# Towards the InfiniBand SR-IOV vSwitch Architecture

Evangelos Tasoulas*, Ernst Gunnar Gran*, Bjørn Dag Johnsen†, Kyrre Begnum‡ and Tor Skeie*
*Simula Research Laboratory †Oracle Corporation ‡Oslo and Akershus University College
{vangelis, ernstgr, tskeie}@simula.no bjorn-dag.johnsen@oracle.com kyrre.begnum@hioa.no

***Abstract*—To meet the demands of the Exascale era and facilitate Big Data analytics in the cloud while maintaining flexibility, cloud providers will have to offer efficient virtualized High Performance Computing clusters in a pay-as-you-go model. As a consequence, high performance network interconnect solutions, like InfiniBand (IB), will be beneficial. Currently, the only way to provide IB connectivity on Virtual Machines (VMs) is by utilizing direct device assignment. At the same time to be scalable, Single-Root I/O Virtualization (SR-IOV) is used. However, the current SR-IOV model employed by IB adapters is a *Shared Port* implementation with limited flexibility, as it does not allow transparent virtualization and live-migration of VMs.**

**In this paper, we explore an alternative SR-IOV model for IB, the virtual switch (*vSwitch*), and propose and analyze two vSwitch implementations with different scalability characteristics. Furthermore, as network reconfiguration time is critical to make live-migration a practical option, we accompany our proposed architecture with a scalable and topology agnostic dynamic reconfiguration method, implemented and tested using OpenSM. Our results show that we are able to significantly reduce the reconfiguration time as route recalculations are no longer needed, and in large IB subnets, for certain scenarios, the number of reconfiguration subnet management packets (SMPs) sent is reduced from several hundred thousand down to a single one.**

## I. Introduction

There is a lot of work going on, both in academia and the industry, to make cloud computing capable of offering High Performance Computing (HPC). With HPC-as-a-Service, traditional HPC users can save capital expenditure, while new user groups that cannot afford to own a private HPC cluster, can get on-demand access. However, the overhead imposed by virtualization combined with the extreme performance demands of HPC kept this idea immaterialized for a long time. During the last ten years, the situation has improved considerably as CPU overhead has been practically removed through hardware virtualization support [1], [2]; memory overhead has been significantly reduced by virtualizing the Memory Management Unit; storage overhead has been reduced by the use of fast SAN storages or distributed networked file systems; and network I/O overhead has been reduced by the use of device passthrough techniques like Single Root Input/Output Virtualization (SR-IOV) [3]. It is now possible for clouds to accommodate virtual HPC (vHPC) clusters using high performance interconnect solutions and deliver the necessary performance [4], [5], [6].

InfiniBand (IB) [7] is an interconnection network technology offering high bandwidth and low latency, thus, is very well suited for HPC and other network demanding workloads. IB accelerates 224 HPC systems in the TOP500 supercomputers list as of November 2014 [8], 44.8% of the list.

To ensure efficient virtualization, while maintaining high bandwidth and low latency, modern IB Host Channel Adapters (HCAs) support SR-IOV. Nevertheless, to achieve transparent live migration of Virtual Machines (VMs) assigned to IB HCAs using SR-IOV has proved to be challenging [5], [6], [9]. Each InfiniBand connected node has three different addresses (LID, GUID, GID – further discussed in section II). When a live migration happens, one or more of these addresses changes. Other nodes communicating with the VM-in-migration lose connectivity and try to find the new address to reconnect to by sending Subnet Administration (SA) path record queries to the IB Subnet Manager (SM) [7].

In [10] we showed that by using *address caching*, one do not have to send repetitive SA queries to reconnect once a VM is live migrated. However, in order to allow a VM to be moved and benefit from such a caching mechanism, each VM should be bound to a dedicated set of IB addresses that follows the VM when the VM migrates. With the current IB SR-IOV *Shared Port* implementation [11], the VMs running on the same hypervisor [12], share one LID address and have dedicated GUID and GID addresses. When a VM with its associated LID is migrated, the connectivity will be broken for the rest of the VMs that share the same LID.

In this paper we propose two implementations of the *Virtual Switch (vSwitch)* [13] architecture with different scalability characteristics, that will allow IB subnets to support transparent virtualization and migration of IB addresses, accompanied with a scalable and topology agnostic dynamic network reconfiguration method to make live migrations of VMs feasible in large vSwitch-based IB subnets.

The rest of the paper is organized as follows: Section II gives background information on Input/Output Virtualization (IOV) and IB addressing schemes, followed by the related work in section III. The IB SR-IOV design overview in section IV emphasizes the pros and cons of the *vSwitch* and the *Shared-Port* architectures. In section V we propose our vSwitch architectures and the dynamic reconfiguration mechanism, followed by an analytical discussion in section VI. We implement a prototype and show the results of the implementation and simulations in section VII, before we conclude in section VIII.

## II. Background

In this section we describe different IOV techniques with a particular focus on SR-IOV. IB addressing schemes are presented as well.

### A. Network I/O Virtualization

IOV is needed to share I/O resources and provide protected access to these resources from the VMs. IOV decouples the logical device, which is exposed to a VM, from its physical implementation [12], [14]. Currently, there are two widespread approaches to IOV, both having their advantages and disadvantages:

*1) Software emulation:* is a decoupled front-end/back-end software architecture. The front-end is a device driver placed in the VM, communicating with the back-end implemented by the hypervisor to provide I/O access. The physical device sharing ratio is high and live migrations of VMs are possible with just a few milliseconds of network downtime [17], but software emulation introduces additional computational overhead.

*2) Direct device assignment:* involves a coupling of I/O devices to VMs, with no device sharing between VMs. Direct assignment, or *device passthrough*, provides near to native performance with minimum overhead. The physical device bypasses the hypervisor and is directly attached to the VM. The downside is limited scalability, as there is no sharing; one physical network card is coupled with one VM. *Single Root IOV (SR-IOV)* allows a physical device to appear through hardware virtualization as multiple independent lightweight instances of the same device. These instances can be assigned to VMs as passthrough devices, and accessed as Virtual Functions (VFs) [3]. The hypervisor accesses the device through a unique (per device), fully featured Physical Function (PF). SR-IOV eases the scalability issue of pure direct assignment. Currently, there is no easy way to live-migrate VMs without a network downtime in the order of seconds when using direct device assignment [15].

HPC interconnection networks rely heavily on hardware offloading and bypassing of the protocol stack and the OS kernel to efficiently reduce latency and increase performance [16]. Thus, currently the only option to provide high performance networking in VMs, is to use a direct device assignment technique. To still be scalable, we, as others working with IB and virtualization [4], [6], [9], chose to use SR-IOV to work with.

Unfortunately, direct device assignment techniques pose a barrier for cloud providers if they want to use transparent live migrations for data center optimization. The essence of live migration is that the memory contents of a VM are copied to a remote hypervisor. Then the VM is paused at the source hypervisor, and the VM's operation is resumed at the destination. When using software emulation methods, the network interfaces are virtual so their internal states are stored into the memory and gets copied as well. Thus the downtime could be brought down to a few milliseconds [17]. In the case of direct device assignment like SR-IOV VFs, the complete internal state of the network interface cannot be copied as it is tied to the hardware [5]. The SR-IOV VFs assigned to a VM will need to be detached, the live migration will run, and a new VF will be attached at the destination. In the case of InfiniBand and SR-IOV, this process will introduce downtime in the order of seconds as discussed by Guay et al. [9], [18]. Moreover, with the currently implemented SR-IOV *Shared Port* model the addresses of the VM will change after the migration, causing additional overhead in the SM and a negative impact on the performance of the underlying network fabric [10].

### B. The InfiniBand Addressing Schemes

InfiniBand uses three different types of addresses [7], [19], [9]. First is the 16 bits Local Identifier (LID). At least one unique LID is assigned to each HCA port and each switch by the SM. The LIDs are used to route traffic within a subnet. Since the LID is 16 bits long, 65536 unique address combinations can be made, of which only 49151 (0x0001-0xBFFF) can be used as unicast addresses. Consequently, the number of available unicast addresses defines the maximum size of an IB subnet.

Second is the 64 bits Global Unique Identifier (GUID) assigned by the manufacturer to each device (e.g. HCAs and switches) and each HCA port. The SM may assign additional subnet unique GUIDs to an HCA port, which is particularly useful when SR-IOV is used.

Third is the 128 bits Global Identifier (GID). The GID is a valid IPv6 unicast address, and at least one is assigned to each HCA port and each switch. The GID is formed by combining a globally unique 64 bits prefix assigned by the fabric administrator, and the GUID address of each HCA port.

## III. Related Work

Guay et al. [9] migrate VMs with SR-IOV VFs. The vGUID of the SR-IOV VF is migrated together with the VM, but the LID address changes. The main goal of their work is to reestablish the communication after a VM has been migrated and the LID address has changed, with the intention to reduce VM migration downtime and avoid reconfiguring the network. Tasoulas et al. [10] migrate VMs with IB VFs and all three addresses, and use a caching mechanism to reestablish connectivity without having to send SA PathRecord queries. A prototype is used to orchestrate the migration process of the IB addresses and the SM is restarted in order to migrate the LID of the VM and trigger the network reconfiguration.

In general, when a lossless network is reconfigured, routes have to be recalculated and distributed to all switches, while avoiding deadlocks. Note that the coexistence of two deadlock free routing functions, the $R_{old}$ and $R_{new}$, during the transition phase from the old to the new one, might not be deadlock free [20]. Zafar et al. [21] discusses the tools and applicable methods on IB architecture (IBA), that would allow the implementation of the *Double Scheme* [22] reconfiguration method. The *Double Scheme* is using Virtual Lanes (VLs) to separate the new and the old routing functions. Lysne et al. [23] use a token that is propagated through the network to mark a reconfiguration event. Before the token arrives on a switch, traffic is routed with the old routing algorithm. After the token arrives and forwarded through the output ports of the switch, the traffic is flowing with the new routing algorithm. The *Skyline* by Lysne et al. [24], speeds up the reconfiguration process by providing a method for identifying the minimum part of the network that needs to be reconfigured. Sem-Jacobsen et al. [25]

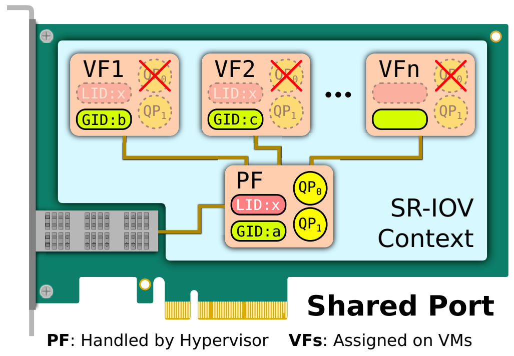

Fig. 1. InfiniBand SR-IOV Shared Port architecture

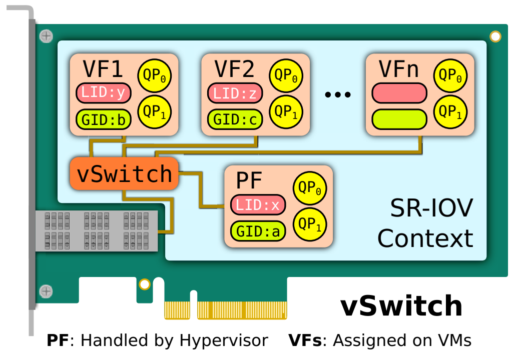

Fig. 2. InfiniBand SR-IOV vSwitch architecture

use the channel dependency graph to create a channel list that is rearranged when traffic needs to be rerouted. The rearranging is happening in such a way, that no deadlocks can occur. Robles-Gómez et al. [26] use close up*/down* graphs to compute a new routing algorithm which is close to the old one, and guarantees that the combination of old and new routing during transition do not allow deadlocks to be introduced. Bermúdez et al. [27] are concerned with the long computation time it takes to compute optimal routing tables in large networks, that consequently delays the IB subnet from becoming operational. They use some quickly calculated, but not optimal, provisional routes and they calculate offline the optimal routes. Since the provisional and the optimal routes are calculated based on the same acyclic graph, deadlock freedom is guaranteed. [27], as well as the rest of the surveyed work, does not consider reconfiguration of dynamic virtualized environments, and in particular does not consider nodes and node IDs that move inside the network.

## IV. InfiniBand SR-IOV Design Overview

The *Shared Port* and *vSwitch* architectures have been suggested by Liss [13]. Only the former one is currently implemented in the IB drivers [11]. In this section, we discuss these two architectures.

### A. SR-IOV Shared Port

The Shared Port architecture is illustrated in Fig. 1. The HCA appears as a single port in the network with a single shared LID and shared Queue Pair[1] (QP) space between the PF and VFs, but multiple GIDs. As shown in Fig. 1, different GIDs are assigned to the VFs and the PF, and the special $QP_0$ and $QP_1$ are owned by the PF. These QPs are exposed to the VFs as well, but the VFs are not allowed to use $QP_0$ (all SMPs coming from VFs towards $QP_0$ are discarded), and $QP_1$ acts as a proxy of the actual $QP_1$ owned by the PF. Shared Port allows for highly scalable data centers that are not limited by the number of VMs, as the LID space is only consumed by physical machines and switches in the network.

[1]A QP is a virtual communication port used by IB applications (consumers) to communicate [7]. $QP_0$ and $QP_1$ are two special purpose QPs, used for IB management packets only.

One shortcoming of the Shared Port architecture is the inability to provide transparent live migration, hindering the potential for flexible VM placement. As each LID is associated with a specific hypervisor, and shared among all VMs residing on the hypervisor, a migrating VM will have its LID changed to the LID of the destination hypervisor. Furthermore, as a consequence of the restricted $QP_0$ access, an SM cannot run inside a VM.

### B. SR-IOV vSwitch

In the vSwitch architecture (Fig. 2) each VF is a complete vHCA, meaning that the VM is assigned a complete set of IB addresses (section II-B) and a dedicated QP space in the hardware. For the rest of the network and the SM, the HCA looks like a switch with additional nodes connected to it; the hypervisor uses the PF and the VMs use the VFs, as shown in Fig. 2. The vSwitch architecture provides transparent virtualization, but at the cost of consuming additional LID addresses. When many LID addresses are in use, more communication paths have to be computed by the SM and more Subnet Management Packets (SMPs) have to be sent to the switches in order to update their Linear Forwarding Tables (LFTs). In particular, the computation of the communication paths might take several minutes in large networks [28]. Moreover, as each VM, physical node, and switch occupies one LID each, the number of physical nodes and switches in the network limits the number of active VMs, and vice versa. Recall that the LID space is limited to 49151 unicast LIDs. Nevertheless, transparent virtualization is a key feature for virtualized data centers with live migration support.

## V. Proposed vSwitch Architecture

Transparent virtualization offered by the vSwitch architecture is important in dynamic virtualized cloud environments. When live migrations take place, each VM should be able to carry with it all of its associated addresses to the destination, something not possible with the Shared Port architecture. In this section, we propose two alternative implementations of the vSwitch architecture with different scalability characteristics, and provide a method for scalable dynamic reconfiguration as VMs are live migrated with their addresses.

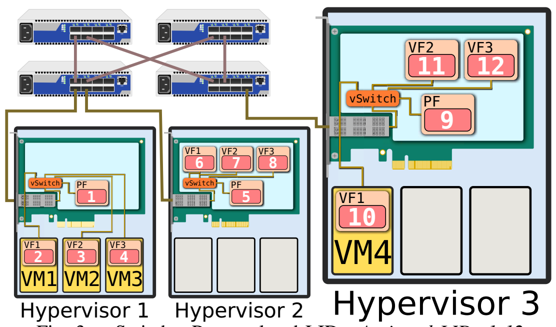

Fig. 3. vSwitch - Prepopulated LIDs. *Assigned LIDs 1-12.*

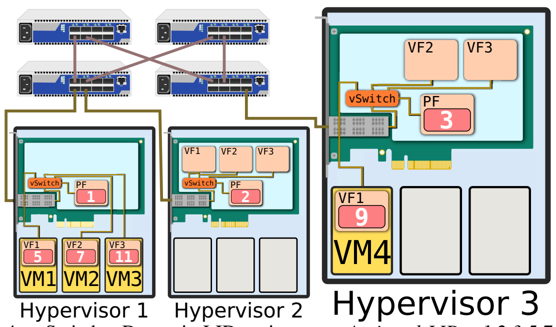

Fig. 4. vSwitch - Dynamic LID assignment. *Assigned LIDs: 1,2,3,5,7,9,11.*

## A. *vSwitch with Prepopulated LIDs*

Our first approach initializes all available VFs with LIDs, even those VFs that are not currently used by any VM, as shown in Fig. 3. In such a scheme, each hypervisor will consume one LID for itself through the PF and one more LID for each additional VF. The sum of all the VFs available in all hypervisors in an IB subnet, gives the maximum amount of VMs that are allowed to run in the subnet. If we assume 16 VFs[2] per hypervisor in the subnet, then each hypervisor consumes 17 LIDs. Then, the theoretical hypervisor limit for a single subnet is ruled by the number of unicast LIDs and is: $\lfloor Topmost\_Unicast\_LID/17 \rfloor) = \lfloor 49151/17 \rfloor = 2891$, and the number of VMs limit is: $2891 \cdot 16 = 46256$. These numbers are actually even smaller since each switch, router, or dedicated SM nodes in the subnet consume LIDs as well. Note that the vSwitch does not need to occupy an additional LID as it can share the LID with the PF.

In a vSwitch architecture with prepopulated LIDs, communication paths are computed for all the LIDs once, when the network is booted. When a new VM needs to be started the system does not have to add a new LID in the subnet, an action that will cause a complete reconfiguration, including the time consuming path computation step [21], [31], [27]. All that needs to be done is to find an available VM slot in one of the hypervisors and use it. An available VM slot is equivalent to an available VF. Another gain of this proposed method is the ability to calculate and use different paths to reach different VMs hosted by the same hypervisor. Essentially, imitating the *LID Mask Control (LMC)* feature to provide alternative paths towards one physical machine, without being bound by the limitation of the LMC that requires the LIDs to be sequential. The freedom to use non-sequential LIDs is particularly useful when a VM needs to be migrated and carry its associated LID to the destination.

On the negative side, the initial computation of the paths will require considerably more time than what it would need without the prepopulation of all LIDs. In the previous example with 16 VFs per hypervisor, when no VMs are running, the initial path computation needs to calculate paths for close to 3000 LIDs. However, the actual paths to be computed are based on more than 49000 LIDs. Also, there is a strict limit on the number of physical nodes in the network and the number of SR-IOV VFs. The summation of the physical nodes (e.g. switches, hypervisors, additional SM nodes) and VFs cannot exceed the unicast LID limit, even if there are no VMs running on the network. On the other extreme, if all of the VFs are occupied by running VMs, there is no option for optimizations by using live migrations, leading to a potentially fragmented network.

[2]Up to 126 VFs are supported on the Mellanox ConnectX®-3 adapters, but 16 are enabled by default [29]. Nonetheless, the max number of VFs may be even smaller as it depends on the PCI Base Address Registers (BAR) size and the available system resources [30].

## B. *vSwitch with Dynamic LID Assignment*

Our second approach dynamically assigns LIDs as illustrated in Fig. 4. With the dynamic LID assignment, the initial path computation will be substantially reduced. Refer to the example given in section V-A, when the network is booting for the first time and no VMs are present, then less than 3000 LIDs will be used for the initial path calculation and LFT distribution. However, when using this method and a new VM is created, a unique free unicast LID has to be used. In this case, a challenge arises because there are no known paths in the network for handling the newly added LID and VM. Computing a new set of paths in order to handle the newly added VM is not an option in a dynamic environment where several VMs may be booted every minute. In large IB subnets, computing a new set of routes can take several minutes, and this procedure would have to repeat each time a new VM is booted.

Fortunately, since we know that all the VFs in a hypervisor share the same uplink with the PF, there is no need to compute a new set of routes. It is only needed to iterate through the LFTs of all the physical switches in the network, copy the forwarding port from the LID entry that belongs to the PF of the hypervisor —where the VM is created— to the newly added LID, and send a single SMP to update the corresponding LFT block of the particular switch.

When comparing the LIDs assigned on VMs on each hypervisor in Fig. 3 and Fig. 4, notice that the LIDs assigned to the VMs in Fig. 3 are sequential, while the LIDs assigned in Fig. 4 are spread. As there is no requirement for sequential LIDs, this layout is a result of VMs being created and destroyed. In the dynamic LID assignment when a new VM is created, the next available LID is used throughout the lifetime of the VM. In an environment with prepopulated LIDs, each VM will inherit the LID that is already assigned to the corresponding VF, and in a network without live migrations, VMs consecutively attached to a given VF will always get the same LID.

The dynamic LID assignment model can resolve the drawbacks of the prepopulated LIDs model described in V-A at the cost of some additional network and runtime SM overhead. Each time a VM is created, the LFTs of all the physical switches in the subnet will need to be updated with the newly added LID associated with the created VM. One SMP per switch is needed to be sent for this operation. The LMC-like functionality is also not available, because each VM is using the same path with its host hypervisor. However, the is no limitation on the total amount of VFs present in all hypervisors, and the number of VFs may exceed that of the unicast LID limit. Of course, not all of the VFs are allowed to be attached on active VMs simultaneously if this is the case, but having more *spare* hypervisors and VFs adds flexibility for disaster recovery and optimization of fragmented networks when operating close to the unicast LID limit.

### *C. Dynamic Reconfiguration with vSwitches*

In a dynamic cloud environment, live migrations should be supported and be scalable. When a VM is migrated and carries its addresses to the destination, a network reconfiguration is necessary. Migration of the virtual or alias GUIDs (vGUIDs), and consequently the GIDs, do not pose a significant burden as they are high level addresses that do not affect the underlying IB routing. For the migration of the vGUID, an SMP has to be sent to the destination hypervisor in order to set the vGUID that is associated with the incoming VM, to the VF that will be assigned to the VM when the migration is completed. However, migration of the LID is not that simple, because the routes have to be recalculated and the LFTs of the physical switches reconfigured. Recalculation of the routes needs a considerable amount of time in the order of minutes on large subnets, posing scalability challenges that may render VM migrations unusable.

The vSwitch has the property that all the VFs of an HCA share the link with the PF. Our topology agnostic dynamic reconfiguration mechanism utilizes this property in a novel way to make the reconfiguration highly efficient in dynamic environments. The LID reconfiguration time is minimized by completely eliminating the path computation, and drastically reducing the path distribution. The method differs slightly for the two proposed vSwitch architectures, but the basis is the same, and as presented in algorithm 1 involves two steps:

a **Update of the LIDs in the participating hypervisors:** one SMP is sent to each of the hypervisors that participate in the live migration, instructing them to set/unset the LID to the corresponding VF.
b **Update of the LFTs on the physical switches:** one or a maximum of two SMPs are sent on one or more switches, forcing them to update their corresponding LFT entries to reflect the new position of the migrated VM.

*1) Reconfiguration with Prepopulated LIDs:* For the vSwitch architecture with Prepopulated LIDs, paths exist for all of the LIDs even if VMs are not running. In order to migrate the LID

**Algorithm 1** Migrate VM and reconfigure the network.

```
 1: procedure UpdateLFTBlock($LFTBlock$, $Switch$)
 2:     // If the LFT block needs to be updated send SMP on the switch to
 3:     // update the LFTBlock. When Swapping LIDs (V-C1), 1 or 2 of all
 4:     // the LFT Blocks may need to be updated per switch. When copying
 5:     // LIDs (V-C2), only 1 of all the LFT Blocks may need to be updated
 6:     // per switch.
 7:     if $LFTBlock$ in $Switch$ needs to be updated then
 8:         Send SMP on $Switch$ to update $LFTBlock$
 9:     end if
10: end procedure
11: procedure UpdateLFTBlocksOnAllSwitches
12:     /* Iterate through all LFTBlocks on all Switches
13:     * and update the LFTBlocks if needed. */
14:     for $LFTBlock$ in $All\_LFTBlocks$ do
15:         for $sw$ in $All\_switches$ do
16:             UpdateLFTBlock($LFTBlock$, $sw$)
17:         end for
18:     end for
19: end procedure
20: procedure MigrateVM(VM, DestHypervisor)
21:     Detach IB VF from $VM$
22:     Start live migration of $VM$ to the $DestHypervisor$
23:     /* Reconfiguration of the network is following */
24:     // The migration procedure of the LID address slightly
25:     // differs in V-C1 and V-C2.
26:     /* Step described in enumeration V-C-a */
27:     Migrate the IB addresses of $VM$
28:     /* Step described in enumeration V-C-b */
29:     UpdateLFTBlocksOnAllSwitches
30: end procedure
31: procedure Main
32:     MigrateVM(VM_to_be_Migrated, toHypervisor)
33: end procedure
```

and keep the balancing of the initial routing, what needs to be done is to swap two LFT entries on all switches; The entry of the LID that is assigned to the VM, with the LID of the VF that is going to be used at the destination hypervisor after the live migration is completed. If VM1 with LID 2 in Fig. 5 needs to be migrated from *hypervisor 1* to *hypervisor 3*, and VF3 with LID 12 on *hypervisor 3* is available and decided to be attached to VM1, the LFTs of the upper left switch in Fig. 5 should be changed as shown. Before the migration LID 2 was forwarded through Port 2, and LID 12 was forwarded through Port 4. After the migration LID 2 is forwarded through Port 4, and LID 12 is forwarded through Port 2. In this case, only one SMP needs to be sent for this update because LFTs are updated in blocks of 64 LIDs per block (further explained in section VI), and both LID 2 and 12 are part of the same block that includes the LIDs 0 - 63. If the LID of VF3 on hypervisor 3 was 64 or greater, then two SMPs would need to be sent as two LFT blocks would have to be updated: the block that contains LID 2 (the VM LID) and the block that contains the LID to be swapped that is bigger than 63. The same swapping procedure is used to update all the switches that need to be updated (as explained in more detail in section VI).

*2) Reconfiguration with Dynamic LID Assignment:* For the vSwitch architecture with Dynamic LID assignment, the path of a VF follows the same path as the path of the corresponding

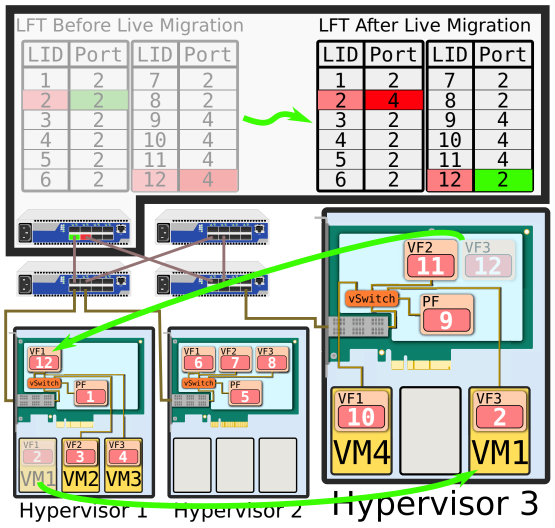

Fig. 5. LFT Updates - LID Swapping. **LIDs** in the LFT tables are forwarded through **Ports**.

PF of the hypervisor where the VM is currently hosted. When a VM moves, the system has to find the LID that is assigned to the PF of the destination hypervisor, and iterate through all the LFTs of all switches and update the path for the VM LID with the path of the destination hypervisor. In contrast to the LID swapping technique that is used in the reconfiguration with prepopulated LIDs, only one SMP needs to be sent at all times to the switches that need to be updated, since there is only one LID involved in the process.

## VI. Analysis of the Reconfiguration Mechanism

In this section we analyze our novel reconfiguration procedure and compare it with a traditional reconfiguration method, a method that would initiate a full network reconfiguration for each network change.

### A. Traditional Reconfiguration Cost

The time, $RC_t$, needed for a full traditional ReConfigration method is the sum of the time needed for the Path Computation, $PC_t$, plus the time needed for the LFTs Distribution, $LFTD_t$, to all switches, as shown in equation 1:

$$RC_t = PC_t + LFTD_t \tag{1}$$

The computational complexity of the paths is polynomially growing with the size of the subnet, and $PC_t$ is in the order of several minutes on large subnets[3] [28].

After the paths have been computed, the LFTs of the switches have to be updated. The LFT distribution time $LFTD_t$ grows linearly with the size of the subnet and the amount of switches. The LFTs are updated on blocks of 64 LIDs so in a small subnet with a few switches and up to 64 consumed LIDs, only one SMP needs to be sent to each switch during path distribution. On the other extreme, in a fully populated IB subnet with 49151 LIDs consumed, 768 SMPs per switch are needed to be sent during path distribution.

The SMPs can use either directed routing or destination based routing. When using directed routing, each intermediate switch has to process and update the headers of the packet with the current hop pointer and reverse path before forwarding the packet to the next hop [7]. In the destination based routing, each packet is forwarded immediately. Naturally, directed routing adds latency to the forwarded packets. Nevertheless, directed routing is used by OpenSM for all SMPs. This is necessary for the initial topology discovery process where the LFTs have not been distributed yet to the switches, or when a reconfiguration is taking place and the routes towards the switches are changing.

Let $n$ be the number of switches in the network; $m$ the number of all LFT Blocks that will be updated on each switch, determined by the number of consumed LIDs; $k$ the average time needed for each SMP to traverse the network before reaching each switch[4]; and $r$ the average time added for each SMP due to the directed routing. Assuming no pipelining, if we break the LFT distribution time $LFTD_t$ further down we get equation 2:

$$LFTD_t = n \cdot m \cdot (k + r) \tag{2}$$

From equation 1 and 2, we get equation 3:

$$RC_t = PC_t + n \cdot m \cdot (k + r) \tag{3}$$

In large subnets $PC_t \gg LFTD_t$, even though the $LFTD_t$ becomes larger when more LIDs, and consequently more LFT Blocks per switch $m$ are used, and when more switches $n$ are present in the network. The $n \cdot m$ part in equation 2 and 3 defines the total number of SMPs that needs to be sent for the reconfiguration.

### B. Reconfiguration Cost when Live Migrating with vSwitches

Using traditional reconfiguration techniques would render VM migrations unusable. In large subnets, the $PC_t$ in equation 3 becomes very large and dominates $RC_t$. If a live migration triggered a full traditional reconfiguration, it would take several minutes to complete.

In our reconfiguration mechanism when vSwitches are used, we eliminate $PC_t$ since we use the already calculated paths to swap or copy LID entries in the LFT of each switch. Furthermore, there is no need to send $m$ SMPs per switch, because when a VM is migrated only one or a maximum of two LIDs are affected depending on which of the proposed vSwitch schemes is used, regardless of the total number of LFT blocks. As a result, only $m' \in \{1, 2\}$ SMPs are needed to be sent to the switches for each migration ($m' = 2$ if the two LID entries are not located in the same LFT block when the LIDs are prepopulated, otherwise $m' = 1$). Also there are certain cases that $0 < n' < n$ switches will need to be updated.

[3]Topology and chosen routing algorithm can have very diverse effects on the time needed to compute the paths.

[4]Switches closer to the SM can be reached faster as they traverse less intermediate switches and cables.

Consider the case that VM1 was migrated from *Hypervisor 1* to *Hypervisor 2* in Fig 5. If LID 2 was swapped with any of the LIDs in *hypervisor 2* (6, 7 or 8), then the upper left switch would not need to be updated at all, because the initial routing already routes LID 2 and LIDs 6, 7 and 8 from the same port (Port 2). In particular for this example $n' = 1$, because only the lower left leaf switch would need to be updated.

Eventually, the cost $vSwitch\_RC_t$ of our reconfiguration mechanism is the one specified in equation 4, and in large subnets, $vSwitch\_RC_t \ll RC_t$.

$$vSwitch\_RC_t = n' \cdot m' \cdot (k + r) \tag{4}$$

One additional improvement possible to implement for our reconfiguration technique is the utilization of destination based routing for the SMP packets. When VMs are migrated, we know that the routes for the LIDs belonging to switches will not be affected. Therefore, destination based routing can guarantee proper delivery of SMPs to the switches and $r$ can be eliminated from equation 4, giving us the final equation 5.

$$vSwitch\_RC_t = n' \cdot m' \cdot k \tag{5}$$

Equation 5 assumes no pipelining of the LFT updates. In practice, pipelining is used by OpenSM when updating the LFTs blocks of the switches. Consequently the time needed for our dynamic reconfiguration mechanism is even less and depends on the pipelining capability of the SM node.

### C. Deadlock Freedom

As discussed in section III, much work has already been done on deadlock-free dynamic reconfiguration. Deadlocks can occur even if two routing functions, the $R_{old}$ and $R_{new}$, are deadlock free individually, but coexist during the transition phase from the old to the new routing function. A common way of analyzing a deadlock-free algorithm is with the Directed Acyclic Graphs (DAG) and the Up*/Down* algorithm [26]. When a packet flows through the network, it can go Up through the DAG, but once it goes Down it cannot go Up again in order to guarantee the deadlock freedom. However, this method assumes that a node ID remains at the same position at all times, and with proper handling, a new routing can be calculated so it can coexist with the old one without introducing deadlocks. In the case of live migration, however, a node ID is allowed to move. A moved node is a node that has been removed from one location in the network, and appeared at a different location without changing the ID of the node. This case cannot be handled as if a node goes down and another node comes up at a different location in the network, because in this case the two nodes would have different IDs.

One way to reduce the chances for a deadlock to occur, is to drain the communication queues with flows towards the migrating-VM by signaling the peer nodes [18]. Although the likelihood for a deadlock will be very low, still if a new communication channel needs to be established, the new peer does not know that it has to wait for the live-migration to complete before sending the connection establishment request, since it did not receive the signal, and a deadlock could occur. Moreover, the application and the live migration will be further slowed down because the live migration cannot start until the peers and the VM-to-be-migrated have drained their queues. Another way would be to iterate through all of the switches that needs to be reconfigured once the migration has started, and before the actual reconfiguration happens set the corresponding LID of the VM participating in the live-migration to be forwarded through port 255 of the switches. This method would force packets flowing towards this LID to be dropped at those switches. Afterwards, the reconfiguration could be applied. This is essentially a partially-static reconfiguration that drops the traffic only towards the migrated VM instead of "freezing" the whole network while reconfiguring. This method would prolong the reconfiguration time as it would add to equation 5 another $n'$ SMPs (1 SMP per switch that needs to be updated, to invalidate the LID of the migrated VM before the actual reconfiguration), and would probably force more packets than needed to be dropped.

In the current implementation of our dynamic reconfiguration method, deadlocks could possibly occur when swapping LIDs and they will be resolved by IB timeouts, the mechanism which is available in IBA. Improved handling of deadlock-free dynamic reconfiguration in the context of live migration is left for future work.

### D. Reconfigure Limited Number of Switches

In the proposed dynamic reconfiguration method, the reconfiguration procedure will iterate through all of the switches as shown by algorithm 1 and depending on the existing LFTs, not all of the switches may need to be updated as described in section VI-B.

This is a deterministic method that guarantees that the initial load balancing will be kept, but it may not be optimal, as there are situations where we can safely reconfigure much less switches without affecting the balancing of the initial routing. A special case is the case of live migration of a VM within a leaf switch. In this special case regardless of the network topology, only the leaf switch needs to be updated. In Fig. 6, if *VM3* moves from *Hypervisor 1* to *Hypervisor 2*, only *switch 1* needs to be updated. *Hypervisor 1* and *Hypervisor 2* are both connected on *switch 1*, so any local changes will not affect the balancing of the rest of the network, nor the rest of the nodes hosted by the leaf switch, since a leaf switch is non-blocking. However, the deterministic method may update more switches. Consider the example that the initial routing algorithm had calculated that traffic from *Hypervisor 4* towards *Hypervisor 1*, follows path $P1$ through switches $12 \rightarrow 9 \rightarrow 5 \rightarrow 3 \rightarrow 1$, and traffic towards *Hypervisor 2* follows path $P2$ through switches $12 \rightarrow 10 \rightarrow 6 \rightarrow 4 \rightarrow 1$. If the dynamic LID assignment model is implemented as one can see in Fig. 6, traffic towards VM3 would follow $P1$ towards *Hypervisor 1* before the migration, and follows $P2$ towards *Hypervisor 2* after the migration. In the worst case, all switches may be updated while only one switch needs to be updated; the leaf switch.

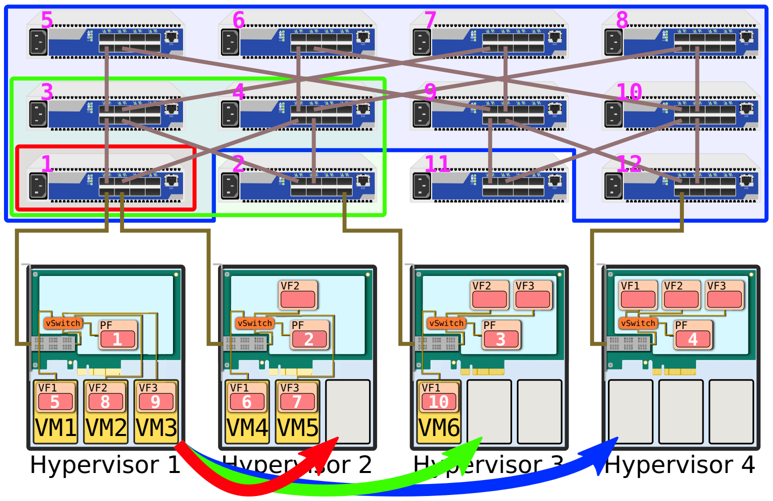

Fig. 6. vSwitch - LFTs Update on Limited Switches. Matching color of arrow and box highlighting switches, illustrates how many switches need to be updated on a minimum reconfiguration when a VM is live migrated in different parts of the network.

Furthermore, if we limit the number of switches that needs to be updated, we can efficiently support multiple parallel migrations and reconfigurations at different parts of the network simultaneously. In the case of live migrations within leaf switches we could have as many concurrent migrations as there exists leaf switches, without introducing interference.

By using the combined skyline [24] of the parts of the network that will be affected by the reconfiguration, a topology-agnostic minimum part of the network that needs to be reconfigured can be found. Nonetheless, the minimum reconfiguration has to be handled with care, in order to avoid degrading the load balancing of the network if the reconfiguration spans more switches than a single leaf switch. As illustrated in Fig. 6, when a VM is moved far from the source hypervisor, from an interconnection point of view, more switches will have to be updated. As a consequence, less concurrent live migrations are allowed in order to ensure that the migrations and reconfiguration will both complete without interfering with other migrations and reconfigurations.

## VII. vSwitch Emulation and Simulation Results

In this section we present the testbed we used in our experiments, and the experiments we carried out for evaluating the efficiency of our proposed vSwitch architecture.

### A. *Testbed*

Our testbed consists of 3 SUN Fire X2270 servers with 4 cores and 6 GB RAM each; 4 HP ProLiant DL360p Gen8 servers with 8 cores (two CPUs) and 32 GB RAM each; 2 HP ProLiant DL360p Gen8 servers with 4 cores and 32 GB RAM each; and 2 InfiniBand SUN DCS 36 QDR switches. The OpenStack Grizzly cloud environment is deployed on Ubuntu 12.04, and a CentOS 7.0 image is used for the virtual machines. The three SUN Fire servers are used as the OpenStack Controller, Network and Storage nodes. The HP machines serve as OpenStack compute nodes. The OpenStack management network is based on Ethernet, while IB is used for the VMs. All compute nodes are equipped with the Mellanox ConnectX®-3 VPI adapters and SR-IOV enabled Mellanox OFED V2.3 drivers. The same version of Mellanox OFED is also installed on the CentOS virtual machines.

### B. *vSwitch Emulation*

Currently, the IB Shared Port architecture is the only architecture supported by hardware, so we emulated the vSwitch environment with prepopulated LIDs that we described in section V-A. We also implemented our dynamic reconfiguration method based on LID swapping in OpenSM. Due to the *Shared Port* implementation, all the VMs share the same LID, so we had to limit the number of VMs allowed to run on a compute node to one for the emulation. If we run more than one VM per node, the connectivity will be broken for all other VMs sharing LID with the migrating VM. Also, a VM that is migrated, can only be moved to an OpenStack compute node where no other VMs are running. We modified OpenStack to allow IB SR-IOV VFs to be used by VMs and when a live migration is triggered the following four steps are executed:

1) The SR-IOV VF is detached from the VM and the live migrations starts.
2) OpenStack signals OpenSM with information about the VM that is migrated and the destination compute node that will host the VM.
3) OpenSM reconfigures the IB network by swapping the LID of the source and destination compute nodes and transferring the GUID to the destination compute node.
4) When the migration is completed, OpenStack attaches the SR-IOV VF that holds the GUID the VM had at the source compute node.

The live-migration traffic and OpenSM signaling is flowing over Ethernet, while the VMs communicate by IB. The purpose

| Nodes | Switches | LIDs | Min LFT Blocks/Switch | Min SMPs Full RC | Min SMPs LID Swap/Copy | Max SMPs LID Swap/Copy |
|---|---|---|---|---|---|---|
| 324 | 36 | 360 | 6 | 216 | 1 | 72 |
| 648 | 54 | 702 | 11 | 594 | 1 | 108 |
| 5832 | 972 | 6804 | 107 | 104004 | 1 | 1944 |
| 11664 | 1620 | 13284 | 208 | 336960 | 1 | 3240 |

TABLE I
NUMBER OF REQUIRED SMPS TO UPDATE LFTS OF ALL SWITCHES FOR THE FAT-TREE TOPOLOGIES USED IN FIG. 7

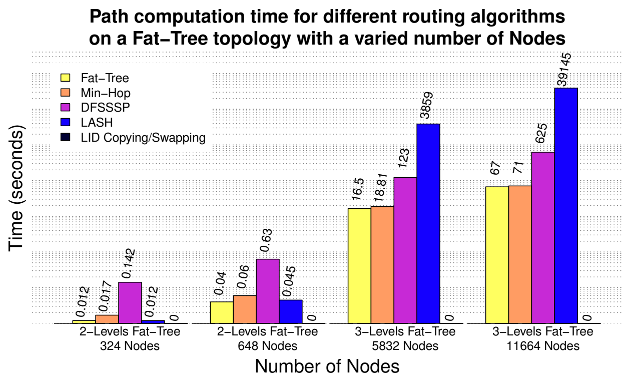

Fig. 7. IBSim Path Computation Results

of the emulated setup was to evaluate the feasibility of our dynamic reconfiguration method.

## C. Simulations

*Ibsim* was used to simulate different IB subnets in OpenSM, in order to calculate the time it takes for different routing algorithms to compute the routing tables. The path computations were executed on the 8-core machines of our testbed. The results are presented in Fig. 7. We simulated four regular Fat-Tree topologies based on 36-port switches and as one can see, when the network grows larger, the path calculation time corresponding to $PC_t$ in equation 3, skyrockets. $PC_t$ is polynomially increasing with the size of the subnet. It takes 0.012s for the fat-tree routing algorithm to compute the routing for 324 nodes, while for a 36 times larger subnet with 11664 nodes and 1620 switches, it takes 67 seconds; ~5583 times more time. DFSSSP, a topology agnostic routing algorithm needs 0.142 seconds for 324 nodes, while it needs 625 seconds for the subnet with 11664 nodes; ~4401 times more time. LASH needed 39145 seconds for the big subnet with 11664 nodes. With a traditional reconfiguration method, the path computation has to be repeated each time a live-migration is happening and LIDs are changing position in the network. Our proposed topology agnostic reconfiguration method eliminates this step. For any topology, and independent of the routing algorithm utilized for the initial path computation, zero time is spent in path recalculation. Thus, live-migration is made possible in vSwitch-enabled IB subnets regardless of the size.

In addition, a full reconfiguration will have to update the complete LFT on each switch and as the network is growing, more switches have to be updated and more SMPs per switch are needed to be sent, as explained in section VI-A. For the same four networks that we simulated in Fig. 7, one can see how many LIDs are consumed and the minimum number of SMPs needed for a full reconfiguration in Table I. Note that the amount of consumed LIDs in a subnet rules the minimum amount of LFT blocks needed to be used on each switch, but not the maximum. Consider the example that we use only three LIDs in a network with one switch and two nodes. If one of the nodes uses the topmost unicast LID, which is 49151, then the whole LFT table on the switch will have to be populated, meaning that 768 SMPs will need to be sent on the single switch — instead of the one that would need to be sent if the two nodes and the switch were using LIDs in the range 1-3. Our reconfiguration method depending on how far a VM is migrated, from an interconnection perspective, will need to send a minimum of only one SMP if the VM is migrated within the same leaf switch, or a maximum of $2 * NumberOfSwitches$ SMPs in the extreme case that all of the switches will need to be updated with two SMPs each, as explained in sections VI-B and VI-D. In particular, for the subnet with 324 nodes in Table I, a full reconfiguration would have to send at least 216 SMPs, while a worst case scenario with our reconfiguration method will send a max of 72 SMPs, 33.3% of the min number of SMPs required for a full reconfiguration, or 66.7% improvement. For the subnet with 11664 nodes, a full reconfiguration would have to send at least 336960 SMPs, while a worst case scenario with our reconfiguration method will send a max of 3240 SMPs, 0.96% of the min number of SMPs required for a full reconfiguration, or 99.04% improvement. The best case scenario for our reconfiguration mechanism is subnet size-agnostic, and will only send one SMP. As the subnet size increases, the savings from our reconfiguration mechanism increases correspondingly.

## VIII. CONCLUSION

In this paper, we proposed and analyzed two vSwitch implementations of the *vSwitch* SR-IOV architecture for IB, with different scalability characteristics, and the intent to overcome the shortcomings posed by the *Shared Port* architecture in virtualized data centers. We also proposed and implemented in OpenSM an accompanying scalable method for dynamically reconfiguring the IB network, when live migrations of VMs are part of the data center.

The first of the proposed implementations involves prepopulation of the LIDs in the available VFs, even when not all VFs are attached to VMs. The initial path computation time in the underlying IB network will increase with such an implementation, because different routes will have to be calculated for each physical node and each VF. Also, the maximum number of VFs plus physical nodes in the network cannot exceed the max number of unicast LID defined by the IBA, even if the VFs are not in use. However, better traffic balancing can be achieved.

The second of the proposed implementations suggest dynamic LID assignment as VMs are created. Then the path computation time is only bound by the number of the physical nodes in the network. This method offers simpler implementation and faster initial network configuration, giving greater flexibility for very large IB subnets, but it compromises on the traffic balancing. There is no limitation on the amount of the available VFs in an IB subnet, but the number of active VFs plus the physical nodes in the network cannot exceed the max number of unicast LID defined by the IBA at any time.

When a VM needs to be migrated, our topology agnostic reconfiguration mechanism will swap in the LFTs of the switches the LID of the VM, with the LID held by the VF at the destination hypervisor for the first of the proposed methods. In the latter one the same path with the one used by the hypervisor that hosts the VM will be used. Our reconfiguration mechanism minimizes the time needed for reconfiguring the network, as the need for path re-computation is eliminated. Moreover, the amount of required reconfiguration SMPs sent on switches is vastly reduced, and for certain scenarios, the difference is from several hundred thousand SMPs down to a single one.

## Acknowledgements

We would like to acknowledge Mellanox Technologies for providing the IB hardware we used in our experiments.

## References

[1] Simon Crosby and David Brown, "The virtualization reality," *Queue*, vol. 4, no. 10, pp. 34–41, 2006.

[2] Gil Neiger, Amy Santoni, Felix Leung, Dion Rodgers, and Rich Uhlig, "Intel Virtualization Technology: Hardware Support for Efficient Processor Virtualization.," *Intel Technology Journal*, vol. 10, no. 3, 2006.

[3] Patrick Kutch, "PCI-SIG SR-IOV Primer: An Introduction to SR-IOV Technology," *Application note*, pp. 321211–002, 2011.

[4] Jithin Jose, Mingzhe Li, Xiaoyi Lu, Krishna Chaitanya Kandalla, Mark Daniel Arnold, and Dhabaleswar K Panda, "SR-IOV Support for Virtualization on InfiniBand Clusters: Early Experience," in *13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2013*. IEEE, 2013, pp. 385–392.

[5] Viktor Mauch, Marcel Kunze, and Marius Hillenbrand, "High performance cloud computing," *Future Generation Computer Systems*, vol. 29, no. 6, pp. 1408–1416, 2013.

[6] Marius Hillenbrand, Viktor Mauch, Jan Stoess, Konrad Miller, and Frank Bellosa, "Virtual InfiniBand clusters for HPC clouds," in *Proceedings of the 2nd International Workshop on Cloud Computing Platforms*. ACM, 2012, p. 9.

[7] InfiniBand Trade Association, "InfiniBand Architecture Specification 1.2.1," 2007.

[8] TOP500, "Top500.org," http://www.top500.org/, 2014, [Online; accessed 12-January-2015].

[9] Wei Lin Guay, S.-A. Reinemo, B.D. Johnsen, T. Skeie, and O. Torudbakken, "A Scalable Signalling Mechanism for VM Migration with SR-IOV over Infiniband," in *IEEE 18th International Conference on Parallel and Distributed Systems (ICPADS), 2012.*, Dec 2012, pp. 384–391.

[10] Evangelos Tasoulas, Ernst Gunnar Gran, Bjørn Dag Johnsen and Tor Skeie, "A Novel Query Caching Scheme for Dynamic InfiniBand Subnets," in *15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. IEEE, 2015.

[11] "Add SRIOV support for IB interfaces," https://www.mail-archive.com/linux-rdma@vger.kernel.org/msg11956.html.

[12] Darren Abramson, Jeff Jackson, Sridhar Muthrasanallur, Gil Neiger, Greg Regnier, Rajesh Sankaran, Ioannis Schoinas, Rich Uhlig, Balaji Vembu,, and John Wiegert, "Intel® Virtualization Technology for Directed I/O," *Intel® Technology Journal*, vol. 10, no. 3, pp. 179—192, 2006.

[13] Mellanox Technologies Liran Liss, "Infiniband and RoCEE Virtualization with SR-IOV," in *2010 OFA International Workshop*, 2010, [Online; accessed 30-January-2015].

[14] Carl Waldspurger and Mendel Rosenblum, "I/O Virtualization," *Communications of the ACM*, vol. 55, no. 1, pp. 66–73, 2012.

[15] Yaozu Dong, Xiaowei Yang, Jianhui Li, Guangdeng Liao, Kun Tian, and Haibing Guan, "High performance network virtualization with SR-IOV," *Journal of Parallel and Distributed Computing*, vol. 72, no. 11, pp. 1471 – 1480, 2012, Communication Architectures for Scalable Systems.

[16] Jiuxing Liu, Wei Huang, Bülent Abali, and Dhabaleswar K Panda, "High Performance VMM-Bypass I/O in Virtual Machines.," in *USENIX Annual Technical Conference, General Track*, 2006, pp. 29–42.

[17] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield, "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.

[18] Wei Lin Guay, *Dynamic Reconfiguration in Interconnection Networks*, Ph.D. thesis, University of Oslo, 2014.

[19] Gregory F Pfister, "An introduction to the InfiniBand architecture," *High Performance Mass Storage and Parallel I/O*, vol. 42, pp. 617–632, 2001.

[20] J. Duato, "A necessary and sufficient condition for deadlock-free routing in cut-through and store-and-forward networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 8, pp. 841–854, Aug 1996.

[21] Bilal Zafar, Timothy M Pinkston, Aurelio Bermúdez, and Jose Duato, "Deadlock-Free Dynamic Reconfiguration Over InfiniBand™ Networks," *Parallel Algorithms and Applications*, vol. 19, no. 2-3, pp. 127–143, 2004.

[22] Ruoming Pang, Timothy Mark Pinkston, and José Duato, "The Double Scheme: Deadlock-free Dynamic Reconfiguration of Cut-Through Networks," in *Parallel Processing, 2000. Proceedings. 2000 International Conference on*. IEEE, 2000, pp. 439–448.

[23] Olav Lysne, José Miguel Montañana, Timothy Mark Pinkston, José Duato, Tor Skeie, and José Flich, "Simple Deadlock-Free Dynamic Network Reconfiguration," in *High Performance Computing-HiPC 2004*, pp. 504–515. Springer, 2005.

[24] Olav Lysne and José Duato, "Fast Dynamic Reconfiguration in Irregular Networks," in *Parallel Processing, 2000. Proceedings. 2000 International Conference on*. IEEE, 2000, pp. 449–458.

[25] Frank Olaf Sem-Jacobsen and Olav Lysne, "Topology Agnostic Dynamic Quick Reconfiguration for Large-Scale Interconnection Networks," in *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*. IEEE Computer Society, 2012, pp. 228–235.

[26] Antonio Robles-Gómez, Aurelio Bermúdez, Rafael Casado, and Åshild Grønstad Solheim, "Deadlock-Free Dynamic Network Reconfiguration Based on Close Up*/Down* Graphs," in *Euro-Par 2008–Parallel Processing*, pp. 940–949. Springer, 2008.

[27] Aurelio Bermúdez, Rafael Casado, Francisco J Quiles, and Jose Duato, "Use of Provisional Routes to Speed-up Change Assimilation in InfiniBand Networks," in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*. IEEE, 2004, p. 186.

[28] Jens Domke, Torsten Hoefler, and Wolfgang E Nagel, "Deadlock-free oblivious routing for arbitrary topologies," in *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International*. IEEE, 2011, pp. 616–627.

[29] Mellanox Technologies, "Mellanox OFED Linux User's Manual," [Online; accessed 30-January-2015].

[30] Mellanox Technologies, "Mellanox Firmware Tools (MFT) User Manual," [Online; accessed 30-January-2015].

[31] Aurelio Bermúdez, Rafael Casado, Francisco J Quiles, Timothy Mark Pinkston, and José Duato, "Evaluation of a Subnet Management Mechanism for InfiniBand Networks," in *Parallel Processing, 2003. Proceedings. 2003 International Conference on*. IEEE, 2003, pp. 117–124.