The Case for the Superfluid Cloud

Filipe Manco Joao Martins Kenichi Yasukata Jose Mendes Simon Kuenzer Felipe Huici
NEC Europe Ltd.

Abstract

The confluence of a number of relatively recent trends including the development of virtualization technologies, the deployment of micro datacenters at PoPs, and the availability of microservers, opens up the possibility of evolving the cloud, and the network it is connected to, towards a *superfluid cloud*: a model where parties other than infrastructure owners can quickly deploy and migrate virtualized services throughout the network (in the core, at aggregation points and at the edge), enabling a number of novel use cases including virtualized CPEs and on-the-fly services, among others.

Towards this goal, we identify a number of required mechanisms and present early evaluation results of their implementation. On an inexpensive commodity server, we are able to concurrently run up to 10,000 specialized virtual machines, instantiate a VM in as little as 10 milliseconds, and migrate it in under 100 milliseconds.

1 Introduction

Cloud and data centers deployments have become pervasive, largely as a result of the availability of inexpensive commodity servers, higher capacity network links and ever improving virtualization technologies. Services such as Amazon EC2, Rackspace, and Microsoft Azure are now commonplace and the adoption of cloud-based technologies is ever accelerating.

More recently, network operators have started getting into the game through the Network Function Virtualization (NFV) trend, whereby network services and processing are run within virtual machines (VMs) on top of commodity hardware servers [2]. In addition, squeezed by tougher competition and constantly increasing bandwidth demands, operators are trying to move away from acting as simple bit pipes and into the more lucrative field of providing services; to achieve this, a number of major telcos have started deploying *micro data centers* (e.g., a single rack of commodity servers) at Points-of-Presence sites, initially to run their own services but in the longer run to rent out those resources to third parties [4].

The trend towards deployment of commodity hardware at network edges goes beyond this: Google deploys large numbers of caches in networks throughout the world as does Akamai, and a recent ETSI white paper on Mobile Edge Computing (MEC) argues for placing servers in aggregation and radio access networks next to base stations and radio network controllers [3]. Other big players such as Intel are also jumping in, calling for the deploying of servers in so-called “smart cells” [6].

One final trend is the recent availability of a number of low-cost, low-energy, single-board *microservers* (e.g., CubieTrucks, Raspberry Pis, fit-PCs, etc.) equipped with ARM, x86 and MIPS CPUs. These microservers open up the possibility of pushing cloud services to the very edge of access networks, in places where energy consumption or space constraints might render the deployment of traditional servers an impossibility.

We believe that the confluence of all of these trends constitutes the basic infrastructure needed towards an evolution of the cloud, and the network it is connected to, towards a *superfluid cloud*: a model where multi-tenant, virtualized software-based services run on common, shared commodity hardware infrastructure deployed throughout the network. Network operators and third parties (e.g., cloud operators, service and content providers, application developers and even end users) would have the the ability to instantiate such services on-the-fly, whenever needed, and run potentially thousands of them on a single inexpensive platform (thus supporting a large number of concurrent users), migrating them near-instantaneously (in milliseconds, allowing for transparent adaptation to changing requirements and network conditions) and deploying them across a wide range of hardware and locations, ranging from base stations and multi-cell aggregation sites all the way to data centers in the core of the network.

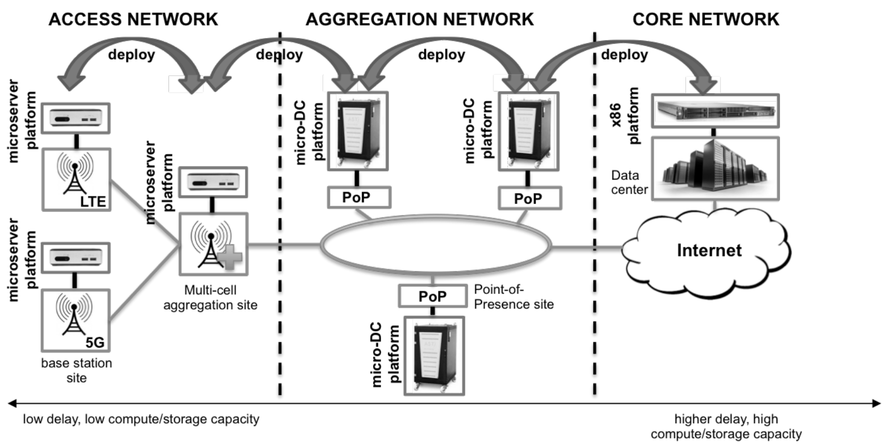


Figure 1: Superfluid cloud architecture. Multi-tenant platforms are deployed throughout the network: at the core, the aggregation network and the edge. Virtualized processing is instantiated on them quickly and whenever needed, in milliseconds, and migrated when conditions change or applications require it.

Figure 1 shows the overall superfluid cloud architecture in greater detail. A set of platforms running on different types of hardware (microservers, small racks, larger x86 deployments) are set up at different points in the network: next to base stations and aggregation sites in access networks, at micro data centers at Point-of-Presence sites in aggregation networks, and at full-fledged data centers in the core network. Each of these platforms is multi-tenant, and network processing and services can be instantiated by third parties on-the-fly, when and where they are needed. End-users, application developers and any tenant decide the trade-off between low-delay access near the edge (left-hand side of the figure) and high compute/storage capacity near the core (right-hand side) [1].

To bring this vision closer to reality, we make a number of specific contributions: (1) optimizations that allow us to concurrently boot as many as 10,000 virtual machines on a single, relatively inexpensive commodity server, and to keep their creation times down to hundreds of milliseconds in the worst case (e.g., for the 10,000th VM); (2) a comparison with LXC containers that shows that minimalistic VMs have creation times on par with containers; and (3) optimizations that reduce migration times from 480 to 82 milliseconds.

2 Motivating Use Cases and Requirements

So far we have painted a rather high level picture of the superfluid cloud architecture and the idea of running virtualized services and processing whenever and wherever needed, throughout different networks and on different types of commodity hardware. To make things more concrete, we now describe a number of illustrating use cases:

- **Virtual CPE:** A number of operators are driving towards virtualizing CPEs (Customer Premise Equipment) and running them in their own data centers, as opposed to customers' homes, in order to reduce maintenance costs and simplify roll out of new features. To support this while keeping costs down, it would be beneficial to be able to run as many of these instances as possible on a single server (in the range of thousands to be inline with the number of residential customers), and for the server to achieve high (cumulative) throughput.
- **Virtual CDNs:** Virtual CDN operators could deploy virtualized content caches at edge networks, growing their infrastructure as their business grows. This points to the need of running many virtual cache instances concurrently, and to provide high throughput, especially for delivery of high definition video. Fast migration could further be used to move a cache to a more efficient place in the network without losing its state (e.g., statistics about content hits).
- **On-the-fly Services:** Mobile customers could pay for on-demand ad removal in order to improve their browsing experience without draining their battery power, or could use aggressive traffic compression at the edge when the cellular load is high. Such use cases would benefit from fast instantiation of the virtualized service, and could go as far as supporting fine-granularity, per-flow processing by quickly creating a VM as the TCP SYN of a flow is first seen. This is similar to JITSU [8], which optimizes VM and connection start-up times on x86 and ARM-based devices, and Cloudlets [12], which proposed offloading processing from mobile devices to the edge.

[1]Some of this is already happening, but only for point solutions by big players who can afford high up-front deployment costs (e.g., Google cache).

- **Latency-sensitive Applications:** Applications such as Google glass, Siri or game servers could be run directly at the edge, perhaps using microserver platforms.
- **Personalized Edge Services:** Per-customer services (e.g., firewalls, parental control, etc.) could be deployed at the edge, and quickly migrated whenever a mobile device changes networks.

While certainly not comprehensive, this list gives an idea of the type of mechanisms that these use cases require. First among these is what we term *massive consolidation*, running large numbers of virtualized instances on a single server. This is important at the edge of the network, since not that many servers are available there; useful for microservers which are resource-constrained; and advantageous in the core where reducing operating costs (energy, cooling) and investment ones (i.e., fewer servers) is key.

In addition, fast migration is required in order to be able to adapt to changing network conditions (e.g., increased link delay at a particular site, flash crowds, etc.). Doing so in less than one hundred milliseconds would even open up the possibility of migrating services without breaking end user connections [10].

Further, fast (e.g., tens of milliseconds) service instantiation and teardown would allow for on-the-fly instantiation of services (and their quick teardown), perhaps even as a connection is created, and high throughput would be needed to support potentially high cumulative rates when running large number of concurrent VMs. Finally, microserver support is needed to be able to push processing all the way to the very edge of the network.

Having identified a set of requirements, in the rest of the paper we explain the progress we have made towards them and provide early evaluation results. It is worth noting that security concerns as to whether certain types of services should be allowed to run (and by whom) are out of scope for this paper, but should certainly be addressed if the vision of a superfluid cloud is to have a chance of becoming reality. Thankfully, there is ongoing work towards using static checking techniques to programmatically decide whether certain network processing should be deployed in a network [13].

3 Implementation

As an initial prototype, we base our implementation on the Xen hypervisor, and the guests (i.e., the virtual machines) on MiniOS, a minimalistic, para-virtualized operating system distributed with the Xen sources. As future work, we are looking to expand this to include KVM and other operating systems for the guests such as OSv [7] and stripped-down versions of Linux. We will further look at carrying out optimizations for containers.

Our work continues along the trend of using specialized, minimalistic VMs (e.g, Mirage[9], ClickOS [11], Erlang on Xen [1], OSv [7], etc.), but takes it further, seeking to optimize the number of VMs that can be concurrently run, how fast they can be instantiated and torn down, how fast they can be migrated, their memory footprint, which platforms they can run on, and the throughput that can be handled. We are further working towards comparing these against containers, and provide some early results of this in this paper.

3.1 Massive Consolidation

Our first and biggest contribution is towards *massive consolidation*: the ability to concurrently run large numbers of VMs (potentially 10,000 or more) on a single, inexpensive commodity server. Out of the box, when we started this work with Xen 4.2, we were limited to only about 300 guests at most due to Linux not being configured to provide enough file descriptors in order for Xen to provide console access to the VMs. While this issue is easily fixed, a number of others existed, so we carried out three major changes and a number of minor modifications in order to push the number of concurrent guests.

The first major change was to the XenStore, a `proc`-like back-end used to keep information about running VMs (e.g., their names and IDs, their virtual mac addresses, etc.). Basic operations such as creation, destruction and migration of VMs need to read and write entries to it, and so their performance is tightly linked to that of the XenStore. To improve it, we have written a streamlined version from scratch we call `lixs` (LIghtweight XenStore); `lixs` is written in c++, consists of about 2500 LoC, and has a pluggable system allowing us to use different storage back-ends (e.g., from a full database to a simple map).

The second major change is to `xl`, the main Xen management command and toolstack used to carry out operations such as VM creation and console access. We partially replace `xl` by `xcl` (XenCtrl Light), a simplified toolstack comprising 600 LoC and tailored to our purposes (e.g., it only supports para-virtualized and PVH modes, and VIF virtual interfaces). Among other things, it reduces the number of required per-guest XenStore entries from 37 to just 17.

The third change is to Xenconsoled, the daemon in charge of providing users with console access to the VMs. Out of the box, the daemon has high CPU overhead during high rates of VM creation (practically 100% for the core assigned to it). To address this, we modify it to use the `epoll` mechanism which scales better to large numbers of watched file descriptors, and we optimize the VM creation process by, for instance, preventing Xenconsoled from listing all existing domains every time a domain is created or destroyed.

Finally, we carried out a set of minor changes such as (1) increasing Linux's maximum number of PTYs, file descriptors and IRQs; (2) upgrading to Xen 4.4 to take advantage of higher numbers of available event channels (virtual interrupts); (3) using a ramfs for dom0's and the guests' root filesystems; and (4) replacing the hotplug script in charge of setting up virtual interfaces and adding them to the back-end switch with a faster, compiled, purpose-built, udevd-like daemon.

3.2 Fast Service Instantiation

Most of the modifications we implemented for massive consolidation apply to this category, and have the effect of reducing VM creation and destruction times.

3.3 Fast Migration

In Xen, performing migration consists of the `xl` toolstack connecting a file descriptor to the stdin/sdtout of an ssh process in order to pipe the VM image out to the receiving host. Benchmarking reveals that the overall migration time for a minimalistic VM is about 400 milliseconds, most of which is spent on copying data over the ssh pipe and only a small portion of which on write system calls.

We optimized this process by implementing a daemon that is deployed on the receiver. The daemon allocates a socket which the toolstack uses to write domain pages to; this is then used in the low-level Xen APIs so that the hypervisor can directly receive the pages. After all pages have been received, the receive daemon gets a final notification to bring up the (restored) domain, completing the process.

As future work, we are looking into optimizing live migration in other to reduce actual downtime, as well as investigating potential bottlenecks in other virtualization technologies such as KVM. We are further planning on evaluating these mechanisms when VMs are carrying out memory-intensive processing.

3.4 Microserver Support

In order to see the feasibility of using microservers as platforms for the superfluid cloud, we attempt to run Xen, KVM and/or containers (`lxc`) on a number of them, ranging from ARM-based CubieTrucks and Raspberry Pis to x86-based Intel NUCs, Intel Edisons and AMD Gizmos. This is still ongoing work, and we are planning on conducting extensive testing to see, among other things, what sort of processing and how many concurrent VMs these microservers can handle.

3.5 High Performance and Throughput

We rely on the performance numbers obtained through the optimized network back-end introduced in [11]. Beyond this, we are in the process of implementing *persistent grants*, a mechanism that provides significant speedups without needing changes to the VMs nor major changes to the network back-end. In addition, the work in [11] had at most 100 concurrent VMs running; scaling the back-end, and in particular the software switch, to larger numbers is ongoing research.

4 Evaluation

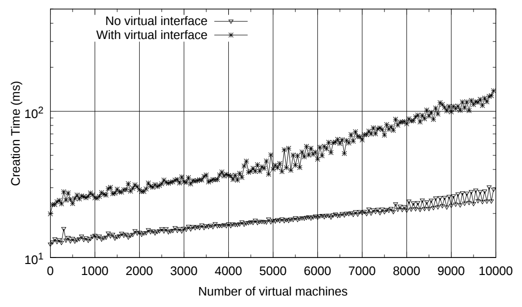
Unless otherwise stated, all experiments were performed on a system with four AMD Opteron 6376@2.3GHz CPUs (64 cores total) and 128GB of memory, costing about $4,000 and running Xen 4.4 and Linux 3.14 for dom0. The virtual machines are based on MiniOS and include basic functionality that allows them to respond to pings; this is one of the ways we use to verify that they are correctly instantiated. In future work we will carry tests with more realistic workloads.

As a first test, we conduct a *boot storm*: we attempt to create up to 10,000 virtual machines on our server as quickly as possible, and for each VM we measure the time it took for it to be created. For comparison purposes, we carry out the same test using LXC containers (kernel version 3.16.3 and 256KB of memory assigned to each container) instead of Xen/VMs.

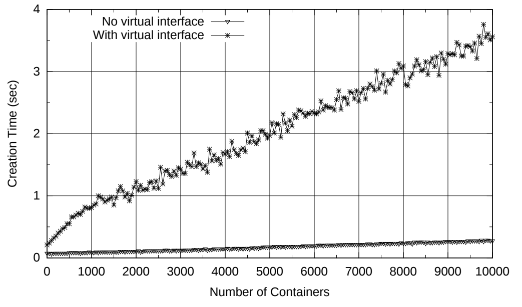
The results for our minimalistic Xen VMs when using all of the optimizations described in the previous section are shown in figure 2(a). We are able to run as many as 10K concurrent virtual machines, with creation times ranging from about 20 milliseconds with a virtual interface and 12 msecs without for the first VM, up to a still rather low 135 msecs with a virtual interface and 30 msecs without one for the 10,000th VM. While these are still rather early results, they are, to the best of our knowledge, the first time this large number of VMs have been concurrently run on commodity hardware.

Figure 2(b) depicts results for LXC containers. We measure a container creation time of about 210 milliseconds with an interface and 70 msecs without one for the first container; and creation times of 3.5 seconds with an interface and 270 msecs without one for the 10,000th container. This shows that minimalistic VMs are a viable alternative to containers whenever strong isolation is a requirement.

Figure 3 shows the effect that using our improved XenStore (`lixs`) and toolstack (`xcl`) have over using the standard `oxenstored` [5] and toolstack (`xl`) when carrying out a boot storm. The bottom line in the graph is there as reference and shows the same (best) results as



(a) Minimalistic Xen VMs (log scale, in milliseconds).



(b) LXC containers (linear scale, in seconds).

Figure 2: Comparison of boot storm times for Xen-based minimalistic virtual machines versus LXC containers. Each point denotes the time it took for the nth VM or container to start up.

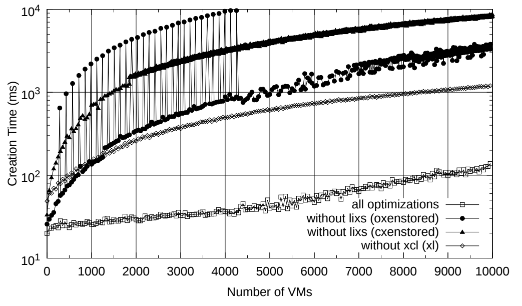


Figure 3: Breakdown of Xenstore and toolstack optimizations (log scale).

figure 2, that is, when all optimizations are enabled. The curve above it reports figures when we switch from `xcl` to the standard `xl`; the next curve are results when using `oxenstored` instead of `lixs`; and the final curve reports figures when relying on the older, C-based `cxenstored`.

Overall, these optimizations bring tangible results. For instance, for the 10,000th VM, using `oxenstore` yields boot times in the range of 2.9 seconds and 1.9 seconds with `xl`, compared to about 135 milliseconds with all optimizations turned on. It is worth noting that the spikes on the `oxenstored` curve might be due to the OCaml garbage collector kicking in; we are currently looking into the issue to confirm this.

To test our migration optimizations, we conducted a test whereby we create an increasing number of VMs on

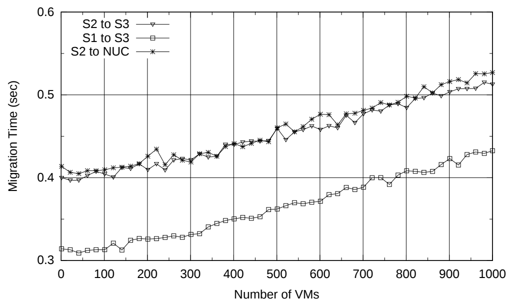


Figure 4: Migration time between servers and between a server and a microserver.

a server and migrate each of them to another server in turn. For each measurement, the receiving server has no VMs running[2] so that we measure the performance of the sending host.

Figure 4 shows migration times between servers and between a server and a microserver. The server labeled S1 has 2x Intel Xeon E5-2697 v2 2.70GHz CPUs (24 cores total), S2 has 4x AMD Opteron 6376@2.3GHz CPUs (64 cores total), S3 an Intel Xeon E5-1630 v3 3.7 GHz CPU and the microserver is an Intel NUC with an i5-4250U 2.6 GHz processor. Tests were done using a 1Gb/s link.

For the S2 to S3 case, it takes 400 msecs for the first VM to be migrated and 512.7 msecs for the 1000th VM. When replaced with a faster sender (S1 to S3) results improve considerably to 314,1 msecs for the first one and 432.6 msecs for the last one; for the server to microserver case (S2 to NUC), it takes 413.7 msecs for the first VM and 526.9 msecs secs for the 1,000th one.

As a further test, we connected two S3 servers over a direct, 10Gb/s link. With this setup, we were able to measure a migration time of only 86.4 msecs (versus 143 msecs on a 1Gb/s link). We further obtained a migration time of 480 msecs when using the standard toolstack as opposed to ours, showing that our optimizations provide a significant improvement.

The reader may have noticed approximately a factor of 5 difference between the 86.4 msecs when migrating from S3 versus 413.7 for S2. We note that the difference in memory throughput between these two machines is approximately a factor 5; we are conducting further tests to confirm this as the cause.

5 Conclusions and Future Work

We have presented the notion of the superfluid cloud, an architecture that allows parties other than infrastructure owners to quickly deploy and migrate virtualized services throughout the network: in the core, at aggregation

[2]To achieve this, after each iteration we destroy the VM at the receiver and re-create it at the sender.

points and at the edge. We have made inroads towards implementing some of the mechanisms required by it, including the ability to concurrently run 10,000 VMs on a single commodity server, instantiate services in 10 milliseconds, and migrate them in under 100 milliseconds.

6 Discussion Topics

We address each of the issues mentioned in the CFP in turn:
Expected feedback: We would like to hear whether people are seeing similar trends or are actively doing research towards some of the mechanisms described in the paper. We are particularly interested in use cases we have missed, ways to improve our work, and potential collaborations.
Controversial points: In the past, we have wondered about the applicability of some of the mechanisms we have developed (e.g., massive consolidation, fast instantiation) to actual products and deployment, so this may prove a point of contention. On our end, our concerns have been allayed a number of times by operators coming up with use cases we had not thought of that leverage these mechanisms.
Workshop discussion: We hope that this paper will generate discussion around the usefulness of the superfluid cloud concept, its mechanisms, and whether there are use cases we should be addressing. At a lower level, there might be a discussion around the use of virtual machines versus containers, or about how Xen-specific some of the results are.
Issues not addressed: We have not yet tested the system under realistic workloads. This is likely to raise potentially difficult scheduling issues when running large numbers of VMs, as well as putting stress on the backend software switch used to connect VMs with the physical interface(s). Also, we have not particularly tested the performance of the different microservers we have.

In addition, as mentioned in the paper, security mechanisms to decide whether certain types of services should be deployed (and where and by whom) are out of scope. Also, we have not looked at management issues to do with potentially having to deal with large numbers of VMs, and having to have an efficient management framework so that its overhead does not shatter the fast instantiation and migration times of our system.
Circumstances causing the idea to fall apart: It could be the case that the trend towards deployment of servers and micro data centers at the edge slows down or stops, or that no practical way of opening up these resources to third parties is found (e.g., because of security or policy issues). However, in our opinion there are enough important players driving these developments to make this unlikely.

References

[1] ERLANG ON XEN. Erlang on Xen. `http://erlangonxen.org/`, July 2012.

[2] ETSI PORTAL. Network Functions Virtualisation: An Introduction, Benefits, Enablers, Challenges and Call for Action. `http://portal.etsi.org/NFV/NFV_White_Paper.pdf`, October 2012.

[3] ETSI PORTAL. Mobile-Edge Computing - Introductory Technical White Paper. `https://portal.etsi.org/Portals/0/TBpages/MEC/Docs/Mobile-edge_Computing_-_Introductory_Technical_White_Paper_V1%2018-09-14.pdf`, September 2014.

[4] FRANK, B., POESE, I., LIN, Y., SMARAGDAKIS, G., FELDMANN, A., MAGGS, B., RAKE, J., UHLIG, S., AND WEBER, R. Pushing cdn-isp collaboration to the limit. *SIGCOMM Comput. Commun. Rev. 43*, 3 (Jul 2013), 34–44.

[5] GAZAGNAIRE T., H. V. OXenstored an efficient hierarchical and transactional database using functional programming with reference cell comparisons. In *Proceedings of the ACM SIGPLAN International Conference on Functional Programming, ICFP* (2009), pp. 203–214.

[6] INTEL. Smart cells revolutionize service delivery. `http://www.intel.de/content/dam/www/public/us/en/documents/white-papers/smart-cells-revolutionize-service-delivery.pdf`.

[7] KIVITY, A., LAOR, D., COSTA, G., ENBERG, P., HAR'EL, N., MARTI, D., AND ZOLOTAROV, V. Osv—optimizing the operating system for virtual machines. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)* (Philadelphia, PA, Jun 2014), USENIX Association, pp. 61–72.

[8] MADHAVAPEDDY, A., LEONARD, T., SKJEGSTAD, M., GAZAGNAIRE, T., SHEETS, D., SCOTT, D., MORTIER, R., CHAUDHRY, A., SINGH, B., LUDLAM, J., CROWCROFT, J., AND LESLIE, I. Jitsu: Just-In-Time Summoning of Unikernels. In *NSDI* (2015).

[9] MADHAVAPEDDY, A., MORTIER, R., SOHAN, R., GAZAGNAIRE, T., HAND, S., DEEGAN, T., MCAULEY, D., AND CROWCROFT, J. Turning down the lamp: software specialisation for the cloud. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing* (Berkeley, CA, USA, 2010), HotCloud'10, USENIX Association, pp. 11–11.

[10] MANCO, F., MARTINS, J., AND HUICI, F. Towards the super fluid cloud. In *Proceedings of the 2014 ACM SIGCOMM Conference (Demo Session))* (New York, NY, USA, 2014), SIGCOMM '14, ACM, pp. 355–356.

[11] MARTINS, J., AHMED, M., RAICIU, C., OLTEANU, V., HONDA, M., BIFULCO, R., AND HUICI, F. Clickos and the art of network function virtualization. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)* (Seattle, WA, Apr 2014), USENIX Association, pp. 459–473.

[12] SATYANARAYANAN, M., BAHL, P., CACERES, R., AND DAVIES, N. The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing 8*, 4 (Oct 2009), 14–23.

[13] STOENESCU, R., OLTEANU, V., POPOVICI, M., AHMED, M., MARTINS, J., BIFULCO, R., MANCO, F., HUICI, F., SMARAGDAKIS, G., HANDLEY, M., AND RAICIU, C. In-net: In-network processing for the masses. In *Proceedings of the European conference on Computer systems* (2015), EuroSys '15, ACM.