered from both physical and virtual machines. In their simplest form, these systems make decisions based on the load of physical CPUs, while the complex ones take into consideration additional aspects such as memory usage, network traffic, service level agreements (SLA), server energy consumption, server thermal state, virtual machine intercommunication etc. [12]. The basic idea of all of these management systems is to maximize the usage of active servers while minimizing the amount of them. One way to achieve this is to pack already active servers more efficiently. However, OpenStack, like many other open source systems, does mainly passive management in which load balancing is done at the time of creation of new virtual machines. Thus, these algorithms are not able to react to load changes of running VMs. Instead, active cloud management systems can better react to changing load patterns as they actively move the VMs among physical machines.

Heterogeneous hardware in cloud environment has been studied, but little from the load balancing perspective. Hirofuchi et al. [45] use shared servers for VMs with less load, and dedicated servers for running heavily loaded VMs. Virtual machines are then moved between these two types of hardware as a function of their processing needs: a shared node stores idle virtual machines and

dedicated nodes run active virtual machines. When a shared server CPU load exceeds 90% the most active virtual machine is moved to a dedicated server, and when the CPU load of a VM is less than 50%, it is returned to the shared node. The only difference between shared server and dedicated server is the amount of memory.

The novel idea of our load balancer is to use specialized, energy-efficient servers ("park servers") for storing idle virtual machines in addition to active monitoring of resources and load based active management of VMs in the physical cluster. The park server is used to store idle VMs and its over-commit ratio is high. Actual computing nodes are used by active VMs and they have a one-to-one mapping of virtual to physical resources. The load balance manager migrates VMs using three functions: 1) move idle VMs to the park server; 2) move a VM from the park server to a computing node when it becomes active; 3) consolidate active VMs within the cluster to minimize the number of active servers. In this way the, VMs that require more capacity can be served using dedicated hardware and VMs that are waiting for work can be packed more densely.

Our study demonstrates that the dynamic workload management with heterogeneous hardware works well using basic heuristics. Our tests showed a potential energy saving of 9% to 48%. Additionally, simulation results showed that in large installations the energy efficiency could be improved by up to 40%. We assume that using a more sophisticated heuristics the results could be clearly better. Also, the hardware setup in our tests was limited. Thus, the next step would be to us a test system in a larger cloud environment.

5.3.3. Simulating CDNs Spanning Thousands of ASes

In this section we make the case for ephemeral CDNs, the ability to build virtual CDNs *on-the-fly* on top of shared, third-party infrastructure. To bring this idea closer to reality we perform large-scale simulations to quantify the effects that ephemeral CDNs would have if deployed and compare these to traditional CDN deployments.

We implement a custom, flow-level CDN simulator from scratch consisting of about 2,000 lines of Python code. At the highest level, the simulator uses the Internet's AS-level topology obtained from the Internet Research Lab (IRL) [49] to build its network graph (48,991 ASes in total). We further use the IRL's data to assign IP address prefixes to ASes, and use longest prefix matching for forwarding packets. In addition, we take advantage of CAIDA's AS ranking [25] to distinguish between content, transit or access ASes [59].

In terms of link speed, we assign a capacity of 40 Gb/s to inter-AS transit links, 10 Gb/s to leaf-AS links, and 25 Mb/s for end user connections (25 Mb/s is the amount recommended by Netflix for 4K video [67]). Video streams are consumed at the rate defined by their video quality: 360p – 1 Mbps, 480p – 2.5 Mbps, 720p – 5 Mbps, 1080p – 8 Mbps, 2K – 10 Mbps and 4K – 20 Mbps [86].

We simulate 200 channels (e.g., the work in [26] cites 150 for a large IPTV system) and apply video quality rates from the range above using a Poisson distribution. For the requests, we use a Zipf distribution, a typical distribution for content popularity, to decide which content/channel they should retrieve. Regarding viewing duration, the authors of [26] describe it as ranging from 1 minute to an hour. Based on this, we use a triangular distribution within that range peaking at 30 minutes (e.g., the duration of a sitcom series). Requests arriving at an access AS that does not have a virtualized content cache are directed to an origin server; in parallel, we instantiate a cache to handle any subsequent requests arriving at that AS.

For each stream request, the simulator keeps track of a number of standard video QoE metrics: (1) *start time*, the time between the video player initiating a request and the moment its buffer is filled up so that it can start playback; (2) *buffering ratio*, the fraction of total session time spent buffering; (3) *buffering event rate*, essentially the number of interruptions that the user experiences throughout the lifetime of the stream; and (4) *playback ratio*, the ratio of download rate to the video code's rate (1 being the optimal).

In order to keep simulation time reasonable and still produce meaningful results, we restrict our simulations to a subset of the overall AS topology containing 1,187 ASes, 2,880 links between them; this is comparable to a large country. Of these, 239 are content provider ASes, meaning that origin servers can be placed in them, and 871 are access provider or "eyeball" ASes – potential sites for content cache deployment.

To show the effects of ephemeral CDNs at large scale, we simulate one million users with at least 100,000 concurrent connections at any one point in time and a peak of 160,000 connections. Streams are generated randomly, with an average rate of 6,670 requests per second. As a point of reference, Rutube, one of the largest video-hosting services in Russia claims to serve up to 65,000 concurrent live streams [78]. We thus believe the simulations should be able to capture large-scale effects.

5.3.4. Opening Up Black Box Networks with Cloudtalk

Many organizations are running large-scale distributed applications in the cloud, benefiting from the abundance of resources and the attractive pay-per-use billing model. Cloud users want their apps to finish as quickly as possible, and they should be able to use the many optimizations that have been proposed to this end. Optimizations often require detailed knowledge of the cloud topology and its resources. In particular, network topology and traffic information is crucial to getting good performance as the network is often the bottleneck; this holds not only in oversubscribed topologies, but even in fully provisioned ones where bottlenecks form at the end-host NICs in many-to-one traffic patterns. Meanwhile, the data centre operators keep topology information confidential, and only optimize placement at virtual machine startup, based on the chosen instance type.

This status quo is suboptimal: application optimizations either ignore the network or profile it continuously to get accurate information. The former results in disgruntled customers and

low network utilization and the latter wastes traffic. Cloud operators can only do network-only optimizations such as load-balancing traffic between available paths; the endpoints of the traffic cannot be moved efficiently by the network because virtual machine migration is too heavyweight to be used on short timescales. We are seemingly stuck: optimizing distributed applications requires in-depth topology knowledge and realtime traffic information that providers are understandably reluctant to give away.

We propose Cloudtalk, a query language that tenants can use to query the cloud network to find the best way to perform a certain task. Cloudtalk leverages the fact that most distributed applications have multiple choices for the source or destination of a transfer: a computation could be run on any of the available instances, or a filesystem read could be serviced by any available replica. When such choices exist, the tenant apps use Cloudtalk to query the cloud operator on what the preferred action would be. The network responds by sorting the alternatives according to expected performance, and the tenants use this ordering to select the best way of performing an action.

So far, Cloudtalk has been implemented in two distinct ways. First, we implemented it as a centralized server that has accurate topology information and can answer long-term queries (not based on existing traffic patterns) by using packet-level simulation. This solution works great for a web-search style application where buffer sizing and base network RTTs are more relevant than instantaneous bandwidth availability. Secondly, we have implemented a distributed version of Cloudtalk that runs in hypervisors and actively monitors the bandwidth availability on any given host. This solution scales to arbitrarily large datacenters and can cope with varying traffic patterns. We have modified the Hadoop and HDFS to utilize Cloudtalk, and ran experiments both on a local cluster and on Amazon EC2. Our results show that it can deliver up to 30% better file read times for HDFS and 80% better write performance. We also show how it can be used to deal with networks where capacity is not uniform across all hosts, and for clouds that span multiple geographic locations.

6. Conclusions

This document constitutes deliverable D1.1 of the SSICLOPS project. It is jointly produced by work packages 1 and 3 focusing on cloud infrastructure within single data centers and across multiple federated data centers. It describes the results achieved by these two work packages during the first year of the SSICLOPS project.

The SSICLOPS project has developed and investigated a number of ideas addressing challenges in various parts of the end-to-end service delivery infrastructure. More specifically, SSICLOPS worked on the networking stack on endpoints, the protocols spoken between the endpoints, and the distribution of workload across the infrastructure.

SSICLOPS has been developing improvements for all three categories and thus shows broad coverage of the end-to-end cloud infrastructure. In these domains, SSICLOPS has published six papers so far documenting significant improvements in throughput and latency. Additionally, partners of the project consortium have contributed to five IETF drafts. One piece of software has been open sourced and released to the community. Accordingly, the project has made good progress towards its objectives of providing improvements of private individual as well as federated cloud infrastructure.

Multiple further ideas have been developed inside the SSICLOPS project and are currently being explored. These are briefly introduced in this deliverable as well. The project partners will continue to work on these ideas and more details will be presented in the next joint WP1/WP3 deliverable at the end of project year two. The project aims at releasing more software as open source, producing further ideas, and investigating their impact.

Creating, testing, and implementing new ideas is one thing, another thing is to put them into practice. WP1 and WP3 of the SSICLOPS project will evaluate the developed mechanisms in real uses cases. Such use cases have been analysed in WP4 of the project and collaboration between the three work packages will produce results relevant to improving the performance of the cloud applications looked at in the SSICLOPS project. Evaluations of using the proposed mechanisms in the use cases will show the impact and the benefits they have on specific applications.

Bibliography

[1] A. Agache and C. Raiciu. "Oh Flow, Are Thou Happy? TCP Sendbuffer Advertising for Make Benefit of Clouds and Tenants". *7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 15)*. USENIX Association, July 2015. URL: https://www.usenix.org/conference/hotcloud15/workshop-program/presentation/agache.

[2] J. Ahn, C. Kim, J. Han, Y.-R. Choi, and J. Huh. "Dynamic Virtual Machine Scheduling in Clouds for Architectural Shared Resources". *Proc. 4th USENIX Conference on Hot Topics in Cloud Ccomputing*. HotCloud'12. USENIX Association, 2012, pp. 19–19. URL: http://dl.acm.org/citation.cfm?id=2342763.2342782.

[3] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. "Data Center TCP (DCTCP)". *Proc. ACM SIGCOMM 2010 Conference*. SIGCOMM '10. ACM, 2010, pp. 63–74. DOI: 10.1145/1851182.1851192.

[4] M. Alizadeh, A. Javanmard, and B. Prabhakar. "Analysis of DCTCP: Stability, Convergence, and Fairness". *Proc. ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*. SIGMETRICS '11. ACM, 2011, pp. 73–84. DOI: 10.1145/1993744.1993753.

[5] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda. "Less is More: Trading a Little Bandwidth for Ultra-low Latency in the Data Center". *Proc. 9th USENIX Conference on Networked Systems Design and Implementation*. NSDI'12. USENIX Association, 2012, pp. 19–19. URL: http://dl.acm.org/citation.cfm?id=2228298.2228324.

[6] M. Alizadeh, S. Yang, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker. "Deconstructing Datacenter Packet Transport". *Proc. 11th ACM Workshop on Hot Topics in Networks*. HotNets-XI. ACM, 2012, pp. 133–138. DOI: 10.1145/2390231.2390254.

[7] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker. "pFabric: Minimal Near-optimal Datacenter Transport". *Proc. ACM SIGCOMM 2013 Conference on SIGCOMM*. SIGCOMM '13. ACM, 2013, pp. 435–446. DOI: 10.1145/2486001.2486031.

[8] J. Antony, P. P. Janes, and A. P. Rendell. "Exploring thread and memory placement on NUMA architectures: Solaris and Linux, UltraSPARC/FirePlane and Opteron/HyperTransport". *High Performance Computing-HiPC 2006*. Springer, 2006, pp. 338–352.

[9] T. Ayar, B. Rathke, L. Budzisz, and A. Wolisz. "TCP over multiple paths revisited: Towards transparent proxy solutions". *Communications (ICC), 2012 IEEE International Conference on*. June 2012, pp. 109–114. DOI: 10.1109/ICC.2012.6363972.

[10] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron. "Towards Predictable Datacenter Networks". *Proc. ACM SIGCOMM 2011 Conference*. SIGCOMM '11. ACM, 2011, pp. 242–253. DOI: 10.1145/2018436.2018465.

[11] J. Beilharz, F. Feinbube, F. Eberhardt, M. Plauth, and A. Polze. "Claud: Coordination, Locality And Universal Distribution". (to appear). IOS Press, 2015.

[12] A. Beloglazov and R. Buyya. "Energy Efficient Allocation of Virtual Machines in Cloud Data Centers". *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*. 2010, pp. 577–578. DOI: 10.1109/CCGRID.2010.45.

[13] A. Beloglazov, J. Abawajy, and R. Buyya. "Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing". *Future Generation Computer Systems* 28.5 (2012). Special Section: Energy efficiency in large-scale distributed systems, pp. 755–768. DOI: http://dx.doi.org/10.1016/j.future.2011.04.017.

[14] A. Beloglazov and R. Buyya. "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in Cloud data centers". *Concurrency and Computation: Practice and Experience* 24.13 (2012), pp. 1397–1420. DOI: 10.1002/cpe.1867.

[15] S. Bensley, L. Eggert, D. Thaler, P. Balasubramanian, and G. Judd. *Datacenter TCP (DCTCP): TCP Congestion Control for Datacenters*. Internet-Draft draft-ietf-tcpm-dctcp-01. Work in Progress. Internet Engineering Task Force, Nov. 1, 2015. URL: https://tools.ietf.org/html/draft-ietf-tcpm-dctcp.

[16] O. Biran, A. Corradi, M. Fanelli, L. Foschini, A. Nus, D. Raz, and E. Silvera. "A Stable Network-Aware VM Placement for Cloud Systems". *Proc. 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012)*. CCGRID '12. IEEE Computer Society, 2012, pp. 498–506. DOI: 10.1109/CCGrid.2012.119.

[17] S. Blagodurov, S. Zhuravlev, A. Fedorova, and A. Kamali. "A case for NUMA-aware contention management on multicore systems". *Proc. 19th international conference on Parallel architectures and compilation techniques*. ACM. 2010, pp. 557–558.

[18] N. Bobroff, A. Kochut, and K. Beaty. "Dynamic Placement of Virtual Machines for Managing SLA Violations". *Integrated Network Management, 2007. IM '07. 10th IFIP/IEEE International Symposium on*. May 2007, pp. 119–128. DOI: 10.1109/INM.2007.374776.

[19] W. Bolosky, R. Fitzgerald, and M. Scott. "Simple but effective techniques for NUMA memory management". *ACM SIGOPS Operating Systems Review*. Vol. 23. 5. ACM. 1989, pp. 19–31.

[20] F. Broquedis, F. Diakhaté, S. Thibault, O. Aumage, R. Namyst, and P.-A. Wacrenier. "Scheduling dynamic OpenMP applications over multicore architectures". *OpenMP in a New Era of Parallelism*. Springer, 2008, pp. 170–180.

[21] F. Broquedis, N. Furmento, B. Goglin, P.-A. Wacrenier, and R. Namyst. "ForestGOMP: an efficient OpenMP environment for NUMA architectures". *International Journal of Parallel Programming* 38.5-6 (2010), pp. 418–439.

[22] N. Calcavecchia, O. Biran, E. Hadad, and Y. Moatti. "VM Placement Strategies for Cloud Scenarios". *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*. June 2012, pp. 852–859. DOI: 10.1109/CLOUD.2012.113.

[23] Y. Cao, M. Xu, X. Fu, and E. Dong. "Explicit Multipath Congestion Control for Data Center Networks". *Proc. Ninth ACM Conference on Emerging Networking Experiments and Technologies*. CoNEXT '13. ACM, 2013, pp. 73–84. DOI: 10.1145/2535372.2535384.

[24] J. Casazza. "First the tick, now the tock: Intel microarchitecture (nehalem)". *Intel Corporation* (2009).

[25] Center for Applied Internet Data Analysis. *AS Rank: AS Ranking*. URL: http://as-rank.caida.org/.

[26] M. Cha, P. Rodriguez, J. Crowcroft, S. Moon, and X. Amatriain. "Watching Television over an IP Network". *Proc. 8th ACM SIGCOMM Conference on Internet Measurement*. IMC '08. ACM, 2008, pp. 71–84. DOI: 10.1145/1452520.1452529.

[27] W. Chen, P. Cheng, F. Ren, R. Shu, and C. Lin. "Ease the Queue Oscillation: Analysis and Enhancement of DCTCP". *Distributed Computing Systems (ICDCS), 2013 IEEE 33rd International Conference on*. July 2013, pp. 450–459. DOI: 10.1109/ICDCS.2013.22.

[28] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph. "Understanding TCP Incast Throughput Collapse in Datacenter Networks". *Proc. 1st ACM Workshop on Research on Enterprise Networking*. WREN '09. ACM, 2009, pp. 73–82. DOI: 10.1145/1592681.1592693.

[29] P. Cheng, F. Ren, R. Shu, and C. Lin. "Catch the Whole Lot in an Action: Rapid Precise Packet Loss Notification in Data Centers". *Proc. 11th USENIX Conference on Networked Systems Design and Implementation*. NSDI'14. USENIX Association, 2014, pp. 17–28. URL: http://dl.acm.org/citation.cfm?id=2616448.2616451.

[30] M. Chiosi, D. Clarke, P. Willis, A. Reid, J. Feger, M. Bugenhagen, W. Khan, M. Fargano, C. Cui, H. Denf, et al. "Network functions virtualisation: An introduction, benefits, enablers, challenges and call for action". *SDN and OpenFlow World Congress*. 2012, pp. 22–24.

[31] A. Corradi, M. Fanelli, and L. Foschini. "VM Consolidation: A Real Case Based on OpenStack Cloud". *Future Gener. Comput. Syst.* 32 (Mar. 2014), pp. 118–127. DOI: 10.1016/j.future.2012.05.012.

[32] M. Dashti, A. Fedorova, J. Funston, F. Gaud, R. Lachaize, B. Lepers, V. Quema, and M. Roth. "Traffic management: a holistic approach to memory placement on NUMA systems". *ACM SIGPLAN Notices* 48.4 (2013), pp. 381–394.

[33] G. Detal, C. Paasch, and O. Bonaventure. "Multipath in the Middle(Box)". *HotMiddlebox '13*. 2013, pp. 1–6. DOI: 10.1145/2535828.2535829.

[34] A. Dixit, P. Prakash, Y. Hu, and R. Kompella. "On the impact of packet spraying in data center networks". *INFOCOM, 2013 Proceedings IEEE*. Apr. 2013, pp. 2130–2138. DOI: 10.1109/INFCOM.2013.6567015.

[35] N. Dukkipati. "Rate Control Protocol (Rcp): Congestion Control to Make Flows Complete Quickly". AAI3292347. PhD thesis. 2008.

[36] L. Eggert and G. Fairhurst. *Unicast UDP Usage Guidelines for Application Designers*. IETF RFC 5405 (Best Current Practice). Nov. 2008. DOI: 10.17487/rfc5405.

[37] L. Eggert, G. Fairhurst, and G. Shepherd. *UDP Usage Guidelines*. Internet-Draft draft-ietf-tsvwg-rfc5405bis-07. Work in Progress. Internet Engineering Task Force, Nov. 1, 2015. URL: https://tools.ietf.org/html/draft-ietf-tsvwg-rfc5405bis.

[38] D. Erickson, B. Heller, N. McKeown, and M. Rosenblum. "Using Network Knowledge to Improve Workload Performance in Virtualized Data Centers". *Cloud Engineering (IC2E), 2014 IEEE International Conference on*. Mar. 2014, pp. 185–194. DOI: 10.1109/IC2E.2014.81.

[39] M. Al-Fares, A. Loukissas, and A. Vahdat. "A Scalable, Commodity Data Center Network Architecture". *Proc. ACM SIGCOMM 2008 Conference on Data Communication*. SIGCOMM '08. ACM, 2008, pp. 63–74. DOI: 10.1145/1402958.1402967.

[40] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. "Hedera: Dynamic Flow Scheduling for Data Center Networks". *Proc. 7th USENIX Conference on Networked Systems Design and Implementation*. NSDI'10. USENIX Association, 2010, pp. 19–19. URL: http://dl.acm.org/citation.cfm?id=1855711.1855730.

[41] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure. *TCP Extensions for Multipath Operation with Multiple Addresses*. RFC 6824. Internet Engineering Task Force, Jan. 2013. URL: http://www.rfc-editor.org/rfc/rfc6824.txt.

[42] M. Gerla, M. Sanadidi, R. Wang, A. Zanella, C. Casetti, and S. Mascolo. "TCP Westwood: congestion window control using bandwidth estimation". *Global Telecommunications Conference, 2001. GLOBECOM '01. IEEE*. Vol. 3. 2001, 1698–1702 vol.3. DOI: 10.1109/GLOCOM.2001.965869.

[43] S. Ha, I. Rhee, and L. Xu. "CUBIC: A New TCP-friendly High-speed TCP Variant". *SIGOPS Oper. Syst. Rev.* 42.5 (July 2008), pp. 64–74. DOI: 10.1145/1400097.1400105.

[44] G. Hampel, A. Rana, and T. Klein. "Seamless TCP Mobility Using Lightweight MPTCP Proxy". *Proc. 11th ACM International Symposium on Mobility Management and Wireless Access*. MobiWac '13. ACM, 2013, pp. 139–146. DOI: 10.1145/2508222.2508226.

[45] T. Hirofuchi, H. Nakada, S. Itoh, and S. Sekiguchi. "Reactive Consolidation of Virtual Machines Enabled by Postcopy Live Migration". *Proc. 5th International Workshop on Virtualization Technologies in Distributed Computing*. VTDC '11. ACM, 2011, pp. 11–18. DOI: 10.1145/1996121.1996125.

[46] M. Honda, F. Huici, G. Lettieri, and L. Rizzo. "mSwitch: A Highly-scalable, Modular Software Switch". *Proc. 1st ACM SIGCOMM Symposium on Software Defined Networking Research*. SOSR '15. ACM, 2015, 1:1–1:13. DOI: 10.1145/2774993.2775065.

[47] C.-Y. Hong, M. Caesar, and P. B. Godfrey. "Finishing Flows Quickly with Preemptive Scheduling". *Proc. ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. SIGCOMM '12. ACM, 2012, pp. 127–138. DOI: 10.1145/2342356.2342389.

[48] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer. "Achieving High Utilization with Software-driven WAN". *Proc. ACM SIGCOMM 2013 Conference on SIGCOMM*. SIGCOMM '13. ACM, 2013, pp. 15–26. DOI: 10.1145/2486001.2486012.

[49] Internet Research Lab. *Internet AS-level Topology Archive*. URL: http://irl.cs.ucla.edu/topology/.

[50] J. Iyengar and I. Swett. *QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2*. Internet-Draft draft-tsvwg-quic-protocol-01. Work in Progress. Internet Engineering Task Force, July 13, 2015. URL: https://tools.ietf.org/html/draft-tsvwg-quic-protocol.

[51] R. Iyer, H. Wang, and L. Bhuyan. "Design and analysis of static memory management policies for CC-NUMA multiprocessors". *College Station, TX, USA, Tech. Rep* (1998).

[52] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat. "B4: Experience with a Globally-deployed Software Defined Wan". *SIGCOMM Comput. Commun. Rev.* 43.4 (Aug. 2013), pp. 3–14. DOI: 10.1145/2534169.2486019.

[53] K. Jang, J. Sherry, H. Ballani, and T. Moncaster. "Silo: Predictable Message Latency in the Cloud". *Proc. 2015 ACM Conference on Special Interest Group on Data Communication*. SIGCOMM '15. ACM, 2015, pp. 435–448. DOI: 10.1145/2785956.2787479.

[54] D. Jayasinghe, C. Pu, T. Eilam, M. Steinder, and I. Whalley. "Improving performance and availability of services hosted on IaaS clouds with structural constraint-aware virtual machine placement". *2011 IEEE International Conference on Services Computing (SCC)*. IEEE, July 2011. DOI: 10.1109/SCC.2011.28.

[55] C. N. Keltcher, K. J. McGrath, A. Ahmed, and P. Conway. "The AMD Opteron processor for multiprocessor servers". *IEEE Micro* 23.2 (2003), pp. 66–76.

[56] J. Kommeri and T. Niemi. "Energy-Efficient Heterogeneous Cluster and Migration". *ENERGY 2014, The Fourth International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies*. Mar. 2014, pp. 51–56.

[57] M. Kuhlewind, D. Wagner, J. Espinosa, and B. Briscoe. "Using data center TCP (DCTCP) in the Internet". *Globecom Workshops (GC Wkshps), 2014*. Dec. 2014, pp. 583–588. DOI: 10.1109/GLOCOMW.2014.7063495.

[58] H. Q. Le, W. J. Starke, J. S. Fields, F. P. O'Connell, D. Q. Nguyen, B. J. Ronchetti, W. M. Sauer, E. M. Schwarz, and M. T. Vaden. "Ibm power6 microarchitecture". *IBM Journal of Research and Development* 51.6 (2007), pp. 639–662.

[59] M. Luckie, B. Huffaker, A. Dhamdhere, V. Giotsas, and K. Claffy. "AS Relationships, Customer Cones, and Validation". *Proc. 2013 Conference on Internet Measurement Conference*. IMC '13. ACM, 2013, pp. 243–256. DOI: 10.1145/2504730.2504735.

[60] R. Ludwig and M. Meyer. *The Eifel Detection Algorithm for TCP*. IETF RFC 3522 (Experimental). Apr. 2003. DOI: 10.17487/rfc3522.

[61] Z. Majo and T. R. Gross. "Matching memory access patterns and data placement for NUMA systems". *Proc. Tenth International Symposium on Code Generation and Optimization*. ACM. 2012, pp. 230–241.

[62] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici. "ClickOS and the Art of Network Function Virtualization". *Proc. 11th USENIX Conference on Networked Systems Design and Implementation*. NSDI'14. USENIX Association, 2014, pp. 459–473. URL: http://dl.acm.org/citation.cfm?id=2616448.2616491.

[63] S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi, and R. Wang. "TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links". *Proc. 7th Annual International Conference on Mobile Computing and Networking*. MobiCom '01. ACM, 2001, pp. 287–297. DOI: 10.1145/381677.381704.

[64] C. Matsumoto. *Time for an SDN Sequel? Scott Shenker Preaches SDN Version 2*. https://www.sdxcentral.com/articles/news/scott-shenker-preaches-revised-sdn-sdnv2/2014/10/. [Online; accessed 11-Dec-2015].

[65] X. Meng, V. Pappas, and L. Zhang. "Improving the Scalability of Data Center Networks with Traffic-aware Virtual Machine Placement". *Proc. 29th Conference on Information Communications (INFOCOM'10)*. IEEE Press, 2010, pp. 1154–1162. URL: http://dl.acm.org/citation.cfm?id=1833515.1833690.

[66] A. Munir, G. Baig, S. M. Irteza, I. A. Qazi, A. X. Liu, and F. R. Dogar. "Friends, Not Foes: Synthesizing Existing Transport Strategies for Data Center Networks". *Proc. 2014 ACM Conference on SIGCOMM*. SIGCOMM '14. ACM, 2014, pp. 491–502. DOI: 10.1145/2619239.2626305.

[67] Netflix. *Can I stream Netflix in Ultra HD?* https://help.netflix.com/en/node/13444. June 2015.

[68] D. S. Nikolopoulos, T. S. Papatheodorou, C. D. Polychronopoulos, J. Labarta, and E. Ayguadé. "User-level dynamic page migration for multiprogrammed shared-memory multiprocessors". *Parallel Processing, 2000. Proceedings. 2000 International Conference on*. IEEE. 2000, pp. 95–103.

[69] Openstack Foundation. *Cells*. URL: http://docs.openstack.org/liberty/config-reference/content/section_compute-cells.html.

[70] Openstack Foundation. *Scheduling*. URL: http://docs.openstack.org/liberty/config-reference/content/section_compute-scheduler.html.

[71] C. Paasch, G. Greenway, and A. Ford. *Multipath TCP behind Layer-4 loadbalancers*. Internet-Draft draft-paasch-mptcp-loadbalancer-00. Work in Progress. Internet Engineering Task Force, Sept. 7, 2015. URL: https://tools.ietf.org/html/draft-paasch-mptcp-loadbalancer.

[72] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal. "Fastpass: A Centralized "Zero-queue" Datacenter Network". *Proc. 2014 ACM Conference on SIGCOMM*. SIGCOMM '14. ACM, 2014, pp. 307–318. DOI: 10.1145/2619239.2626309.

[73] J. Postel. *User Datagram Protocol*. Legacy RFC 768 (Internet Standard). Aug. 1980. DOI: 10.17487/rfc768.

[74] C. Raiciu, M. Handley, and D. Wischik. *Coupled Congestion Control for Multipath Transport Protocols*. RFC 6356. Internet Engineering Task Force, Oct. 2011. URL: http://www.rfc-editor.org/rfc/rfc6356.txt.

[75] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley. "Improving Datacenter Performance and Robustness with Multipath TCP". *Proc. ACM SIGCOMM 2011 Conference*. SIGCOMM '11. ACM, 2011, pp. 266–277. DOI: 10.1145/2018436.2018467.

[76] V. Rajput, S. Kumar, and V. Patle. "Performance Analysis of UMA and NUMA Models". *International Journal of Computer Science Engineering & Technology* 2.10 (2012), pp. 1457–1458.

[77] I. Rhee, L. Xu, S. Ha, A. Zimmermann, L. Eggert, and R. Scheffenegger. *CUBIC for Fast Long-Distance Networks*. Internet-Draft draft-ietf-tcpm-cubic-00. Work in Progress. Internet Engineering Task Force, June 18, 2015. URL: https://tools.ietf.org/html/draft-ietf-tcpm-cubic.

[78] Rutube. *From Zero to 700 Gbit per Second – How One of the Russia's Largest Video-Hosting Service Uploads its Videos [S nulya do 700 gigabit v secundu — kak otgruzhaet video odin iz krupneishih videohostingov Rossii]*. http://habrahabr.ru/company/rutube/blog/269227/. Oct. 2015.

[79] F. Schmidt, O. Hohlfeld, R. Glebke, and K. Wehrle. "Santa: Faster Packet Delivery for Commonly Wished Replies". *Proc. 2015 ACM Conference on Special Interest Group on Data Communication*. SIGCOMM '15. ACM, 2015, pp. 597–598. DOI: 10.1145/2785956.2790014.

[80] D. Schwalb, J. Kossmann, M. Faust, S. Klauck, M. Uflacker, and H. Plattner. "Hyrise-R: Scale-out and Hot-Standby Through Lazy Master Replication for Enterprise Applications". *Proc. 3rd VLDB Workshop on In-Memory Data Mangement and Analytics*. IMDM '15. ACM, 2015, 7:1–7:7. DOI: 10.1145/2803140.2803147.

[81] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi. "Design and Implementation of a Consolidated Middlebox Architecture". *Proc. 9th USENIX Conference on Networked Systems Design and Implementation*. NSDI'12. USENIX Association, 2012, pp. 24–24. URL: http://dl.acm.org/citation.cfm?id=2228298.2228331.

[82] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar. "Making Middleboxes Someone else's Problem: Network Processing As a Cloud Service". *Proc. ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. SIGCOMM '12. ACM, 2012, pp. 13–24. DOI: 10.1145/2342356.2342359.

[83] S. Srikantaiah, A. Kansal, and F. Zhao. "Energy Aware Consolidation for Cloud Computing". *Proc. 2008 Conference on Power Aware Computing and Systems*. HotPower'08. USENIX Association, 2008, pp. 10–10. URL: http://dl.acm.org/citation.cfm?id=1855610.1855620.

[84] C. Su, D. Li, D. S. Nikolopoulos, M. Grove, K. Cameron, and B. R. De Supinski. "Critical path-based thread placement for NUMA systems". *ACM SIGMETRICS Performance Evaluation Review* 40.2 (2012), pp. 106–112.

[85] S. Subramanian, G. Nitish Krishna, M. Kiran Kumar, P. Sreesh, and G. R. Karpagam. "An Adaptive Algorithm For Dynamic Priority Based Virtual Machine Scheduling In Cloud". *IJCSI International Journal of Computer Science Issues* 9.6 (2012).

[86] G. Sullivan, J. Ohm, W.-J. Han, and T. Wiegand. "Overview of the High Efficiency Video Coding (HEVC) Standard". *Circuits and Systems for Video Technology, IEEE Transactions on* 22.12 (Dec. 2012), pp. 1649–1668. DOI: 10.1109/TCSVT.2012.2221191.

[87] S. Thibault, F. Broquedis, B. Goglin, R. Namyst, and P.-A. Wacrenier. "An efficient OpenMP runtime system for hierarchical architectures". *A Practical Programming Model for the Multi-Core Era*. Springer, 2008, pp. 161–172.

[88] B. Vamanan, J. Hasan, and T. Vijaykumar. "Deadline-aware Datacenter TCP (D2TCP)". *SIGCOMM Comput. Commun. Rev.* 42.4 (Aug. 2012), pp. 115–126. DOI: 10.1145/2377677.2377709.

[89] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, G. A. Gibson, and B. Mueller. "Safe and Effective Fine-grained TCP Retransmissions for Datacenter Communication". *Proc. ACM SIGCOMM 2009 Conference on Data Communication*. SIGCOMM '09. ACM, 2009, pp. 303–314. DOI: 10.1145/1592568.1592604.

[90] X. Wei, C. Xiong, and Ed. *MPTCP proxy mechanisms*. Internet-Draft draft-wei-mptcp-proxy-mechanism-02. Work in Progress. Internet Engineering Task Force, June 30, 2015. URL: https://tools.ietf.org/html/draft-wei-mptcp-proxy-mechanism.

[91] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron. "Better Never Than Late: Meeting Deadlines in Datacenter Networks". *Proc. ACM SIGCOMM 2011 Conference*. SIGCOMM '11. ACM, 2011, pp. 50–61. DOI: 10.1145/2018436.2018443.

[92] H. Wu, Z. Feng, C. Guo, and Y. Zhang. "ICTCP: Incast Congestion Control for TCP in Data Center Networks". *Proc. 6th International COnference*. Co-NEXT '10. ACM, 2010, 13:1–13:12. DOI: 10.1145/1921168.1921186.

[93] J. Xiao and Z. Wang. "A Priority Based Scheduling Strategy for Virtual Machine Allocations in Cloud Computing Environment". *Cloud and Service Computing (CSC), 2012 International Conference on*. Nov. 2012, pp. 50–55. DOI: 10.1109/CSC.2012.16.

[94] L. Ye, L. Mhamdi, and M. Hamdi. "Efficient UDP-based congestion aware transport for data center traffic". *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*. Oct. 2014, pp. 46–51. DOI: 10.1109/CloudNet.2014.6968967.

[95] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz. "DeTail: Reducing the Flow Completion Time Tail in Datacenter Networks". *SIGCOMM Comput. Commun. Rev.* 42.4 (Aug. 2012), pp. 139–150. DOI: 10.1145/2377677.2377711.

[96] J. F. Zazo, S. Lopez-Buedo, Y. Audzevich, and A. W. Moore. "A PCIe DMA Engine to Support the Virtualization of 40 Gbps FPGA-accelerated Network Appliances". *2015 International Conference on ReConFigurable Computing and FPGAs, ReConFig15, Cancun, Mexico, December 7-9, 2015*. (to appear). Dec. 2015.

[97] S. Zhuravlev, S. Blagodurov, and A. Fedorova. "Addressing shared resource contention in multicore processors via scheduling". *ACM SIGARCH Computer Architecture News*. Vol. 38. 1. ACM. 2010, pp. 129–142.

[98] A. Zimmermann and R. Scheffenegger. *Using the TCP Echo Option for Spurious Retransmission Detection*. Internet-Draft draft-zimmermann-tcpm-spurious-rxmit-00. Work in Progress. Internet Engineering Task Force, July 20, 2015. URL: https://tools.ietf.org/html/draft-zimmermann-tcpm-spurious-rxmit.

[99] A. Zimmermann, R. Scheffenegger, and B. Briscoe. *The TCP Echo and TCP Echo Reply Options*. Internet-Draft draft-zimmermann-tcpm-echo-option-00. Work in Progress. Internet Engineering Task Force, June 30, 2015. URL: https://tools.ietf.org/html/draft-zimmermann-tcpm-echo-option.

Appendices

A. mSwitch: A Highly-Scalable, Modular Software Switch

M. Honda, F. Huici, G. Lettieri, and L. Rizzo. "mSwitch: A Highly-scalable, Modular Software Switch". *Proc. 1st ACM SIGCOMM Symposium on Software Defined Networking Research*. SOSR '15. Santa Clara, California: ACM, 2015, 1:1–1:13. ISBN: 978-1-4503-3451-8. DOI: 10.1145/2774993.2775065

B. Santa: Faster Packet Delivery for Commonly Wished Replies

F. Schmidt, O. Hohlfeld, R. Glebke, and K. Wehrle. "Santa: Faster Packet Delivery for Commonly Wished Replies". *Proc. 2015 ACM Conference on Special Interest Group on Data Communication*. SIGCOMM '15. London, United Kingdom: ACM, 2015, pp. 597–598. ISBN: 978-1-4503-3542-3. DOI: 10.1145/2785956.2790014

C. A PCIe DMA Engine to Support the Virtualization of 40 Gbps FPGA-accelerated Network Appliances

J. F. Zazo, S. Lopez-Buedo, Y. Audzevich, and A. W. Moore. "A PCIe DMA Engine to Support the Virtualization of 40 Gbps FPGA-accelerated Network Appliances". *2015 International Conference on ReConFigurable Computing and FPGAs, ReConFig15, Cancun, Mexico, December 7-9, 2015*. (to appear). Dec. 2015

D. Oh Flow, Are Thou Happy? TCP sendbuffer advertising for make benefit of clouds and tenants

A. Agache and C. Raiciu. "Oh Flow, Are Thou Happy? TCP Sendbuffer Advertising for Make Benefit of Clouds and Tenants". *7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 15)*. Santa Clara, CA: USENIX Association, July 2015. URL: https://www.usenix.org/conference/hotcloud15/workshop-program/presentation/agache

E. Hyrise-R: Scale-out and Hot-Standby through Lazy Master Replication for Enterprise Applications

D. Schwalb, J. Kossmann, M. Faust, S. Klauck, M. Uflacker, and H. Plattner. "Hyrise-R: Scale-out and Hot-Standby Through Lazy Master Replication for Enterprise Applications". *Proc. 3rd VLDB Workshop on In-Memory Data Mangement and Analytics*. IMDM '15. Kohala Coast, HI, USA: ACM, 2015, 7:1–7:7. ISBN: 978-1-4503-3713-7. DOI: 10.1145/2803140.2803147

F. Claud: Coordination, Locality And Universal Distribution

J. Beilharz, F. Feinbube, F. Eberhardt, M. Plauth, and A. Polze. “Claud: Coordination, Locality And Universal Distribution”. (to appear). IOS Press, 2015