COSCAnet: virtualized sockets for scalable and flexible PaaS applications

Steffen Kächele
Institute of Distributed Systems
University of Ulm, Germany
Email: steffen.kaechele@uni-ulm.de

Franz J. Hauck
Institute of Distributed Systems
University of Ulm, Germany
Email: franz.hauck@uni-ulm.de

***Abstract*—Platform as a Service (PaaS) eases cloud deployment by automating placement decisions, scaling, and maintenance of the infrastructure. Yet, most PaaS offerings restrict network support to HTTP so that applications needing other protocols can hardly be deployed. COSCAnet is a PaaS-cloud network layer that virtualizes the well-known socket interface for UDP and TCP. Thus, legacy applications and protocol implementations can be reused in a PaaS cloud, and most of the typical network restrictions of PaaS can be released. COSCAnet also supports scale-out of applications. Our evaluations show that COSCAnet is not only more flexible but also has lower overhead and higher throughput compared to typical setups, while having only low demands on the underlying network infrastructure. As a result, COSCAnet is a leap forward towards more flexible PaaS systems that support arbitrary application protocols.**

I. Introduction

Platform as a Service (PaaS) [1] provides the highest abstraction layer to run own applications in the cloud. It lowers the burden of entrance to the cloud as customers do not have to maintain virtual machines and their platform software (e.g., an application server) [2]. Such platforms supply customers with a predefined environment consisting of programming-language support and well-defined APIs. They typically abstract from cloud characteristics such as underlying infrastructure, dynamic placement, scaling, and pay-per-use accounting [3].

Recently, cloud computing has become attractive for a growing number of application domains. Many of them require highly flexible and high-performance networking [4], [5]. However, we observed two issues that hinder current PaaS solutions from broader usage. (i) While PaaS does not actually specify the level of abstraction, current PaaS solutions usually limit communication to an application-specific request-response scheme (i.e. HTTP). Thus, PaaS clouds have became tailored to a specific application type (i.e. web applications) and cannot be used for applications needing other protocols. Examples are virtual desktop applications[1] (e.g. RDP), multimedia conferencing applications (e.g. SIP, RTP), game servers using proprietary protocols[2], e-mail servers (e.g. IMAP), servers for bulk data provision (e.g. FTP), instant messaging servers and directory services. (ii) Restricting communication to an application-specific protocol makes it easier to route network traffic on application level. Providers often use modified web servers as proxies and load balancers (e.g. Nginx). Yet on huge setups, Layer 7 load balancing is rather complex and does not scale, especially when using SSL [5].

This paper presents COSCAnet, a unique approach for a PaaS network layer (NaaS) that supports elastic applications but still provides a standard socket interface. These allow cloud applications to use arbitrary application protocols on top of UDP and TCP. Yet, COSCAnet also provides basic but protocol-independent mechanisms for scale-out. It further isolates traffic of different tenants and has integrated pay-per-use accounting. Our solution is efficient and has low hardware requirements on network components. This allows us to operate COSCAnet not only with professional network appliances but also with typical low cost commodity hardware.

The next section will present related work. Section III introduces the COSCAnet concepts whereas Section IV presents implementation aspects. Before we conclude, we evaluate COSCAnet and compare it to other cloud setups in Section V.

II. Related work

In recent years, several academic and industrial cloud computing platforms have been developed. Google App Engine[3], Amazon Beans Talk[4] and Cloud Foundry[5] are popular PaaS platforms. They allow to host servlet-based Java applications, but restrict applications to communicate solely with HTTP.

Current IaaS (Infrastructure as a Service) solutions provide more comprehensive network support. Customers deploy their software in terms of virtual machine images, but have to do many administrative tasks such as application reconfiguration for seamless scaling, which is after all a complex task. Hardware virtualization technologies such as Xen [6] and KVM are normally used to provide an isolated environment for each application. Hardware virtualization consumes a considerable amount of resources with respect to CPU and memory, which directly impacts the number of services that can effectively be consolidated onto a single physical machine [7]. It is also subject to significant performance penalties with respect to networking [8] and unstable TCP/UDP throughput [9], [10]. Thus, special network solutions for virtual machines try to optimize network performance [11], [12], e.g. by offloading TCP functionality such as acknowledging [10] and congestion control [13]. Other approaches completely outsource network

This work was partially supported by the Ernst Wilken Foundation Ulm.

[1] e.g. http://www.moka5.com

[2] e.g. http://www.onlive.com

[3] http://appengine.google.com

[4] http://aws.amazon.com/elasticbeanstalk

[5] http://www.cloudfoundry.com

processing [14]. Besides performance issues, virtual machines (VM) provide low-level network access that raises security concerns when VMs of different tenants are in the same network. Tseng et al. [15] isolate traffic by assigning each user its own VLAN tag. In contrast, COSCAnet virtualizes network access on network level (Layer 3). It conceptually isolates tenants by binding virtual sockets to separate virtual application addresses and uses firewall rules.

Yet another approach is SDN (Software Defined Networks). A popular specification is OpenFlow [16] where the data path of a switch consists of a flow table and an action associated with each flow entry. One option is to forward packets to an external controller to determine more complex routing decisions. OpenFlow, however, requires Openflow-compliant hardware. COSCAnet proposes a network layer for applications (i.e. virtualized sockets) that provides PaaS features such as transparent network elasticity. It could, however, use Openflow in its low-level network architecture.

III. The COSCAnet Approach

COSCAnet is a novel and flexible network layer for PaaS clouds. We identified four principle requirements: (i) The network layer should be as transparent as possible for cloud applications. It especially needs to provide a *standard interface* that does not restrict the way applications may communicate. (ii) COSCAnet should smoothly integrate with typical cloud network architectures and be effective with *cheap COTS hardware*. (iii) COSCAnet should support *multi-tenancy* and thus isolate network traffic of different tenants. (iv) To enable migration and scale-out of applications, applications should be addressed in a *location-transparent* way.

The main component of COSCAnet is a middleware layer that manages communication on behalf of applications. It provides standard sockets for UDP and TCP[6] that enable applications to implement arbitrary protocols on top. With the help of a dedicated network component, e.g. a router or network appliance, COSCAnet ensures that incoming requests are routed to one of many worker nodes of an application, which achieves elasticity. COSCAnet was designed as part of the run-time layer of our PaaS platform *COSCA* [17]. However, as it was developed as a self-contained library, COSCAnet and its concepts can also be used in other cloud environments. In the following, we briefly describe the main concepts of COSCAnet that are (i) virtual sockets, (ii) routing mechanisms based on virtual application addresses, (iii) elasticity, (iv) state-aware communication, and (v) isolation and accounting.

A. Virtual sockets

The COSCAnet middleware provides ordinary sockets to the application. These have exactly the same interface as sockets of the operating system, but are enriched by cloud-compliant functionality. While an application can rely on pure socket communication, the middleware manages distribution and cloud features, e.g., request routing and elasticity.

COSCAnet sockets can be seen as virtualized sockets. They first virtualize addressing. In a general network setup, each computing node is associated with its individual IP address. Instead, COSCAnet introduces and manages so-called *virtual application addresses* that address applications and services rather than nodes. In the simplest case, each application gets its own virtual application IP address. Services offered by a cloud application can thus be accessed regardless of the node on which the service is provided. Applications, in turn, can transparently operate on their IP address by opening arbitrary ports, even the same port several times in different application instances. With such a decoupling, a PaaS platform is able to achieve location transparency and deploy an application on arbitrary nodes. It can also be used to implement scale-out as we will see later. In order to reduce the number of IP addresses, we also support IP addresses shared among applications. Thus, multiple applications—typically of the same user—can open ports on the same virtual application IP address (e.g. a web server on Port 80 and an FTP server on Port 21).

[6]Other transport protocols could be added.

B. Routing

Generally, redirecting packets and connections can be done on link, network, transport or application layer of the *ISO/OSI model*. Current solutions operate either on application layer (PaaS) or on link layer using hardware emulation techniques (IaaS) (cf. Section II). In our approach, we use IP routing techniques (network layer) to redirect packets on the transport layer, in our case UDP and TCP segments. Redirecting on this layer has two major benefits. (i) It provides a maximum of generality since Internet applications communicate via IP and its transport protocols. Thus, there are no restrictions on application protocols. (ii) IP, UDP and TCP are low-level packet-based protocols that cause little state and generate low parsing overhead in routers. To supply TCP and UDP sockets, we do not need to emulate an entire network device. Instead, our virtualized sockets use original sockets from the operating system implementing the underlying network stack and communicate with an external router about routing decisions.

Whenever a new virtual socket is created and bound to the virtual application address, the router is informed about the new endpoint and its node's physical address. The router forwards incoming packets for a virtual address to the registered physical endpoint where it is delivered to the application. A closing socket de-registers the endpoint. COSCAnet also ensures that packets from the application appear as being sent from the virtual address and port number. In total, this achieves location transparency, as only COSCAnet has to deal with physical addresses.

C. Elasticity

For elasticity, a PaaS system starts the same application instance on multiple nodes, e.g., according to load parameters. COSCAnet is entirely transparent to this procedure but helps to implement it for arbitrary application protocols. Whenever an application component binds a socket to a port number already in use by another component of the same application on a different node, COSCAnet assumes that both components help each other for load balancing by elasticity. The second component will not get an error message, instead a second physical endpoint is created and the router stores a second alternative route for incoming packets. Routing is done with

transport-layer knowledge. Thus, packets of the same TCP connection are always forwarded to the same node.

For efficient scaling, one option is a round robin algorithm to distribute packets, but this method is not aware of the varying workload of nodes. For example, if two nodes host an instance of a particular application, their workload may vary due to their unique set of other hosted applications. We thus use a weighted round-robin scheme that is aware of the current system workload. For that, our virtual sockets calculate the appropriate weights either via a predefined metric such as CPU utilization in per cent, the CPU run-queue length, or a custom application-specific metric (e.g. current request rate arriving at an HTTP web application). In order to avoid continuous weight updates on the router, we introduce a fuzzy-logic scheme with three different load values (i.e. *low*, *medium* and *high*). Only when the load changes from one value to the other does the middleware update the weight correspondingly at the router. COSCAnet can thus map the typical, dynamic load of cloud computing nodes to routing decisions where less loaded nodes will get more requests.

D. State-aware communication

State within application instances may potentially interfere with forwarding requests to one of multiple virtual sockets. As already mentioned, packets of the same TCP connection are automatically routed to the same target. Furthermore, virtual sockets support persistent routing decisions for TCP and UDP based on the external address and configurable for individual ports or entire applications. We realize this by using stickiness capabilities of routers.

Web applications that use sessions are a typical use case. The state of a web session is stored only in one of the instances, and the router ensures that packets from the same session are always forwarded to this instance. Stickiness can be released after a defined time of inactivity. For web applications this timeout should typically correspond to the session timeout. Alternatively, applications can support state transfer between instances, in case clients get connected to another instance. For such applications, connection persistence is not a necessity but an optimization avoiding state transfers.

E. Isolation and accounting

Cloud platforms typically host lots of applications in the same cluster. Even worse with respect to security, they often execute applications of various customers on a single node. In order to isolate individual users, security regulations become important. Our virtual sockets achieve isolation by restricting application access to their assigned virtual application IP address. Explicit binding to other IP addresses will be denied. Furthermore, firewall features within virtual sockets restrict the usage of ports. To enforce security constraints, however, further support of the run-time environment is required (e.g., in Java, the use of a security manager that disallows reflection is a must [17]). An optional accounting module in our virtualized sockets captures the amount of traffic transferred through the network.

IV. Implementation

We have implemented our approach for commodity Linux systems. For native applications, we implemented a shared object that transparently overwrites standard sockets with COSCAnet sockets (i.e. via `LD_PRELOAD`). We also implemented our virtualized sockets as a platform-independent bootstrap library for Java. For both cases, application code does not need to be modified. Thus, virtual sockets are easy to integrate into a runtime environment that loads legacy applications. When applications require more sophisticated control of virtual socket creation, e.g., different parts of the same application should get their own IP address, we also provide Java socket providers that can be loaded (e.g. via setSocketFactory()). COSCAnet sockets internally use standard sockets reusing the protocol stack of the operating system.

A. Routing and Routing Providers

Implementation of the COSCAnet routing can be done in two different ways, using NAT or direct server return. In both cases, either a software-based router (i.e. a dedicated node) or a professional hardware load-balancer appliance can be used.

A first setup uses *Network Address Translation (NAT)*. Incoming packets arrive at the NAT router and are forwarded to physical endpoints according to the established translation rules at socket-binding time. Cloud nodes and their physical endpoints have private IP addresses. Unlike traditional NAT routers, the COSCAnet router may have multiple NAT entries for the same virtual address and implements load awareness as described in Section III-C. The NAT router pretends to have all virtual application IP addresses of cloud applications. Incoming packets are inspected on IP and transport level to identify the applicable NAT entries. Packets from virtual sockets have to pass the NAT router in order to get their sender address translated to the virtual application address.

Direct server-return is a more optimized and thus our preferred setup. A direct-return controller pretends to have all virtual application IP addresses, by answering ARP requests, whereas nodes may have one or more virtual IP addresses, but do not answer on ARP requests. Incoming packets arrive at the direct-return controller that forwards them according to the routing rules established at socket-binding time. It implements elasticity and load awareness. Forwarding requires just the replacement of the MAC address so that the packet will arrive at the targeted node which is located in the same subnet. As the targeted node maintains an interface for the virtual application IP addresses of its virtual sockets it can immediately send packets in return without involvement of the direct-return controller. This is especially useful as responses are typically much larger than requests. The controller will thereby not become a bottleneck with respect to bandwidth.

In order to use different routers, controllers and load balancers, COSCAnet comes with different routing providers, small modules communicating with the router. As we designed the routing-provider API as generically as possible, it should be easy to develop providers for other load balancers. As software-based routers have recently attracted interest due to the availability of low cost multi-core CPUs [18], [19], [20], we propose a provider for them. For commodity setups, we implement two *Linux kernel* routing providers—one with a direct server-return algorithm and one for NAT. On the router a daemon process listens on a dedicated network socket for incoming requests that assign and manage virtual application

IP addresses and their routes. For packet redirection, we use Linux Netfilter, which is well-known and highly mature (e.g., the IP Virtual Server Netfilter module which has been part of the Linux kernel since 2004). Routing providers also manage the address assignment on the computing node, e.g. binding a socket on direct return leads to the configuration of a virtual interface with the virtual application IP address and without responding to ARP requests. For high performance setups, our LTM provider fully integrates F5 BigIP LTM series load balancers into a COSCAnet setup. The provider uses an SSH connection for an on-the-fly configuration of appropriate routing rules.

B. Socket Primitives

In order to retain socket semantics, COSCAnet intercepts socket primitives. Application start-up and socket creation do not affect cloud routing unless the application binds the socket to a port and address. When an application binds its first socket, COSCAnet manages reachability of the virtual application IP address. It assigns the address at the NAT router or the controller that has to respond to appropriate ARP requests. Thus, incoming packets will arrive there first. For a direct-return setup each node also has to configure a virtual interface of the application address. In any case of *binding* and *unbinding*, COSCAnet creates and destroys the appropriate forwarding rule as discussed in Section III-B. At application termination COSCAnet shuts down the acquired interfaces down. Other socket operations do not require any interception by COSCAnet.

On bind, COSCAnet checks whether the port is available. Firewall-like rules provided for each application or virtual application IP address can restrict available ports. COSCAnet allows for binding TCP server sockets of multiple instances of the same application to the same port, leading to multiple forwarding rules in the router enabling scalability. As UDP sockets can be used for incoming and outgoing datagrams, COSCAnet allows for binding to the same port in multiple instances. However, stickiness policies in the router will direct incoming datagrams to the same instance. Thus, applications using UDP sockets for client-side operations should not explicitly bind to a port and thus get an individual port.

V. Experiments

In this section, we empirically evaluate our COSCAnet prototype. Although a major advantage of COSCAnet is to support arbitrary TCP- and UDP-based protocols, we limit our evaluation to HTTP request workloads in our experiments as this allows us to compare our approach with other cloud platforms.

Setup: We use Quad Core machines with an Intel^{TM}Xeon^{TM}CPU E3-1220 at 3.1 GHz and 16 GB DDR3 RAM. The operating system is Ubuntu 12.04.2 (64-bit server edition). All machines are connected to a gigabit switched network. Our *Native* setup represents a typical non-virtualized server; clients being directly connected to the server via a physical network. *OpenStack* is a IaaS setup that uses KVM virtualized instances. To keep performance impact at a minimum, we use large instances (m1.large) with four virtual CPUs. We tuned standard configuration in this setup by enabling a virtio paravirtualization driver, and use a bridged network

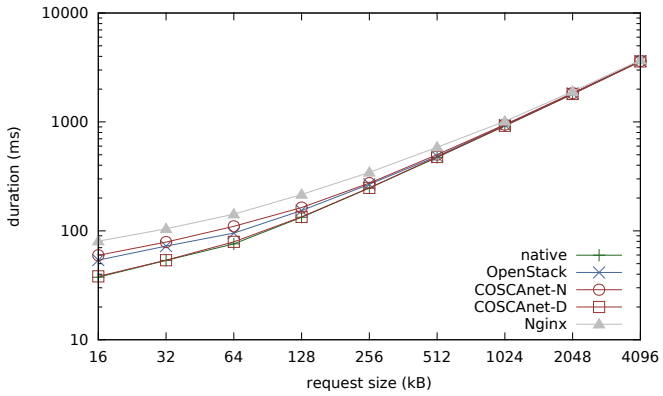

Fig. 1. Request duration depending on size (Exp. H1)

(no quantum). Further, we consider three non-virtualized PaaS setups with a load balancer running on an additional machine equipped with two network devices. The first represents current state-of-the-art PaaS using *Nginx* as HTTP load balancer. As COSCAnet can use different routing providers, we evaluate two of them: (i) direct server-return as *COSCAnet-D*, and (ii) the NAT-based setup as *COSCAnet-N*.

Experimental Methodology: We use httperf [21] to generate synthetic HTTP workloads. On the server side, we run a Jetty 2.0.4 servlet container that serves our test servlet. Each experimental result is averaged over 10 runs. Between the single tests, we wait until all connections have been completely torn down (i.e. pass the TIME_WAIT state and no longer appear in `/dev/net/tcp`).

Experiment H1 Sequent HTTP requests: In this experiment, we establish a single TCP connection (i.e. HTTP persistent connection) and generate a series of HTTP requests. On the client side, we measure the time to process 100 requests with sizes between 16 and 4096 kByte (Fig. 1). The logarithmic y-axis indicates the time the 100 requests take. Basically, we observe two phenomena: (i) The overhead introduced by the configurations notably differs, while (ii) the performance converges to native configuration for larger request sizes. In web setups, however, smaller request sizes are more likely and thus imply larger differences. COSCAnet-D introduces only marginal overhead (up to 4%). It benefits from the direct server-return topology that uses a very fast remapping with no need to modify packets. MAC addresses just have to be replaced for forwarding as destination nodes have sockets bound to an application's IP address. The return path does not need any mapping at all. COSCAnet-N introduces considerably higher overhead (up to 11%) as it needs to inspect and modify incoming and outgoing packets. Application-level proxying (Nginx) causes the highest overhead (up to 114%). As the proxy has to forward HTTP requests and responses, messages traverse the entire protocol stack two more times compared to previous setups. Our hardware virtualization setup OpenStack causes an overhead of up to 44%. It is introduced mainly due to hardware virtualization and scheduling between the guest and host system when sending packets (measurements on Eucalyptus and KVM-only provide us similar results). Yet, it lacks scaling capabilities that might be realized via a NAT-based controller or an HTTP proxy which would introduces further overhead as became obvious from the previous results.

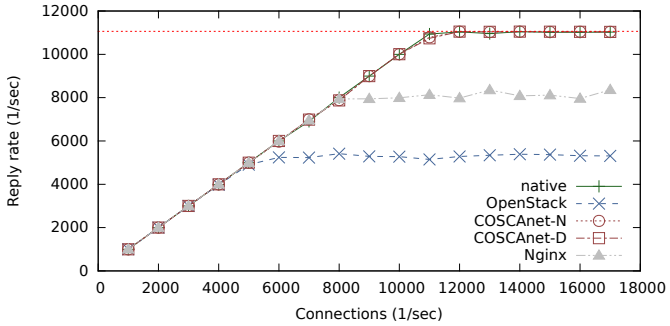

Fig. 2. HTTP processing rate, 10kB response (Exp. H2)

Experiment H2 Downloading 10 kByte: This experiment simulates multiple clients downloading web pages. For each client request, we open its own TCP connection. The experiment sends requests in a variable rate and observes the rate of arriving responses. Figure 2 shows the result for a 10 kByte page. The dotted line represents the theoretical maximum of requests/s[7]. We observe that all COSCAnet variants reach the maximum of 11060 requests/s. We observe 6307 requests/s (57%) in the OpenStack setup, and 8362 requests/s (76%) in the Nginx setup. The latter generated high CPU load on the load-balancer node. The experiment shows that COSCAnet can provide an optimum process rate for a fully-fledged PaaS system that is able to relinquish virtualization.

VI. Conclusion

Due to the emergence of cloud computing, new types of applications are moving to the cloud. Future cloud applications will require highly efficient and flexible network support that complies with typical features of cloud computing. We introduced our COSCAnet approach that uses a network virtualization on IP level disburdening network support from most restrictions. It abstracts from many cloud characteristics such as placement decisions and supports PaaS-like features such as pay-per-use, elasticity and location transparency. By using virtualized sockets, cloud users do not have to rewrite their application in order to benefit from cloud mechanisms. Our prototype contains router software and preloadable libraries for Java and C/C++ that could be integrated into existing PaaS platforms. It shows throughput rates known from native setups and can overcome typical bottlenecks at load balancers by using direct server-return techniques. For HTTP applications, we measure a significantly higher HTTP request rate than in established cloud setups. Thus, COSCAnet widens the cloud for new types of applications.

In future work, we plan to extend our solution for supporting live migration of existing TCP connections. Coupled with existing thread migration approaches [22] we want to migrate applications with long-term network connections. As future cloud architectures are expected to support dependability, we are also working on extensions for fault-tolerance. In conjunction with migration support, COSCAnet could provide TCP connection handover to manage a transparent switch of computing nodes, e.g. in a replicated setup [23].

[7]TCP handshake and teardown, protocol header as well as Ethernet checksum, preamble, interframe gap and maximum transfer unit are considered.

References

[1] P. Mell and T. Grance, "The NIST definition of cloud computing," 2009.

[2] S. Kächele, C. Spann, F. J. Hauck, and J. Domaschka, "Beyond IaaS and PaaS: An extended cloud taxonomy for computation, storage and networking," in *UCC'13: Proc. of the 6th IEEE/ACM Int. Conf. Utility and Cloud Comp.* IEEE, 2013.

[3] S. Kächele and F. J. Hauck, "Component-based scalability for cloud applications," in *CloudDP '13: Proc. of the 3rd Int. Workshop on Cloud Data and Platforms*. ACM, 2013, pp. 19–24.

[4] S. Rixner, "Network virtualization: Breaking the performance barrier," *ACM Queue*, vol. 6, no. 1, p. 37:36 ff., 2008.

[5] E. Keller and J. Rexford, "The "platform as a service" model for networking," in *Proc. of Workshop on Res. on Enterpr. Netw.* USENIX, 2010, pp. 4–4.

[6] Barham et al., "Xen and the art of virtualization," in *Proc. of the 19th ACM Symp. on Oper. Sys. Princ.* ACM, 2003, pp. 164–177.

[7] Y. Mei, L. Liu, X. Pu, and S. Sivathanu, "Performance measurements and analysis of network I/O applications in virtualized cloud," in *Proc. of the 3rd Int. Conf. on Cloud Comp.*, 2010, pp. 59 –66.

[8] A. Menon, J. R. Santos, Y. Turner, G. J. Janakiraman, and W. Zwaenepoel, "Diagnosing performance overheads in the Xen virtual machine environment," in *Proc. of the 1st ACM/USENIX Int. Conf. on Virtual Exec. Env.* ACM, 2005, pp. 13–23.

[9] G. Wang and T. Ng, "The impact of virtualization on network performance of Amazon EC2 data center," in *Proc. of INFOCOM*, 2010, pp. 1–9.

[10] A. Kangarlou, S. Gamage, R. R. Kompella, and D. Xu, "vSnoop: Improving TCP throughput in virtualized environments via acknowledgement offload," in *Proc. of the 2010 ACM/IEEE Int. Conf. for High Perf. Comp., Netw., Storage and Analysis*. IEEE, 2010, pp. 1–11.

[11] A. Menon, A. L. Cox, and W. Zwaenepoel, "Optimizing network virtualization in Xen," in *Proc. of the Ann. Techn. Conf.* USENIX, 2006.

[12] Zhang et al., "Evaluating and optimizing I/O virtualization in kernel-based virtual machine (KVM)," in *Netw. and Par. Comp.*, ser. LNCS, Ding et al., Ed. Springer, 2010, vol. 6289, pp. 220–231.

[13] S. Gamage, A. Kangarlou, R. R. Kompella, and D. Xu, "Opportunistic flooding to improve TCP transmit performance in virtualized clouds," in *Proc. of the 2nd ACM Symp. on Cloud Comp.* ACM, 2011, pp. 24:1–14.

[14] H. Eiraku, Y. Shinjo, C. Pu, Y. Koh, and K. Kato, "Fast networking with socket-outsourcing in hosted virtual machine environments," in *Proc. of the 2009 ACM Symp. on Applied Comp.* ACM, 2009, pp. 310–317.

[15] H.-M. Tseng, H.-L. Lee, J.-W. Hu, T.-L. Liu, J.-G. Chang, and W.-C. Huang, "Network virtualization with cloud virtual switch," in *Proc. of the 17th Int. Conf. on Par. and Dist. Systems*, Dec. 2011, pp. 998–1003.

[16] McKeown et al., "Openflow: enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.

[17] S. Kächele, J. Domaschka, and F. J. Hauck, "COSCA: an easy-to-use component-based PaaS cloud system for common applications," in *Proc. of the 1st Int. Workshop on Cloud Comp. Platf.*, ser. CloudCP '11. ACM, 2011.

[18] P. Costa, "Bridging the gap between applications and networks in data centers," *SIGOPS Oper. Syst. Rev.*, vol. 47, no. 1, pp. 3–8, Jan. 2013.

[19] T. Marian, K. S. Lee, and H. Weatherspoon, "Netslices: scalable multi-core packet processing in user-space," in *Proc. of the 8th ACM/IEEE Symp. on Arch. for Netw. and Comm. Sys.* ACM, 2012, pp. 27–38.

[20] L. Rizzo, "Netmap: a novel framework for fast packet i/o," in *Proc. of the Ann. Techn. Conf.* USENIX, 2012.

[21] D. Mosberger and T. Jin, "httperf - a tool for measuring web server performance," *SIGMETRICS Perform. Eval. Rev.*, vol. 26, pp. 31–37, 1998.

[22] Bouchenak et al., "Zero overhead Java thread migration," INRIA, Tech. Rep., 2002.

[23] F. J. Hauck, S. Kächele, J. Domaschka, and C. Spann, "The COSCA PaaS platform: on the way to flexible and dependable cloud computing," in *Proc. of the 1st Europ. Workshop on Dep. Cloud Comp.*, ser. EWDCC '12. NY, USA: ACM, pp. 1–2.