Live Migration of an Entire Network (and its Hosts)

Eric Keller[†], Soudeh Ghorbani[*], Matt Caesar[*], Jennifer Rexford[‡]
[†] *University of Colorado* [*] *UIUC* [‡] *Princeton University*

ABSTRACT

Live virtual machine (VM) migration can move applications from one location to another without a disruption in service. However, applications often consist of multiple VMs and rely on the state of the underlying network for basic reachability, access control, and QoS functionality. Rather than migrating an individual VM, we show how to migrate an *ensemble*—the VMs, the network, and the management system—to a different set of physical resources. Our LIME (LIve Migration of Ensembles) design leverages recent advances in Software Defined Networking (SDN) for a clear separation between the controller and the data-plane state in the switches. Transparent to the application running on the controller, LIME clones the data-plane state to a new set of switches, and then incrementally migrates the traffic sources (*e.g.,* the VMs). During this transition, both networks deliver traffic and LIME maintains synchronized state. Experiments with our initial prototype, built on the Floodlight OpenFlow controller, suggest that network migration does not have to be a disruptive, middle-of-the-night maintenance event, but can become an integral network management mechanism completely transparent to applications.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Network Operating Systems*

General Terms

Design, Experimentation, Management

Keywords

Virtualization, Migration, Software-defined networking

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Hotnets '12, October 29–30, 2012, Seattle, WA, USA.
Copyright 2012 ACM 978-1-4503-1776-4/10/12 ...$10.00.

1. INTRODUCTION

Server virtualization decouples server software from the underlying compute hardware, enabling dynamic repositioning of virtual machines (VMs) to consolidate servers, balance load, perform planned maintenance, and optimize user performance. Once a research novelty, live VM migration is fast becoming an invaluable management tool in public and private clouds [5, 2, 15]. However, a VM rarely acts alone. VMs are often part of multi-tier web applications, with significant interaction between neighboring tiers. Placing these VMs near each other improves performance and reduces network overhead. Migrating a single VM in isolation could lead to significant performance degradation and high bandwidth costs to "backhaul" traffic to the other VMs. As such, migrating an application may require joint migration of a group of related VMs [3, 6].

These applications are often tightly coupled with the underlying network. Beyond providing basic reachability between the VMs, the network applies resource-allocation policies like routing and packet scheduling, and directs traffic through virtual appliances like load balancers and firewalls. Though early cloud offerings gave tenants a relatively simple view of the network, customers increasingly demand more sophisticated functionality. In fact, cloud providers could offer each "tenant" a virtual network with the topology and configuration customized to the application [14, 4]. However, today's VM migration techniques do not handle this network state, limiting VM migration to a single subnet or VLAN, or requiring manual intervention to configure the network devices at the new location.

1.1 The Case for Ensemble Migration

With applications' growing dependence on network state, we believe the network should migrate along with the VMs. Live migration of all (or part) of a network *ensemble* would enable or simplify many management tasks. We discuss some examples here.

Moving between cloud providers: Ensemble migration can allow applications to move between different cloud providers. This can be used by customers to

change which provider is hosting their application. Or it can be used by the cloud provider to transparently offload a given tenant to a different cloud provider to satisfy demand without turning away customers.

Reallocation of resources to reduce energy use: Ensemble migration can allow a datacenter operator to vacate a given cooling zone by moving everything in that zone to a different location within the datacenter. In doing so, the operator can shut down a given zone temporarily to save on cooling costs.

Remapping virtual networks for improved tenant performance: We envision that in the near future, cloud providers will give tenants an abstraction of a virtual topology of hosts and switches. With live migration, the cloud provider can remap the tenant's virtual network to a mapping that provides better performance as resources are freed up.

Inexpensively reducing side-channels: By periodically migrating entire virtual networks, a cloud provider can limit the time a tenant shares server and network resources with other (possibly adversarial) tenants, as a sort of "moving target defense" against side-channel attacks. This allows the cloud provider to get the financial benefit of a shared and oversubscribed infrastructure as opposed to a more robust solution which statically partition all resources.

Planned maintenance: Migrating a single switch, and its incident links, would enable planned maintenance on network equipment without disrupting existing services.

Imminent failure avoidance : Rather than just let a service die and then recover, with ensemble migration we can more gracefully transition from failing to recovering. In the case of a given region failing (*e.g.,* a datacenter that lost power to one portion and is running on backup power), an operator can move subsets of the ensemble away from failed regions. In the case of nano datacenters which have been proposed as a 'run until complete failure' mechanism, with ensemble migration, an operator can move a nano datacenter from one container to another to improve reliability, performance, and cost. Finally, migration is useful for disaster preparation, such as evacuating a running data center in advance of a hurricane.

Rapid deployment: With ensemble migration, one can "pre-package" ensembles and quickly move them from a staging area into production use with minimal effort.

Troubleshooting: Production networks may not be fully equipped to provide in-depth troubleshooting when errors occur. With ensemble migration, an operator can move an ensemble (or portion of an ensemble) to a test environment to monitor the execution more closely.

1.2 LIME as a General Management Layer

Ensemble migration can serve as a powerful management tool that cloud administrators can apply liberally, but only if the technique is both general and efficient.

To be *general*, the solution must work correctly across a wide variety of unmodified control-plane software and end-host applications. By providing a general migration layer, the solution can separate out functionality that are common across all control software applications. Rather than requiring each application to incorporate and customize special purpose functions to deal with underlying changes in the physical infrastructure, it is better to provide an abstraction of this functionality for all to use. This is particularly useful in the case of multi-tenant infrastructures, where the control-plane software may be controlled by a different party than the operator of the infrastructure.

To be *efficient*, ensemble migration cannot simply "freeze" the network while installing state in new network devices—the network is simply too important to the running applications. While the earlier VROOM [13] project showed how to perform live migration of a single network element (virtual router running BGP or OSPF), migrating multiple network elements running arbitrary control-plane software, while minimizing VM "backhaul" traffic during the transition, remains an unsolved problem.

To address these challenges we propose LIME (LIve Migration of Ensembles), a general and efficient solution for joint migration of VMs and the network. LIME draws on the recent trend toward Software Defined Networking (SDN), where control-plane software runs on a logically-centralized controller that installs packet-handling rules in the switches. LIME runs on the controller platform, and presents the (unmodified) control application with a single, consistent view of the network while seamlessly migrating switch and server state. LIME incrementally copies parts of the ensemble, while continuing to carry traffic. In fact, two copies of a switch may forward traffic and generate events at the same time.

2. ENSEMBLE MIGRATION IN AN SDN

LIME performs live migration of an ensemble of VMs and switches in a Software Defined Network (SDN). After a brief overview of SDN and the state in the switches, we discuss the challenges of making ensemble migration both general and efficient.

2.1 Software Defined Networking (SDN)

In a Software Defined Network (SDN), a logically-centralized controller manages how the switches forward packets. The controller exchanges control messages with the switches using a standard protocol, such as OpenFlow [10], and offers a (possibly higher-level)

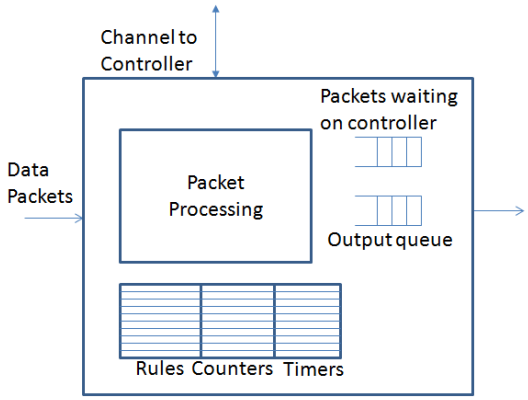

Figure 1: SDN switch state.

API to controller applications. The switches provide a simple packet-forwarding abstraction, such as a table with a prioritized list of rules that match packets on patterns and perform actions. For example, OpenFlow switches match on the input port and packet header fields (e.g., MAC addresses, IP addresses, TCP/UDP ports, VLAN tags, etc.), including wildcards for "don't care" bits, and perform actions like dropping, forwarding, flooding, or directing a packet to the controller for processing.

When migrating an SDN switch, we must think beyond the list of pattern-action rules. A rule may include traffic counters, as well as timers for deleting expired rules, as shown in Figure 1. Each switch has an ordered, reliable channel with the controller. In addition to (un)installing rules, the control channel is useful for querying traffic statistics, sending data packets, and learning about topology changes. The control messages experience delay, so a good API includes a way to notify the controller when the switch has applied a set of commands (e.g., the OpenFlow "barrier" message). In addition to traditional packet queues, the switch may also buffer packets awaiting further instruction from the controller (e.g., to handle "send to the controller" actions).

2.2 Ensemble Migration Challenges

Inspired by live VM migration, ensemble migration could proceed in three steps: (i) iteratively copy the VM and switch state, (ii) freeze the old network for the final state transfer, and (iii) start the new network. In this solution, only one network handles data traffic and interacts with the controller at a time. However, freezing all of the VMs for a final copy of all of the VM and switch state would lead to long outages to finish transferring all of that state. While many datacenter scale services are designed to tolerate component failure, they are not typically designed to withstand failure of entire collections of servers. Instead, LIME should *allow individual VMs to start running in the new location, while the rest of the state transfer continues.*

Yet, VMs in the new location need a network to communicate. LIME could conceivably "freeze and copy" a migrating switch by (i) copying the switch state from one physical switch to another, (ii) freezing for the final state transfer, and (iii) simultaneously starting the switch in the new location and disabling the switch in the old location. To ensure that data traffic flows during the transition, LIME can install tunnels between "neighboring" switches that temporarily reside in different networks. However, temporarily freezing an entire switch can lead to a period of complete packet loss. In addition, when some switches have migrated and others have not, a packet may traverse a tunnel between the two networks at many hops in its journey. Even if two VMs both reside in the new network, the path between them may traverse switches still running in the old network, requiring multiple traversals across the tunnels. To minimize packet loss and backhaul traffic, LIME should *allow two physical switches to act a single switch at the same time.*

Running the old and new switches at the same time avoids down time and ensures that VMs in the same network communicate entirely within that network, and traffic between VMs in different networks crosses the boundary only once. However, during migration, LIME must construct a consistent view of a single switch, even though applications may have (unspecified) dependencies on the ordering of events. This problem may seem similar to the recent "consistent updates" work, which ensures that a packet in flight experiences a single network policy at each hop in its journey [11]. However, in contrast, LIME must satisfy *application-specific* dependencies between *multiple* packets that traverse *different* instances of the same switch during a migration.

3. LIME ARCHITECTURE

LIME (LIve Migration of Ensembles) is a general and efficient solution for joint migration of VMs and the network. LIME builds upon virtualization technologies and SDN to efficiently migrate subcollections of components at a time.

As shown in Figure 2, LIME provides a general layer that gives network operators the freedom to manage the network while eliminating the need for controller applications to deal with these changes. Instead, it is the responsibility of LIME and the underlying virtualization layer to maintain the abstraction provided to each of the virtual networks. LIME enables network operators (or automated processes) to initiate an ensemble migration through an API for specifying which elements to collectively migrate, and where to migrate them. LIME works closely with a network virtualization layer below (such as FlowVisor [12] or FlowN [7]) to allocate re-

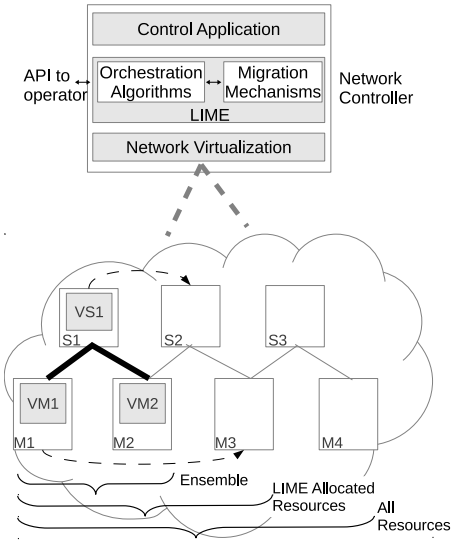

Figure 2: LIME architecture with example. The virtual network consists of two VMs and a virtual switch which are presently mapped to physical devices within a larger infrastructure. To migrate one VM and one virtual switch to different physical devices, LIME works with the virtualization layer to extend the resources allocated to the virtual network during the migration.

sources for the migration process. Importantly, LIME masks the effects of migration from controller applications. These applications are presented with a network abstraction that remains constant throughout the migration, allowing them to continue operating in a seamless fashion.

To migrate an ensemble from one location to another, LIME performs a sequence of operations to network state. In particular, LIME first *clones* one or more switches, and then iteratively *migrates* VMs to new locations. This way, ingress traffic (from a VM) is processed entirely within the network where that VM is located and only gets tunneled to the other network when the destination VM is in a different network – reducing backhaul traffic. When the destination VM is also in the new network, the traffic does not need to traverse a longer path through the old switches. Once all VMs are migrated, no traffic flows through the original switches, so they can be deleted.

In performing this migration, LIME (in conjunction with the virtualization layer) must maintain the logical connections as seen by the control application (*i.e.,* the virtual links between virtual hosts and virtual switches). To do this, LIME must perform a sequence of translations of packet-forwarding rules. In particular, LIME alters the rule's matching patterns and actions to maintain the logical connection as seen by the control application — either altering the virtual port to be the physical port (which can change during migration) or by creating a tunnel when the neighbor is not directly connected. LIME must also go beyond simple rule modification as switches can be cloned and therefore traffic from two instances of a cloned switch to a VM must be 'merged'. The rule translation mechanism in LIME takes care of this by duplicating rules when a rule matches on an input port where traffic can now come from two locations.

Finally, LIME consists of a collection of primitives that ensure certain ordering conditions on operations are met, allowing the preservation of semantics of packet forwarding and control application behavior. We discuss this in the next section.

4. CONSISTENT VIEW OF A SWITCH

To perform live migration without compromising correctness, LIME must hide the effects of migration from end hosts and controller applications, even when multiple instances of a switch carry traffic and generate events. In this section we describe requirements toward this goal and initial steps taken in LIME to respect (unspecified) dependencies between packets, and preserve the rich API to controller applications. As future work, we will formalize these requirements with models and provide proofs of correctness. To simplify the discussion, we assume a switch undergoing migration consists of two switch instances, though in practice multiple instances may coexist.

4.1 Respecting Inter-Packet Dependencies

Data packets must be handled in a way that could have happened in a migration-free setting.

In a best-effort network, packets may experience delay, loss, or out-of-order delivery, even in the absence of migration. As such, LIME does not need to deliver all the packets at exactly the same time as in any particular run of a migration-free network. Yet, LIME must respect (unspecified) dependencies between packets. LIME ensures that the new switch instance starts with an up-to-date snapshot of the rules, but we must also address the various ways rules can change during the migration process. We ensure that only LIME can modify the rules, and that LIME respects packet dependencies when communicating with the switches to update the rules and handle events:

Prevent the switches from changing the rules: While a migration is in progress, the switch instances should *not* change rules autonomously. For example, switches should not apply soft timeouts that delete rules after a period of inactivity, since one switch instance might delete a rule when the other does not. LIME can still offer controller applications an API with soft timeouts, as discussed in Section 4.2.

Respect packet dependencies while updating rules: The controller cannot ensure that both switches apply new commands at exactly the same time. Unfortunately, some packets have (unspecified) application dependencies. For example, one packet may trigger a response from the receiver, and these two packets may traverse different instances of the same switch. LIME must prevent any inconsistent handling of these packets (e.g., if the first packet was handled by a newly-installed rule, the second should not be handled by an older rule). Hence, during an ongoing migration, LIME updates rules in two phases. In the first phase, LIME installs a rule with the same pattern, but with a "drop" (or "send to controller") action to discard (or serialize) the packets. Once LIME knows that both switch instances have installed the new rule (*e.g.,* through the use of a barrier), phase two installs the rule with the action specified by the application. This ensures that any dependent packet is dropped (or sent to the controller) or processed appropriately by the updated rule. Note that the packet loss (or extra controller traffic) only affects traffic matching the pattern in the updated rule, and only during phase 1.

Respect packet dependencies when receiving events: Each switch instance sends control messages (e.g., "packet in" events) to the controller over an ordered, reliable channel. LIME must merge the control messages from the two switches, while respecting (unwritten) dependencies. Fortunately, most packets do not have dependencies, and could easily be reordered in a best-effort network; for these packet-in events, LIME can safely merge control messages from the two switches into a single stream. However, dependencies can arise if a rule has *multiple* actions—both "send to the controller" and "forward". The forwarded packet could reach the receiver, triggering a response packet that might traverse a different instance of the same switch. A packet-in event triggered by the response packet could reach the controller before the first one—something that would never happen with a single switch. To serialize these events during migration, LIME replaces any such rule with a single "send to the controller" action, and handles the packet accordingly.

4.2 Preserving Application Semantics

Control applications should not require modification to work correctly under ensemble migration.

Controller applications should be shielded from the ensemble migration taking place beneath them. The applications should continue to enjoy a rich API that works correctly throughout the migration, with LIME combining the results from multiple switches and emulating any necessary API features:

Combining information from both switch instances: To preserve the illusion of a single switch, LIME must combine information from both switch instances. If the application issues a "barrier" command (to receive notification when a set of control commands complete), LIME should wait to receive notification from both switches before notifying the application. If a link fails at one switch instance, LIME should report a link failure to the application; similarly, if either switch instance fails, LIME should report a switch failure. If the application queries traffic counters, LIME should query both switches and sum the results; furthermore, after migration completes, LIME must maintain traffic statistics from the old switch (or combined results from the two switches) to answer future queries correctly. A key requirement is that the data coming from the switches *can* be combined correctly. Some statistics (e.g., 95th-percentile traffic load on a link, something that is expensive to compute even on a single switch) cannot be combined across links on two different switches. Fortunately, the many important sources of information on today's switches are easily combined, and future switch features can be designed to simplify aggregation of information from multiple switches.

Retaining a rich control API: Control applications benefit from having a rich API that can, and arguably should, differ from the low-level interface to the switches. As discussed above, LIME does not use certain switch features (e.g., soft timeouts) during the migration process, to ensure packet dependencies are respected. This does not mean that controller applications cannot use these features. If the API to controller applications supports soft timeouts, LIME can realize this abstraction during the transition by (for example) polling the traffic counters on the switches and deleting the rule in *both* switches if no traffic matches the rule in *either* switch for a period of time. Similarly, if the API supports rules that simultaneously send a packet to the controller and forward to an output port, LIME can install a rule with a "send the controller" action, and apply the appropriate policy to the packet at the controller, without the application's knowledge.

We envision that SDNs will evolve to offer controller applications an API with higher-level programming abstractions, deviating more and more from the low-level API to the underlying switch hardware. Future SDN platforms may completely shield the programmer from nitty-gritty details like timeouts and rules. In our future work, we plan to explore how LIME can capitalize on this trend to simplify support for ensemble migration.

5. PROTOTYPE AND EVALUATION

We implemented a prototype of LIME on top of the Floodlight controller [8]. LIME is implemented as a layer residing between the Floodlight NOS and the controller applications. As part of this implementation, we expose an API to enable the network operator to is-

sue migration commands. With this prototype we are able to provide some initial insights into the value of LIME. Overall, evaluation with our initial prototype suggests that LIME is capable of performing migration with substantially reduced outage time (measured by packet loss), without imposing much overhead.

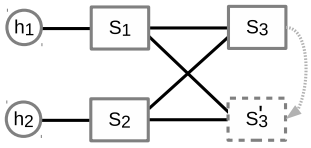

Figure 3: Topology used for experiments.

We evaluate the benefits of the switch cloning supported in LIME (as compared to freeze-and-copy) by deploying a network (shown in Figure 3) within the Mininet HiFi (High Fidelity) emulator [9]. In this experiment, hosts h_1 and h_2 are communicating via switches s_1, s_3, and s_2. The controller application installs rules on each of the switches to provide a path between the hosts. To measure performance of migrating the network in isolation, h_1 continually pings h_2 with 64-byte packets while performing a migration of s_3. Each experiment is repeated 10 times, with results reported as an average. As Table 1 shows, packet loss is 0% with LIME while it is quite considerable with *freeze-and-copy*.

Table 1: Packet loss during migration.

Approaches	Average packet loss	Std dev	Max	Min
LIME (clone)	0	0	0	0
freeze-and-copy	32	4.22	40	30

As LIME is an extra layer in software, there is going to be extra processing overhead in the controller. To evaluate this overhead, we used the *cbench* controller benchmarking tool [1]. We run cbench in "throughput" mode, emulating 10 switches, with 100 hosts attached to each switch. Our (unoptimized) LIME prototype is able to process 9046.16 packet-in events per second, as compared to 9702.34 with Floodlight without LIME – an overhead of only about 7%.

6. CONCLUSIONS AND FUTURE WORK

Live ensemble migration can be a powerful management tool for network operators. Leveraging recent advances in SDN, with LIME, we show how to migrate an entire ensemble in a way that both supports arbitrary controller application software and efficiently orchestrates the migration process. Our evaluation with our initial prototype shows the promise that ensemble migration can be an integral tool rather than a rate maintenance event.

As future work, we plan to explore algorithms which make use of the general framework for orchestrating a migration. This includes determining the optimal grouping of components, the best migration approach to use for that group, the migration order across groups, and optimizing migration based on appropriate network measurements. In addition, we plan to integrate and investigate technologies like redundancy elimination to reduce the overhead of copying the state for multiple related VMs. Finally, we plan to extend LIME for other management needs, such as cloning ensembles for the purposes of failover or load balancing.

7. REFERENCES

[1] cbench OpenFlow controller benchmark. See `http://www.openflow.org/wk/index.php/Oflops`.

[2] NTT, in collaboration with Nicira Networks, succeeds in remote datacenter live migration, August 2011. `http://www.ntt.co.jp/news2011/1108e/110802a.html`.

[3] S. Al-Kiswany, D. Subhraveti, P. Sarkar, and M. Ripeanu. VMFlock: Virtual machine co-migration for the cloud. In *High Performance Distributed Computing*, 2011.

[4] BigSwitch Networks. Perspectives: Networking needs a VMware. `http://www.bigswitch.com/wp/about-us/`.

[5] Cisco and VMWare. Virtual machine mobility with VMware VMotion and Cisco data center interconnect technologies, 2009.

[6] U. Deshpande, X. Wang, and K. Gopalan. Live gang migration of virtual machines. In *High Performance Distributed Computing*, 2011.

[7] D. Drutskoy, E. Keller, and J. Rexford. Scalable network virtualization in software-defined networks. *IEEE Internet Computing: Special Issue on Virtualization (to appear)*, Mar/Apr 2013.

[8] Floodlight OpenFlow Controller. `http://floodlight.openflowhub.org/`.

[9] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown. Reproducible network experiments using container based emulation. In *Proc. CoNEXT (to appear)*, 2012.

[10] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling innovation in campus networks. *SIGCOMM Computer Communications Review*, 38(2), 2008.

[11] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker. Abstractions for network update. In *ACM SIGCOMM*, August 2012.

[12] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar. Can the production network be the testbed? In *Operating Systems Design and Implementation*, October 2010.

[13] Y. Wang, E. Keller, B. Biskeborn, J. van der Merwe, and J. Rexford. Virtual routers on the move: Live router migration as a network-management primitive. In *ACM SIGCOMM*, 2008.

[14] K. C. Webb, A. C. Snoeren, and K. Yocum. Topology switching for data center networks. In *Hot-ICE Workshop*, March 2011.

[15] D. Williams, H. Jamjoom, and H. Weatherspoon. The Xen-blanket: Virtualize once, run everywhere. In *ACM European Conference on Computer Systems (Eurosys)*, April 2012.