Bridging the Gap Between Applications and Networks in Data Centers

Paolo Costa
Imperial College London
costa@imperial.ac.uk

1. INTRODUCTION

Modern data centers host tens (if not hundreds) of thousands of servers and are used by companies such as Amazon, Google, and Microsoft to provide online services to millions of individuals distributed across the Internet. They use commodity hardware and their network infrastructure adopts principles evolved from enterprise and Internet networking. Applications use UDP datagrams or TCP sockets as the primary interface to other applications running inside the data center. This effectively isolates the network from the end-systems, which then have little control over how the network handles packets. Likewise, the network has limited visibility on the application logic. An application injects a packet with a destination address and the network just delivers the packet. Network and applications effectively treat each other as *black-boxes*.

This strict separation between applications and networks (also referred to as *dumb network*) is a direct outcome of the so-called *end-to-end argument* [49] and has arguably been one of the main reasons why the Internet has been capable of evolving from a small research project to planetary scale, supporting a multitude of different hardware and network technologies as well as a slew of very diverse applications, and using networks owned by competing ISPs.

Despite being so instrumental in the success of the Internet, this black-box design is also one of the root causes of inefficiencies in large-scale data centers. Given the little control and visibility over network resources, applications need to use low-level hacks, e.g., to extract network properties (e.g., using `traceroute` and IP addresses to infer the network topology) and to prioritize traffic (e.g., increasing the number of TCP flows used by an application to increase its bandwidth share). Further, a simple functionality like multicast or anycast routing is not available and developers must resort to application-level overlays. This, however, leads to inefficiencies as typically multiple *logical* links are mapped to the same *physical* link, significantly reducing application throughput. Even with perfect knowledge of the underlying topology, there is still the constraint that servers usually have only a single network interface. Any logical topology with a fan-out higher than one cannot be mapped optimally to the physical network.

This black-box design is also detrimental from the perspective of network operators. Since the network has little or no knowledge of the semantics of application data, traffic engineering is hard. For instance, the network cannot easily distinguish between online, time-sensitive, traffic vs. background or delay-tolerant traffic. Likewise, inferring the flow dynamics and dependencies is a non-trivial task.

These shortcomings suggest that it may be a good time to revisit some of the early design decisions and verify if the original assumptions still hold. In particular, we maintain that data centers exhibit quite unique properties compared to traditional Internet systems.
First, in data centers, hardware and network technologies are very homogeneous, with x86 servers and Ethernet being by far the most popular choice. Second, the bulk of core services (e.g., GFS [32] or BigTable [24]) and applications (e.g., MapReduce [28] jobs or web services) running is rather limited. Third, the network topology is known and, at least to some extent, customizable. Finally, most components, including software (e.g., hypervisors), are owned and controlled by a single entity. This means that customization of software, hardware, and network is feasible and full compatibility with legacy technologies is a non-goal.

Given these differences, we argue that data centers should not be considered as mini-Internets but, instead, they should be seen as an opportunity to rethink established approaches in networking and investigate alternative solutions, aiming to improve performance and reduce development and management complexity.

We investigated some of these opportunities in our recent work [14, 19, 26, 38]. We will briefly report on some of these projects in Section 2 while in Section 3 we discuss our current research activity, which focuses on simplifying and enhancing the interaction between applications and network in a cloud computing scenario. We then outline the research challenges in Section 4 and discuss the related work in Section 5. Finally, we conclude the paper in Section 6 with brief ending remarks.

2. BACKGROUND: CAMCUBE

In the CamCube project [14, 26], we have been undertaking a clean-slate approach to the design of data centers, borrowing ideas from the fields of high performance computing (HPC), distributed systems, and networking. We use a direct-connect topology, similar to those used in HPC,

in which servers are directly connected to each other using cross-over cables, creating a 3D torus topology (or k-ary 3-cube) [46], like the one depicted in Figure 1. Servers are therefore responsible for routing all the internal traffic. Switches are only used to connect CamCube servers to the external networks but are not used to route intra-CamCube traffic.

The key benefit of CamCube is that by using a direct-connect topology and letting servers handle packet forwarding and processing, it completely removes the distinction between computation and network devices.
This enables services to easily implement custom-routing protocols (e.g., multicast or anycast) as well as efficient in-network services (e.g., caching or in-network aggregation), without incurring the typical overhead (path stretch, link sharing, etc.) and development complexity introduced by overlays.

Rather than attempting to fix the problems incrementally, CamCube demonstrates that by designing a data center cluster from the ground up, including the network topology and the network stack, it is possible to achieve higher performance, simplify the design and development of applications, and reduce cluster costs.

As an example of the benefits that can be achieved with this platform, we briefly describe Camdoop [26], a MapReduce system running on CamCube that supports on-path aggregation of data streams.

A common property of MapReduce (and in general of "Big Data" applications) is that often data is aggregated during the process and the output size is a fraction of the input size. This motivated us to explore a different approach to improve the performance of MapReduce. Rather than increasing the bandwidth, e.g., like in [34, 56], we focus on decreasing the traffic by pushing aggregation from the edge into the network core.

Camdoop builds aggregation trees with the sources of the intermediate data as the children and roots at the servers executing the final aggregation. A *convergecast* is performed, where all on-path servers intercept packets, aggregate their content in a new packet and forward it to the upstream server. This significantly reduces network traffic because at each hop only a fraction of the data received is forwarded. This enables achieving a speed-up of up to two orders of magnitude compared to Hadoop and Dryad/DryadLINQ [26].

Camdoop is a paradigmatic example of the philosophy underlying CamCube. Instead of treating application traffic just as bits to ship from one end-host to another (as in today's networks), Camdoop exploits application knowledge to improve network performance.

Although we evaluated CamCube through a custom testbed, using x86 servers and Ethernet cables, our approach aligns well with the recent trend of deploying 3D torus-based appliances in clusters [10, 12]. Unfortunately, however, these platforms use traditional network stacks like TCP/IP and MPI, which are oriented towards point-to-point server communication and completely hide the underlying topology, thus inhibiting the implementation of any in-network functionality. We are currently porting our network stack to one SeaMicro appliance. This will allow us to take advantage of the SeaMicro hardware, thus further increasing the performance of CamCube.

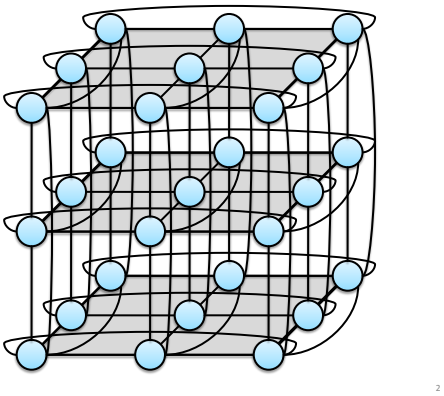

Figure 1: A 3-ary 3-cube (or 3D torus)

3. NETWORK-AS-A-SERVICE (NAAS)

The experience accumulated with CamCube confirms our expectations that a tighter integration and cooperation between applications and network could be beneficial for both. We are currently investigating how to apply similar ideas in the more challenging context of cloud computing as opposed to the production data center environment targeted by CamCube. This requires striking a balance between the fine-grained control offered by CamCube to application developers and the flexibility and isolation required by cloud providers as well as the ability to be incrementally deployed and fully compatible with existing cloud applications.

3.1 Motivation

Cloud computing has made it possible for small companies or even single individuals to access virtually unlimited resources in large data centers for running computationally demanding tasks. This has triggered the rise of "big data" applications, which operate on large amounts of data. These include traditional batch processing applications, such as data mining, data indexing, log collection and analysis, and scientific applications [2, 3, 7, 55], as well as real-time stream processing, web search and advertising [4, 6, 11].

These applications typically adopt a *partition/aggregate* model: a large input data set is distributed over many servers, and each server processes a share of the data. Locally generated intermediate results must then be aggregated to obtain the final result. For example, this is how MapReduce [28] or mainstream web search engines [6] are built.

An open challenge of the partition/aggregate model is that it results in high contention for network resources in data centers when a large amount of data traffic is exchanged between servers. Facebook reports that, for 26% of processing tasks, network transfers are responsible for more than 50% of the execution time [25]. This is consistent with other studies, showing that the network is often the bottleneck in big data applications [16, 34, 35].

Improving the performance of such *network-bound applications* in data centers has attracted much interest from the research community. A class of solutions focuses on reducing bandwidth usage by employing overlay networks to distribute data [25] and to perform partial aggregation [40]. However, this requires applications to reverse-engineer the physical network topology to optimize the layout of overlay

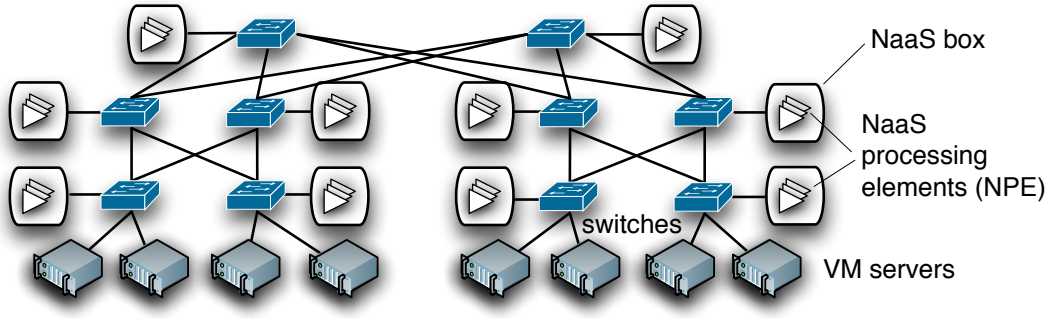

Figure 2: NaaS architecture in a data center.

networks and, as we discussed in Section 1, even with full network visibility, it would still be inefficient.

Other proposals increase network bandwidth through more complex topologies [15, 34] or higher-capacity networks [56]. These approaches, however, exhibit two main drawbacks. First, to fully exploit the extra bandwidth available, complex techniques must be used to infer application demands and traffic patterns at runtime [16, 21]. Second, network over-provisioning can significantly increase the data center operational and capital expenditures, up to 5 times according to some estimates [39]. This, in turn, impacts tenants costs as well. For example, Amazon AWS recently introduced Cluster Compute instances with full-bisection 10 Gbps bandwidth, with an hourly cost of 16 times the default Amazon EC2 instance [1].

3.2 Overview

Recently, triggered by the availability of multi-core CPUs at low cost, there has been a renewed interest in software-based routers, [30, 36, 41, 42, 48], and FPGA-based [45] programmable routers. These proposals usually aim to replace traditional functionality of switches and routers with a custom software implementation. In contrast, we argue that the flexibility provided by these implementations should be offered to tenants to implement part of the application logic in the network.

Instead of over-provisioning the network, our goal is to optimize network traffic by exploiting application-specific knowledge in the network core. We term this approach *"Network-as-a-service"* (NaaS) [27] because it allows tenants to customize the service that they receive from the network. Our goal is to enable tenants to efficiently, easily, and safely control network operations while still allowing the provider to decide how resources are allocated and shared across tenants.

NaaS-enabled tenants can deploy custom routing protocols, including multicast services [22] or anycast/incast protocols, as well as more sophisticated mechanisms, such as content-based routing [23] and content-centric networking [37]. This would allow applications to distribute data and requests efficiently through custom multicast and anycast protocols instead of inefficient overlay networks.

Further, by modifying the content of packets on-path, tenants could efficiently implement advanced network services, such as in-network data aggregation and smart caching, that are *application-specific* (as opposed to traditional *application-agnostic* network services). Parallel processing systems such as MapReduce would greatly benefit because data can be aggregated on-path, thus reducing execution times [26]. Data intensive applications can reduce network traffic by exploiting customized redundancy elimination techniques [17] or by caching popular data items within the network, which decreases latency and bandwidth usage compared to end-host-only deployments. Our preliminary simulation results using simple yet representative cloud traffic patterns show that the NaaS approach can reduce the median flow completion time of applications by approximately 60%–90% [27].

3.3 Preliminary Architecture

Figure 2 shows the envisioned data center architecture for NaaS. Compared to existing setups, NaaS requires network devices to be capable of efficiently executing tenant-provided code. To logically distinguish the functionality of traditional packet switching from the more advanced NaaS functionality, we colloquially refer to the network component responsible for executing tenant code as a *NaaS box*. They can be implemented as separate devices connected via high bandwidth links to switches like in SideCar [52] or can be integrated with the switch hardware, e.g., using a hybrid solution comprising a NetFPGA-based hardware layer [45], coupled with a flexible software stack. NaaS boxes host instances of *NaaS processing elements*, NPEs, which carry out application-specific packet processing. For a given tenant application, NPEs execute on the subset of NaaS boxes that are along the routing paths between the virtual machines (VMs) in the network.

A critical requirement for the implementation of NaaS boxes is their ability to process packets at line rate. We assume that network switches (and hence NaaS boxes) are interconnected in a (potentially oversubscribed) fat-tree topology [15], as shown in Figure 2. This topology, increasingly adopted in data centers [13], has the property that it requires only switches with a limited port count (e.g., for a 27,648-server cluster, only 48-port switches are needed). This means that the worst-case rate that a NaaS box must support is limited to tens of Gbps, which can be supported by today's NetFPGAs (with four 10 Gbps ports [9]) and also by commodity servers [30, 36, 48]. While rates may increase in the near future due to the deployment of 10 Gbps Ethernet in data centers, we expect that advances in multi-core platforms and FPGA technology can still meet our goals. Distributed software router designs like in RouteBricks [30] can also be used to scale to higher rates.

4. RESEARCH CHALLENGES

In order to see a widespread adoption of NaaS, a range of research challenges have to be overcome.

C1: Integration with current data center hardware and software. Existing data centers constitute a significant investment for large cloud providers like Google or Amazon. For NaaS to become successful, it is important that the techniques devised can be incrementally deployed in today's cloud data centers and be compatible with existing solutions for powering, cooling, and interconnecting servers [20]. Also, we need to ensure that NaaS can coexist with legacy applications and cloud services, without negatively impacting them.

C2: Scalability. The number of tenant applications running in the network will be at least one or two orders of magnitude higher than what is currently supported by programmable routers. Even with only a moderate number of tenants in a data center, the NaaS infrastructure must handle the concurrent execution of a large number of NPEs because each tenant requires multiple NPEs depending on the complexity of the physical network topology. Similar to how cloud computing providers require scalable and efficient management solutions to handle tenant VMs, NaaS infrastructure will necessitate automatic management of NPEs.

C3: Performance isolation. Unlike previous proposals on flexible data center networking, which typically assume only a handful of *cooperative* and *trusted* services, a NaaS model exposes in-network processing to tenants. Therefore, NaaS must be able to execute *malicious* or *poorly written* code without impacting the performance of other tenants, including non-NaaS tenants. This will require the development of novel mechanisms to ensure performance isolation, both *locally*, e.g., using lightweight OS containers, and *globally*, e.g., using in-network solutions that efficiently partition the bandwidth among multiple tenants [19].

C4: Resource allocation. A key requirement for NaaS is that the provider does not have to reveal too much information to the tenants about the internal network configuration. Besides the obvious privacy reasons, this is also important because it allows the provider to retain the ability to dynamically (re-)allocate the tenant's resources (end-hosts and networking devices). For instance, the provider could decide to run the code of a tenant only on a subset of networking devices if this improves the overall performance without significantly impacting the tenant. This requires investigating novel allocation schemes that take into account the new capabilities offered by NaaS and the performance/cost tradeoffs involved, including new models of fairness, e.g., [33].

C5: Programming abstractions. The additional flexibility and performance provided by NaaS must not come at the cost of additional complexity for the users. This can be achieved in two ways. For popular applications, e.g., MapReduce or key-value stores, the provider can offer bespoke versions that have been *internally* modified to exploit the NaaS functionality (e.g., by using in-network aggregation) while still maintaining the current *external* interface that tenants are familiar with. For custom applications, instead, we need to devise novel programming abstractions that strike a balance between expressiveness and complexity. For example, this may include high-level domain specific languages such as Cloud Haskell [31], as well as low-level languages like Click [44], which allow for finer-grained control. Also, it is important to decouple the *virtual* topology offered to the tenants from the *physical* topology in order to *i)* enable providers to re-arrange the internal configuration without impacting existing code and to *ii)* enable tenants to move to another provider without having to completely rewrite their applications.

C6: Pricing model and incentives. Finally, cloud providers will require new charging models for NaaS offerings. We have already started exploring some of the tradeoffs [18] but more investigation is required to also account for in-network computation. However, we expect that a mutual benefit for providers and tenants will be achieved. Taking advantage of NaaS will in many cases yield a reduction in overall traffic, e.g., by using in-network aggregation or optimized forwarding. This would be beneficial for NaaS tenants, which can improve their performance and, hence, would be happy to pay a premium. Beside this additional income, the provider should also benefit because it can re-allocate the unused bandwidth to the non-NaaS tenants, which therefore will also see an improvement in their performance, potentially leading to an even higher revenue.

5. RELATED WORK

There have been several proposals to support network programmability. Here, we briefly summarize the main related work, highlighting the challenges (as described in the previous section) introduced by the NaaS model.

5.1 Programmable Routers

Triggered by a desire to lower costs and increase performance, there has recently been a renewed interest in software- [30,36,41,42,48] and FPGA-based [45] programmable routers, including commercially available products [5,8]. These proposals aim to replace traditional switch and router operations (e.g., IPv4 forwarding) with software implementations to reduce costs and increase flexibility. The NaaS model leverages these efforts but goes beyond them: it offers the flexibility of programmable packet-processing to tenants. This, however, challenges current design assumptions. First, we expect that the number of independent tenants services that NaaS-devices must be able to support will be at least one or two orders of magnitude higher than what can be supported by current programmable routers [29] (challenge *C2*). Second, these services are written by third-parties and compete with each other for resources. They must be isolated from other components without impacting tenant performance (*C3*).

5.2 Software Defined Networking (SDN)

SDN has been recently proposed as a flexible way for centrally managing the network control plane [43, 53]. The NaaS model is complementary to these efforts—while SDN focuses on packet forwarding and targets network administrators, NaaS allows arbitrary on-path application-specific packet processing and is geared to cloud tenants. Although NaaS shares some of the research challenges of SDN, e.g., providing an abstract view of the network, it also introduces new ones. For example, the OpenFlow interface [43] is too limited to express the wide range of possible computations required by NaaS (*C5*). NaaS also requires new mecha-

nisms to virtualize and isolate packet processing (*C3*) and scheduling protocols to allocate in-network resources to tenants (*C4*).

5.3 Active Networks (ANs)

ANs were proposed more than a decade ago as way to simplify network management by allowing packets to carry code that is executed by routers [54, 57]. NaaS shares some ideas with this body of work. However, there are also some important differences. First, ANs were mostly intended to be used to implement new routing protocols rather than in-network processing services like those targeted by NaaS. This means that both the API and their implementation are ill-suited to capture the complexity and performance requirements posed by NaaS. Second, NaaS exposes network programmability to cloud tenants. This will require novel programming abstractions for non-expert developers (*C5*) as well as appropriate hardware and software performance isolation (*C3*), and resource allocation mechanisms (*C4*). Finally, NaaS targets data center networks rather than Internet WANs. This allows for a higher degree of customization of hardware and software and it potentially enables overcoming some of the implementation issues that in the past hampered a widespread deployment of ANs.

5.4 Middleboxes

While switches and routers focus on *packet forwarding*, middleboxes are responsible for *packet processing*. This includes functionality such as WAN optimizers, proxies, IDSes, NATs, and firewalls. Traditionally, middleboxes have been implemented on close and difficult to extend hardware platforms. Recently, however, there have been several proposals that exploit software-centric implementations to provide more flexibility and simplify management, e.g., [50, 51]. NaaS integrates these efforts by focusing on *application-specific* packet processing such as in-network aggregation or caching, which introduces new challenges (*C2* and *C3*). However, through NaaS, tenants can also implement functions analogous to those usually provided by middleboxes but with the possibility of customizing them based on applications needs, e.g., optimizing traffic compression according to the specific application workload.

6. CONCLUSIONS

Data center networks exhibit significant differences compared to traditional IP-based networks. This motivated us to explore alternative solutions. In particular, we argue that a better integration of applications and network can be beneficial for both entities. We investigated these opportunities in the context of the CamCube project and, more recently, with NaaS.

We believe that the NaaS model has the potential to revolutionize current cloud computing offerings by increasing the performance of tenant applications—through efficient in-network processing—while reducing development complexity. Cloud providers will benefit too because by offering a better service to tenants and by optimizing network usage, they can increase their customer base (and revenue due to higher utilization). Our initial results show that NaaS does not require pervasive adoption in a data center to be cost-effective [27].

We believe that by combining (distributed) computation and network in a single, coherent, abstraction NaaS would provide a significant step towards realizing the vision of "the data center is the computer" [47].

7. REFERENCES

[1] Amazon EC2 Pricing. `http://goo.gl/vtYE`.

[2] Amazon HPC. `http://goo.gl/jHuAm`.

[3] Big Data @ Foursquare . `http://goo.gl/FAmpz`.

[4] Big Data in Real Time at LinkedIn. `http://goo.gl/6OzCN`.

[5] Cisco Application Oriented Networking. `http://goo.gl/NnHlI`.

[6] Google Tree Distribution of Requests . `http://goo.gl/RpB45`.

[7] Hadoop Wiki: PoweredBy. `http://goo.gl/Bbfu`.

[8] Juniper Application Service Modular Line Card. `http://goo.gl/0EnU7`.

[9] NetFPGA-10G. `http://goo.gl/sa0fS`.

[10] SeaMicro SM15000 Family. `http://goo.gl/g7bGt`.

[11] Twitter Storm. `http://goo.gl/Y1AcL`.

[12] Why Torus-Based Clusters? `http://goo.gl/EHTj6`.

[13] James Hamilton's Blog, 2011. `http://goo.gl/yyqyB`.

[14] Abu-Libdeh, H., Costa, P., Rowstron, A., O'Shea, G., and Donnelly, A. Symbiotic Routing in Future Data Centers. In *SIGCOMM* (2010).

[15] Al-Fares, M., Loukissas, A., and Vahdat, A. A Scalable, Commodity Data Center Network Architecture. In *SIGCOMM* (2008).

[16] Al-Fares, M., Radhakrishnan, S., Raghavan, B., Huang, N., and Vahdat, A. Hedera: Dynamic Flow Scheduling for Data Center Networks. In *NSDI* (2010).

[17] Anand, A., Sekar, V., and Akella, A. SmartRE: An Architecture for Coordinated Network-Wide Redundancy Elimination. In *SIGCOMM* (2009).

[18] Ballani, H., Costa, P., Karagiannis, T., and Rowstron, A. The Price Is Right: Towards Location-independent Costs in Datacenters. In *HotNets* (2011).

[19] Ballani, H., Costa, P., Karagiannis, T., and Rowstron, A. Towards Predictable Datacenter Networks. In *SIGCOMM* (2011).

[20] Barroso, L. A., and Hölzle, U. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan & Claypool Publishers, 2009.

[21] Bazzaz, H. H., Tewari, M., Wang, G., Porter, G., Ng, T. S. E., Andersen, D. G., Kaminsky, M., Kozuch, M. A., and Vahdat, A. Switching the optical divide: Fundamental challenges for hybrid electrical/optical datacenter networks. In *SOCC* (2011).

[22] Cain, B., and Towsley, D. F. Generic Multicast Transport Services: Router Support for Multicast Applications. In *IFIP NETWORKING* (2000).

[23] Carzaniga, A., and Wolf, A. L. Forwarding in a Content-Based Network. In *SIGCOMM* (2003).

[24] Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A., and Gruber, R. E. Bigtable: A Distributed Storage System for Structured Data. In *OSDI* (2006).

[25] Chowdhury, M., Zaharia, M., Ma, J., Jordan, M. I., and Stoica, I. Managing Data Transfers in Computer Clusters with Orchestra. In *SIGCOMM* (2011).

[26] Costa, P., Donnelly, A., Rowstron, A., and O'Shea, G. Camdoop: Exploiting In-network Aggregation for Big Data Applications. In *NSDI* (2012).

[27] Costa, P., Migliavacca, M., Pietzuch, P., and Wolf, A. L. NaaS: Network-as-a-Service in the Cloud. In *Hot-ICE* (2012).

[28] Dean, J., and Ghemawat, S. MapReduce: Simplified Data Processing on Large Clusters. In *OSDI* (2004).

[29] Dobrescu, M., Argyraki, K., and Ratnasamy, S. Toward Predictable Performance in Software Packet-Processing Platforms. In *NSDI* (2012).

[30] Dobrescu, M., Egi, N., Argyraki, K., Chun, B.-G., Fall, K., Iannaccone, G., Knies, A., Manesh, M., and Ratnasamy, S. RouteBricks: Exploiting Parallelism To Scale Software Routers. In *SOSP* (2009).

[31] Epstein, J., Black, A., and Jones, S. P. Towards Haskell in the cloud. In *Haskell Symposium* (2011).

[32] Ghemawat, S., Gobioff, H., and Leung, S.-T. The Google File System. In *SOSP* (2003).

[33] Ghodsi, A., Sekar, V., Zaharia, M., and Stoica, I. Multi-resource Scheduling for Packet Processing. In *SIGCOMM* (2012).

[34] Greenberg, A., Hamilton, J. R., Jain, N., Kandula, S., Kim, C., Lahiri, P., Maltz, D. A., Patel, P., and Sengupta, S. VL2: A Scalable and Flexible Data Center Network. In *SIGCOMM* (2009).

[35] Guo, C., Wu, H., Tan, K., Shiy, L., Zhang, Y., and Luz, S. DCell: A Scalable and Fault-Tolerant Network Structure for Data Centers. In *SIGCOMM* (2008).

[36] Han, S., Jang, K., Park, K., and Moon, S. PacketShader: A GPU-Accelerated Software Router. In *SIGCOMM* (2010).

[37] Jacobson, V., Smetters, D. K., Thornton, J. D., Plass, M. F., Briggs, N. H., and Braynard, R. L. Networking Named Content. In *CoNEXT* (2009).

[38] Jalaparti, V., Ballani, H., Costa, P., Karagiannis, T., and Rowstron, A. Bridging the Tenant-Provider Gap in Cloud Services. In *SOCC* (2012).

[39] Kandula, S., Padhye, J., and Bahl, P. Flyways To De-Congest Data Center Networks. In *HotNets* (2009).

[40] Logothetis, D., Trezzo, C., Webb, K. C., and Yocum, K. In-situ MapReduce for Log Processing. In *USENIX ATC* (2011).

[41] Lu, G., Guo, C., Li, Y., Zhou, Z., Yuan, T., Wu, H., Xiong, Y., Gao, R., and Zhang, Y. ServerSwitch: A Programmable and High Performance Platform for Data Center Networks. In *NSDI* (2011).

[42] Marian, T., Lee, K. S., and Weatherspoon, H. NetSlices: Scalable Multi-Core Packet Processing in User-Space. In *ANCS* (2012).

[43] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. OpenFlow: enabling innovation in campus networks. *CCR 38*, 2 (2008).

[44] Morris, R., Kohler, E., Jannotti, J., and Kaashoek, M. F. The Click Modular Router. In *SOSP* (1999).

[45] Naous, J., Gibb, G., Bolouki, S., and McKeown, N. NetFPGA: Reusable Router Architecture for Experimental Research. In *PRESTO* (2008).

[46] Parhami, B. *Introduction to Parallel Processing: Algorithms and Architectures*. Kluwer Academic Publishers, 1999.

[47] Patterson, D. A. The Data Center Is The Computer. *Commununications of ACM 51*, 1 (2008).

[48] Rizzo, L. netmap: a novel framework for fast packet I/O. In *Usenix ATC* (2012).

[49] Saltzer, J. H., Reed, D. P., and Clark, D. D. End-to-end Arguments in System Design. *ACM TOCS 2*, 4 (1984).

[50] Sekar, V., Egi, N., Ratnasamy, S., Reiter, M., and Shi, G. Design and Implementation of a Consolidated Middlebox Architecture. In *NSDI* (2012).

[51] Sherry, J., Hasan, S., Scott, C., Krishnamurthy, A., Ratnasamy, S., and Sekar, V. Making Middleboxes Someone Else's Problem. In *SIGCOMM* (2012).

[52] Shieh, A., Kandula, S., and Sirer, E. G. SideCar: Building Programmable Datacenter Networks without Programmable Switches. In *HotNets* (2010).

[53] Tavakoli, A., Casado, M., Koponen, T., and Shenker, S. Applying NOX to the Datacenter. In *HotNets* (2009).

[54] Tennenhouse, D., Smith, J., Sincoskie, W., Wetherall, D., and Minden, G. A survey of active network research. *Communications Magazine, IEEE 35*, 1 (1997).

[55] Thusoo, A., Shao, Z., Anthony, S., Borthakur, D., Jain, N., Sen Sarma, J., Murthy, R., and Liu, H. Data warehousing and analytics infrastructure at facebook. In *SIGMOD* (2010).

[56] Wang, G., Andersen, D. G., Kaminsky, M., Kozuch, M., Ng, T. S. E., Papagiannaki, K., and Ryan, M. c-Through: Part-time optics in data centers. In *SIGCOMM* (2010).

[57] Wetherall, D. Active Network Vision and Reality: Lessons from a Capsule-Based System. In *SOSP* (1999).