Zero-Loss Virtual Machine Migration with IPv6 Segment Routing

Yoann Desmouceaux
École Polytechnique, Cisco Systems
yoann.desmouceaux@polytechnique.edu

Mark Townsley
Cisco Systems, École Polytechnique
townsley@cisco.com

Thomas Heide Clausen
École Polytechnique
thomas.clausen@polytechnique.edu

***Abstract*—With the development of large-scale data centers, Virtual Machine (VM) migration is a key component for resource optimization, cost reduction, and maintenance. From a network perspective, traditional VM migration mechanisms rely on the hypervisor running at the destination host advertising the new location of the VM once migration is complete. However, this creates a period of time during which the VM is not reachable, yielding packet loss.**

This paper introduces a method to perform zero-loss VM migration by using IPv6 Segment Routing (SR). Rather than letting the hypervisor update a locator mapping after VM migration is complete, a logical path consisting of the source and destination hosts is pre-provisioned. Packets destined to the migrating VM are sent through this path using SR, shortly before, during, and shortly after migration – the virtual router on the source host being in charge of forwarding packets locally if the VM migration has not completed yet, or to the destination host otherwise. The proposed mechanism is implemented as a VPP plugin, and feasibility of zero-loss VM migration is demonstrated with various workloads. Evaluation shows that this yields benefits in terms of session opening latency and TCP throughput.

I. Introduction

Data-centers are shifting to hosting more and more heterogeneous workloads, whose lifetimes can vary greatly, and which can exhibit rapidly changing resource demands and inter-application dependencies. Furthermore, the development of live virtual machine (VM) migration [1] (and more recently, of container migration [2]) has allowed for extended flexibility in the assignment of tasks to machines. In this context, many architectures have been developed in which workload migration is used as a baseline to achieve different goals [3]: energy minimization [4], network usage minimization [5], operational cost minimization [6], maintenance [7], *etc.*

From a network perspective, a challenge raised by live migration lies in maintaining connectivity to a VM after it has been migrated. Indeed, hypervisors normally assume that VMs are migrated within a single LAN, using Reverse ARP (RARP) to advertise the new location of a VM after migration. Traditional techniques to overcome this issue rely on Layer 2 overlays, such as VXLAN [8] or NVGRE [9]. Other approaches include making use of Mobile IP [10] or of LISP [11]. However, with the emergence of IPv6 [12] data-centers, new opportunities appear, both for the addressing of workloads and for re-engineering the data-path. Among the proposed approaches for mobility within IPv6 data-centers, Identifier-Locator Addressing (ILA) [13] has been proposed at the IETF, using high-order bytes of addresses to denote locators and low-order bytes of addresses to denote identifiers.

A drawback of all these approaches is that they incur a period of time, during which packets addressed to the migrating VMs are lost. This raises concerns for applications that are intolerant to packet losses (*e.g.* UDP-based delay-critical applications, virtual network functions, ...).

A. Statement of Purpose

The purpose of the paper is to introduce a VM mobility solution, which provides "zero-loss" capabilities: assuming that the network is not lossy, any packet destined to a migrating VM will eventually be received. To that purpose, the Segment Routing (SR) [14] architecture is leveraged.

SR is an architecture which allows packets within a designated domain to be added an extraneous header, designating an ordered list of *segments* through which the packet is expected to go. Segments represent abstract functions to be performed on packets, and can be as simple as *forward to next segment* (enabling source routing and traffic engineering), but can also represent more complicate instructions (from custom encapsulation or routing behavior to complete virtual network functions). This paper specifically targets the IPv6 flavour of SR, in which segments are represented by means of IPv6 addresses, and where an IPv6 extension header [15] contains the list of segments. Backwards compatibility with non-SR-aware routers is maintained, by having the waypoints modify the destination address of the packet to that of the next segment, allowing traversal of arbitrary routers while travelling between two segments.

The idea underlying this paper is to use SR to solve the locator/identifier mapping synchronization problem during VM migration, by letting the gateway (proactively) route packets destined to a migrating VM through a path comprising the old and the new hosts. This way, the responsibility of the old host is reduced to (i) forwarding packets locally while the migration is not complete and (ii) forwarding packets to the next segment (the new host) once migration has completed. This way, no packets are lost during the migration, whereas traditional solutions would require a mapping to be updated *once* migration has completed, leading to potential packet losses during this period of time. Furthermore, the overhead on the old host is kept to a minimum, since no tunnel has to be opened and no Layer-2 overlay is required.

978-3-903176-14-0

B. Related Work

Many solutions have been proposed to address the issue of maintaining Layer-3 network connectivity during and after VM migration. The simplest solutions involve using a Layer-2 overlay [8], [9], and relying on hypervisors sending gratuitous Reverse ARP messages after migration. In addition to the operational complexity incurred by running such overlays, the VM is not reachable on the new host until the RARP has propagated, leading to potential packet losses.

Another simple solution consists of creating an IP tunnel between the source and the destination hosts. In [16], the Xen hypervisor is modified so that after migration, packets reaching the hypervisor at the source host are tunnelled towards the new host. After migration, the VM uses two different addresses (an "old" and a "new" address), and Dynamic DNS is used so that external clients can reach the VM via the new address. The drawback of this approach is that the hypervisor must co-operate with the destination host during an unpredictable amount of time by performing tunnelling. This is incompatible with recent architectures, where packets destined to VMs are handled by virtual switches rather than by the hypervisor itself.

In [17], Mobile IP is used to assist the migration process. Traffic from/to the VM is routed through a home agent, which tunnels it to the correct server hosting the VM. It is the role of the hypervisor to update the home agent with the new location of the VM once it has migrated. Thus, after VM migration, packets can wrongfully reach the source host before registration of the new location is complete. This approach is improved in [18], by configuring, before migration, a dummy secondary interface for the destination network, and swapping primary and secondary interfaces after migration. This requires co-operation with the VM as two interfaces are used.

In [19], LISP is used to address the issue of Layer-3 connectivity during VM migration. Packets destined to a VM traverse a LISP router, which encapsulates them towards the current resource locator of the VM. The hypervisor is modified so that, after VM migration, the mapping system is updated to reflect the new resource locator of the VM. This avoids triangular routing, but once again the VM is not reachable during the period of time when the mapping is being updated.

Finally, some other approaches are orthogonal to the context of this paper, but worth mentioning. In [20], TCP options are used to facilitate migration of TCP connections. In a Software Defined Networking (SDN) context, [21] proposes a framework for virtual router migration, in which control planes and data planes are migrated during two distinct phases.

II. SR-based Migration

This paper assumes that a VM is located within an IPv6 data-center, and accessible through a *virtual address* (VIP). Each machine within the data-center is accessible at a *physical address* (PIP), and can host several VMs. A *location database* maintains a mapping between each virtual address and the physical address of the server hosting the corresponding VM. Located at the edge of the data-center, a gateway advertises

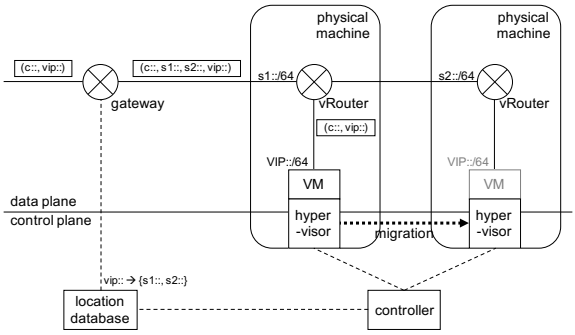

Figure 1. SR VM Migration: migrating VM.

(*e.g.* with BGP) the whole virtual address prefix. When receiving traffic for a given VIP, the router will steer it through an SR policy consisting of one segment, namely the PIP of the machine hosting the VM. Each machine is running a virtual router (in our implementation, VPP[1]), which is connected to the physical interface(s) of the server, and to each VM by means of a virtual interface (in our implementation, a `vhost-user` interface). In sum, under normal operation (when the VM is running on a single host), the gateway will tunnel traffic for a VM towards the machine hosting it.

The mechanism introduced in this paper assumes that a controller is running in the data-center, which decides about the allocation of VMs to physical machines. When the controller decides to move a VM from a host to another, it proactively modifies the routing tables of the gateway, so that the traffic for the corresponding VIP is steered into an SR policy consisting of two segments, corresponding to the old and new machines hosting the VM. This way, when the VM is migrating, the gateway will steer traffic to the old host. As long as the VM has not completed migration, traffic will be intercepted by the old host (figure 1); as soon as migration is complete, the old host will simply forward traffic to the next segment, namely the new host. This way, no synchronization is required between the old host and the network: traffic is loosely sent to a logical path comprising both hosts.

III. Detailed Specification

This section introduces a formal description of the SR functions necessary to perform zero-loss migration, as well as the behavior of the gateway.

A. Segment Routing Functions

Three SR functions are defined:

1) Forward to Local (fw): The fw function simply forwards the packet to the next segment. In SR terminology [22], this corresponding to the END behavior.

2) Forward to Local if Present (fwp): The fwp function forwards the packet to the last segment (skipping intermediary segments), only if the corresponding VIP v is present locally; otherwise it acts as the fw function and forwards the packet to

[1]https://gerrit.fd.io/r/vpp

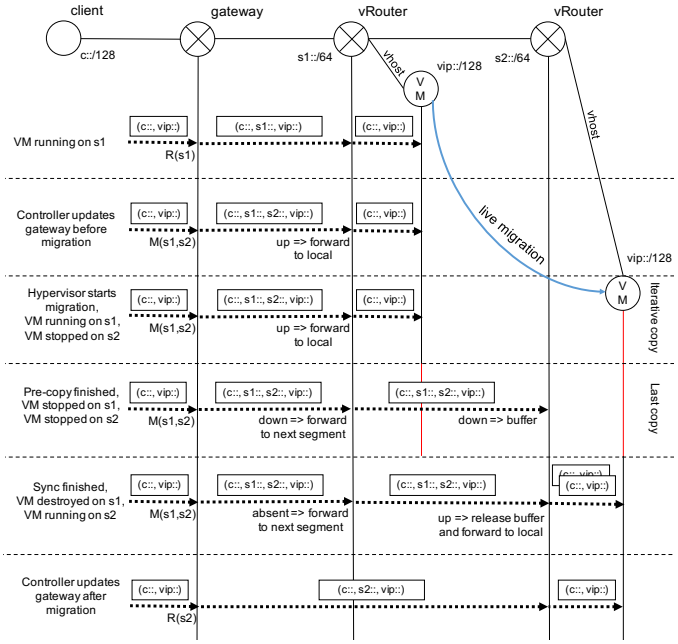

Figure 2. SR Migration: detailed example.

the next segment. The forwarding decision is made by the virtual router by inspecting its routing table, and seeing whether the entry for v corresponds to a virtual (local) interface, whose link-status is up. This corresponds to the END.S behavior [22] with a custom policy for forwarding decisions.

3) Buffer and Forward to Local (bfw): The bfw function inspects the last segment v. If v is not present locally (*i.e.* if the corresponding virtual interface is not ready), it will buffer the packets for further delivery. Otherwise, it flushes any buffered packet to the virtual interface corresponding to v, and forwards the current packet to the last segment v.

This is achieved thanks to a packet buffer in the virtual router, which must be provisioned with a size large enough to handle all potential packets coming during the VM downtime phase. If a VM is expected to receive traffic at rate of r packets/s and to be down during Δt seconds, buffers must be provisioned with a size of $r \cdot \Delta t$ packets.

B. Gateway Operation

For each VIP v, the controller maintains an entry $L(v)$ in the gateway which is either $R(s_1)$ ("running on s_1") or $M(s_1, s_2)$ ("migrating from s_1 to s_2"), depending on the state of v. Conceptually, this forms a mapping $v \mapsto L(v)$, where $L(v)$ is the location of v. Practically, this is implemented by means of routing adjacencies in the FIB of the gateway. The gateway uses the *transit behavior* T.INSERT [22] for v, that is, this routing adjacency triggers insertion of an SR header on packets destined to v. When the gateway receives a packet for v, if the corresponding entry $L(v)$ is $R(s_1)$, then a SR header[2] $(s_1::fw, v)$ is inserted. If the corresponding entry $L(v)$ is $M(s_1, s_2)$, then a SR header $(s_1::fwp, s_2::bfw, v)$ is inserted.

[2]SR headers are shown in reverse packet order (traversal order).

C. Detailed Migration Process

1) Normal Operation: Under "normal operation", that is, when a VM v is running on a server s_1, the gateway is configured to map v to $L(v) = R(s_1)$. Thus, packets destined to v are forwarded to s_1 by means of a single-hop SR tunnel.

2) Migration Operation: Figure 2 provides an example of the operation that occurs when a VM is migrated. When the controller decides to move v from s_1 to s_2, it updates the gateway so as to remap the entry for v to $L(v) = M(s_1, s_2)$. It then queries the hypervisors running on s_1 and s_2 to initiate the live migration process.

As a first step, the VM is stopped on s_2 and running on s_1, while the memory of v is iteratively copied by the hypervisor (out-of-band) from s_1 to s_2. The gateway inserts an SR header $(s_1::fwp, s_2::bfw, v)$ in the packets it receives for v. Since the virtual router on s_1 sees that the virtual interface of the VM is up, upon triggering of the fwp function, it will simply forward these packets to that interface (skipping the s_2 segment).

As a second step, the hypervisor has finished iteratively copying the memory of the VM to the new host, and needs to perform the copy of the last memory pages as well as the CPU state. At this point, the VM is stopped on both s_1 and s_2. The gateway still inserts an SRH $(s_1::fwp, s_2::bfw, v)$ in packets it receives. Now that the VM is stopped on s_1 (which the virtual router at s_1 detects by inspecting the status of the corresponding virtual interface, now in link-down state), the fwp function will simply forward packets to the next segment. Since s_2 is not yet running the VM (which the virtual router at s_2 detects by inspecting the status of the corresponding virtual interface, also in link-down state), the bfw function will temporarily buffer the packet.

As a third step, the VM is started back on host s_2. Before the controller (and thus the gateway) is notified of this, the gateway has a stale mapping $L(v) = M(s_1, s_2)$, and the same SRH is inserted in the packets destined to v. Thus, the first server s_1 forwards these to s_2. Now, the bfw function on s_2 will trigger release of the buffered packets to the virtual interface of the VM, and further packets are forwarded without buffering.

As a fourth step, the controller is notified of the end of the migration, and can update the gateway to map v to $L(v) = R(s_2)$: normal operation resumes, packets reach s_2 directly.

IV. Evaluation

A. Testbed and Methodology

The SR functions described in section III-C have been implemented as VPP plugins. No collaboration between VPP and the hypervisor is needed; rather, forwarding decisions are made from within VPP by inspecting the state of the `vhost-user` interface corresponding to the VM of interest. Buffers for the bfw function are sized to 8192 packets (consuming 16 MB of RAM), allowing to sustain MTU-sized traffic at ≈ 1 Gbps for a typical VM downtime of 100 ms.

A simple testbed is set up with three physical machines, as in figure 1: the first one plays the role of a gateway

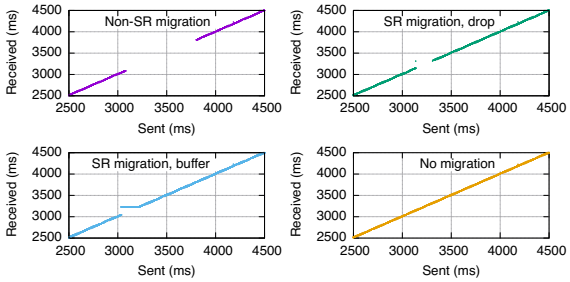

Figure 3. Illustration of SR migration: *ping* experiment for the different mechanisms.

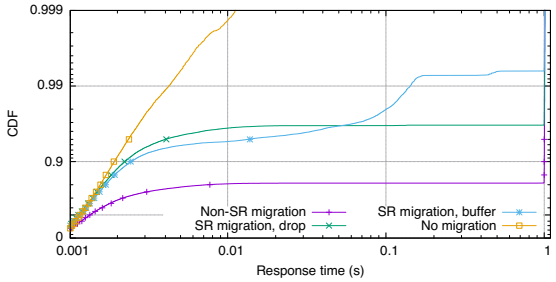

Figure 4. Evaluation of SR migration: response times for the static HTTP workload, $\lambda = 1500\ s^{-1}$ (log-log scales).

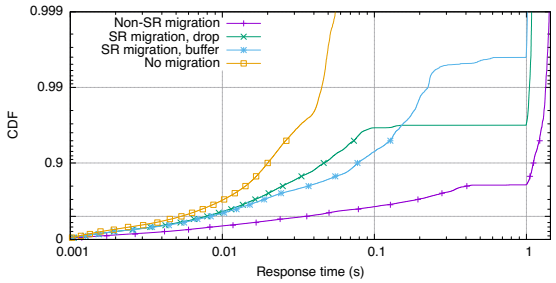

Figure 5. Evaluation of SR migration: response times for the dynamic HTTP workload, $\lambda = 1500\ s^{-1}$ (log-log scales).

gw, and the two other ones s_1, s_2 represent compute nodes, which are candidates to host virtual machines. A client VM v_1 (representing the "outside" of the data-center) is attached to the gateway. A VM v_0 represents a server that will be migrated. That VM is first running on s_1, then migrated from s_1 to s_2. In order to reflect this, the gateway is configured in the $M(s_1, s_2)$ state, *i.e.* it inserts $(s_1{::}fwp, s_2{::}bfw, v_0)$ in all packets destined to v_0. Four mechanisms are compared:

1) no-migration, *i.e.* the VM is not migrated;
2) non-SR migration ("baseline scenario"), *i.e.* the gateway is first in state $R(s_1)$, and once the migration is complete the controller puts the gateway in state $R(s_2)$ (without going through a migration state $M(s_1, s_2)$);
3) SR migration without buffer, *i.e.* the gateway is in state $M(s_1, s_2)$ but packets received on s_2 while the VM is not yet up are dropped instead of buffered;
4) SR migration with buffer ("zero-loss migration" as in section III-C), *i.e.* the gateway is in state $M(s_1, s_2)$ and packets received on s_2 while the VM is not yet up are buffered.

The baseline scenario serves as an illustration of mechanisms such as LISP-based migration [19], wherein packet loss occurs due to the locator mapping being updated only after migration.

B. Ping (illustration)

In order to illustrate the behavior of these four mechanisms, a *ping*[3] is run between the client VM v_1 and the server VM v_0, while v_0 is migrated. One packet is sent every millisecond, and the RTT for each packet is recorded. Figure 3 shows the time at which an echo answer is received as a function of when the corresponding echo request was sent. Without migration, each answer is received approximately 0.1 ms after the corresponding query is sent. With *non-SR migration*, packets were lost during 722 ms (corresponding to the VM downtime, plus the network reconfiguration time), whereas with *SR migration without buffer*, packets were lost for only 174 ms (corresponding to the VM downtime). Finally, with *SR migration with buffer*, 175 packets were buffered while the VM was down, and replied to as soon as the VM went up again.

[3]In its standard implementation, the *ping* utility adapts its sending rate if it sees that probes are not replied to. Here, it was recompiled so that packets are sent with the same rate, no matter how many of them are not answered.

C. HTTP Workload (ants)

To understand the behavior of SR-migration when facing a delay-sensitive workload, a simple evaluation scenario is carried out. The server VM v_0 is set up with an Apache HTTP server – serving a default static file (whose size is 12 KB). As previously, the VM is first running on s_1, then migrated from s_1 to s_2. A traffic generator is attached to the gateway, sending a Poisson stream of 6000 queries with rate $\lambda = 1500\ s^{-1}$, during which the VM is migrated. The experiment is repeated 10 times, for each of the four previously-introduced evaluation scenarios. Response times for all queries are recorded and reported in figure 4.

The majority of queries are executed when the VM is not migrating, and exhibit small response times ($\leq$ 3 ms). The other queries correspond to those that started while the VM was migrating. With *non-SR migration*, TCP SYN packets are lost when the VM is down, but some are also lost after the VM has restarted on s_2, due to the reconfiguration delay. Due to the SYN retransmit delay of 1 s, those queries exhibit a response time $\geq$ 1 s (more than 19% of queries are concerned). With *SR migration without buffer*, TCP SYN packets are lost when the VM is down, but as soon as the VM is up they are successfully transmitted again. This explains why queries suffering from a SYN retransmit are less numerous than with the baseline scenario: less than 4% of queries have a response time greater than 1 s. Finally, with *SR migration with buffer*, TCP SYN packets are buffered while the VM is down. The corresponding response time is simply delayed by the VM downtime ($\approx$ 100 ms in these experiments), rather by the SYN retransmit delay. Less than 0.7% of queries exhibit a response

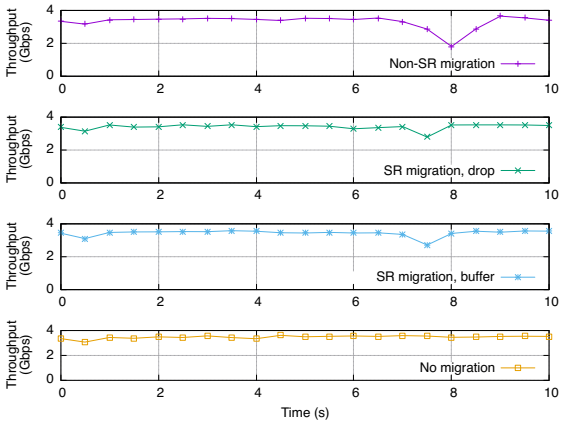

Figure 6. Evaluation of SR migration: instantaneous aggregate throughput, for one *iperf* run with 50 clients.

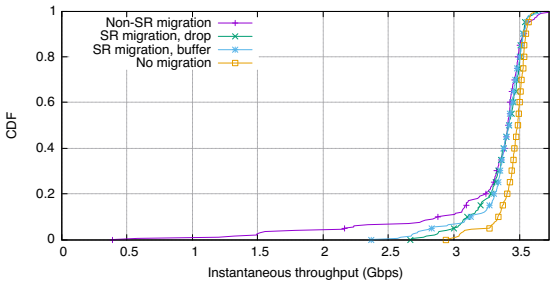

Figure 7. Evaluation of SR migration: distribution of the instantaneous aggregate throughput, over the 10 *iperf* runs with 50 clients.

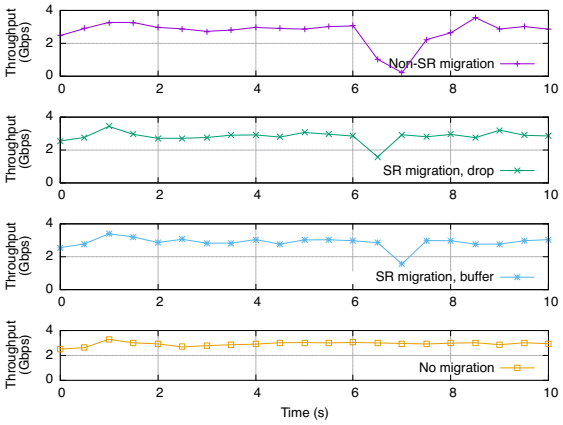

Figure 8. Evaluation of SR migration: instantaneous aggregate throughput, for one *iperf sink* run with 50 clients.

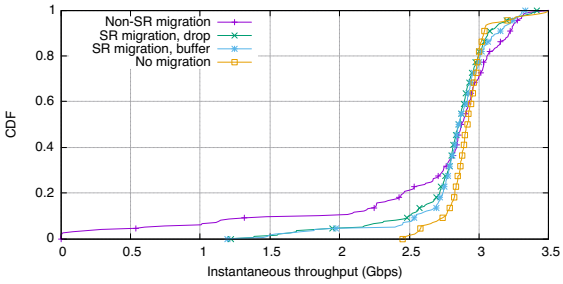

Figure 9. Evaluation of SR migration: distribution of the instantaneous aggregate throughput, over the 10 *iperf sink* runs with 50 clients.

time greater than 1 s, whereas more than 99.2% of queries have a response time lower than 200 ms.

A similar experiment is then performed on a more realistic workload. This time, the workload consists in drawing a random number k (from an exponential distribution with mean $\mathbb{E}[k] = 4$) and serving k copies of the previous static file. This allows to induce variability in the response times, as well as more network traffic. Again, the experiment is repeated 10 times, and client response times are recorded: results are depicted in figure 5. With *non-SR migration*, more than 19% of queries are replied to within 1 s or more. In comparison, less than 4% of queries exhibit a response time greater than 1 s with *SR migration without buffer*, and this drops to 0.4% for *SR migration with buffer*. Furthermore, 99.5% of queries are served within less than 200 ms with the latter mechanism.

D. Iperf Workload (elephants)

The previous set of experiments gave insight on the behavior of the different migration mechanisms in presence of short-lived flows. To understand the influence of SR-migration on longer connections, an experiment with parallel TCP flows is performed. The same experimental platform as in the previous section is used – except that this time, a delay of 2.5 ms is added between the VM v_1 (representing clients) and the VM v_0 (representing the server to be migrated), so as to represent clients outside the data center. An instance of *iperf* with 50 parallel connections is started on the client VM, for 10 seconds (the rationale for using many parallel connections is to smooth the effects of TCP congestion control). In the meantime, the server VM is migrated from machine s_1 to s_2, using the same 4 mechanisms as before. The experiment is repeated 10 times.

Figure 6 depicts the instantaneous throughput (aggregated over the 50 subflows) for one of the 10 runs, collected every 0.5 s. Before and during VM migration, the throughput is rather stable, oscillating around 3.5 Gbps. When the migration ends, around $t = 7s$, there is a drop in throughput, due to the VM downtime ($\approx$ 150 ms in these experiments). While this drop is significant with *non-SR migration* (going to 1.8 Gbps in the run reported in figure 6), it remains reasonably high with *SR migration* (2.7 Gbps in this experiment). The low performance of *non-SR migration* can be explained by TCP reducing its congestion window, due to the large amount of time when no packets are received by the VM. With *SR migration*, this downtime is lower, and thus the congestion window is less reduced. The impact of buffering, versus dropping packets while the VM is not yet up on the new host, seems to be negligible.

In order to quantify these behaviors, figure 7 depicts the distribution of the instantaneous aggregate throughput (collected every 0.5 s), over all the 10 runs. With the *no migration* scenario, the instantaneous throughput stays between 2.9 and

3.7 Gbps. *SR migration* is able to maintain the throughput above 2.3 Gbps at all times, whereas with *non-SR migration* the instantaneous throughput can drop to 0.4 Gbps during the migration phase. With *non-SR migration*, the throughput is lower than 2.2 Gbps for 5% of the time (compared to never with *SR migration*), and lower than 2.9 Gbps for 10% of the time (compared to 5% of the time with *SR migration*).

E. Iperf Sink Workload

The same set of experiments is repeated, except that in this case the *iperf* instance running in the VM is acting as a sink. This can model use-cases where the VM receives a lot of data – for instance, if it is an HTTP proxy or a firewall. In such a scenario, packets lost during the migration process are actual data packets (rather than simple TCP acknowledgements packets, as in the previous set of experiments), thus packet loss is expected to have a greater influence on the overall quality of service.

As previously, figure 8 reports the instantaneous throughput (aggregated over the 50 subflows) for one of the 10 runs, collected every 0.5 s. This time, the drop in throughput when using *non-SR migration* is more critical, going to 0.2 Gbps. Figure 7 depicts the CDF of the instantaneous aggregate throughput as collected every 0.5 s, over the 10 runs. *Non-SR migration* still exhibits low performance, with the 5-th percentile for throughput being 0.6 Gbps, as compared to 1.9 Gbps for *SR migration*. Furthermore, in this case, there is a (small yet perceptible) benefit from buffering packets vs dropping them when using *SR migration*, as each x-th percentile for throughput is greater with *SR migration with buffer* than with *SR migration with drop*.

V. Conclusion

This paper has introduced a mechanism to perform zero-loss VM migration in IPv6 data-centers. Contrary to traditional approaches, which maintain connectivity to migrating VMs reactively by updating the location of VMs after they have migrated, this paper introduces a proactive mechanism, which pre-provisions a logical path through which packets will flow shortly before, during, and shortly after, migration. This is enabled by the use of IPv6 Segment Routing, allowing to steer packets destined to a migrating VM through this logical path. This way, coupling between the data plane and the hypervisor is reduced to a minimum: the only operation that a virtual router performs upon receipt of a packet, is checking whether the corresponding interface is up, and forward the packet locally or to the next segment accordingly. Implementation on a real virtual router, VPP, and evaluation by means of diversified workloads, show that the proposed mechanism is indeed able to provide zero-loss VM migration, with benefits both in terms of TCP throughput and session opening latency.

This paper has focused on VMs processing traffic from/to the outside of the data-center, *i.e.* egress-facing servers or virtual network functions. An interesting question to be investigated is how to use similar techniques to provide seamless migration for VMs handling intra-data-center traffic.

References

[1] C. Clark *et al.*, "Live migration of virtual machines," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*. USENIX Association, 2005, pp. 273–286.

[2] R. Bolla *et al.*, "Seamless and transparent migration for tcp sessions," in *Personal, Indoor, and Mobile Radio Communication (PIMRC), 2014 IEEE 25th Annual International Symposium on*. IEEE, 2014, pp. 1469–1473.

[3] R. W. Ahmad *et al.*, "A survey on virtual machine migration and server consolidation frameworks for cloud data centers," *Journal of Network and Computer Applications*, vol. 52, pp. 11–25, 2015.

[4] C. Ghribi, M. Hadji, and D. Zeghlache, "Energy efficient VM scheduling for cloud data centers: Exact allocation and migration algorithms," in *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*. IEEE, 2013.

[5] D. kakadia, N. Kopri, and V. Varma, "Network-aware virtual machine consolidation for large data centers," in *NDM '13 Proceedings of the Third International Workshop on Network-Aware Data Management*. ACM (6), 2013.

[6] D. Zeng, L. Gu, and S. Guo, "Cost minimization for big data processing in geo-distributed data centers," in *Cloud Networking for Big Data*. Springer, 2015, pp. 59–78.

[7] Y. Wang *et al.*, "Virtual routers on the move: live router migration as a network-management primitive," in *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4. ACM, 2008, pp. 231–242.

[8] M. Mahalingam *et al.*, "Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks," RFC 7348, Aug. 2014. [Online]. Available: https://rfc-editor.org/rfc/rfc7348.txt

[9] P. Garg and Y.-S. Wang, "NVGRE: Network Virtualization Using Generic Routing Encapsulation," RFC 7637, Sep. 2015. [Online]. Available: https://rfc-editor.org/rfc/rfc7637.txt

[10] D. B. Johnson, J. Arkko, and C. E. Perkins, "Mobility Support in IPv6," RFC 6275, Jul. 2011. [Online]. Available: https://rfc-editor.org/rfc/rfc6275.txt

[11] D. Saucez *et al.*, "Designing a deployable internet: the locator/identifier separation protocol," *Internet Computing, IEEE*, vol. 16, no. 6, pp. 14–21, 2012.

[12] S. E. Deering, "Internet Protocol, version 6 (IPv6) specification," in *Requests For Comments*. IETF, 1998, no. 2460.

[13] T. Herbert, "Identifier-locator addressing for network virtualization," https://tools.ietf.org/html/draft-herbert-nvo3-ila-01, 2015.

[14] C. Filsfils *et al.*, "The segment routing architecture," in *2015 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2015, pp. 1–6.

[15] ——, "IPv6 Segment Routing Header (SRH)," Internet Engineering Task Force, Internet-Draft draft-ietf-6man-segment-routing-header-14, 2018, work in Progress.

[16] R. Bradford *et al.*, "Live wide-area migration of virtual machines including local persistent state," in *Proceedings of the 3rd international conference on Virtual execution environments*. ACM, 2007, pp. 169–179.

[17] Q. Li *et al.*, "Hypermip: Hypervisor controlled mobile ip for virtual machine live migration across networks," in *High Assurance Systems Engineering Symposium, 2008. HASE 2008. 11th IEEE*. IEEE, 2008, pp. 80–88.

[18] H. Watanabe *et al.*, "A performance improvement method for the global live migration of virtual machine with ip mobility," in *Proceedings of the Fifth International Conference on Mobile Computing and Ubiquitous Networking (ICMU 2010)*, vol. 94, 2010, pp. 1–6.

[19] P. Raad *et al.*, "Achieving sub-second downtimes in large-scale virtual machine migrations with lisp," *IEEE Transactions on Network and Service Management*, vol. 11, no. 2, pp. 133–143, 2014.

[20] U. Kalim *et al.*, "Seamless migration of virtual machines across networks," in *Computer Communications and Networks (ICCCN), 2013 22nd International Conference on*. IEEE, 2013, pp. 1–7.

[21] D. M. F. Mattos and O. C. M. B. Duarte, "Xenflow: Seamless migration primitive and quality of service for virtual networks," in *Global Communications Conference (GLOBECOM), 2014 IEEE*. IEEE, 2014, pp. 2326–2331.

[22] C. Filsfils *et al.*, "SRv6 Network Programming," Internet Engineering Task Force, Internet-Draft draft-filsfils-spring-srv6-network-programming-05, 2018, work in Progress.