Ensuring Deadlock-Freedom in Low-Diameter InfiniBand Networks

Timo Schneider
Computer Science Department
ETH Zurich, Switzerland
timos@inf.ethz.ch

Otto Bibartiu
Computer Science Department
ETH Zurich, Switzerland
ottob@student.ethz.ch

Torsten Hoefler
Computer Science Department
ETH Zurich, Switzerland
htor@inf.ethz.ch

***Abstract*—Lossless networks, such as InfiniBand use flow-control to avoid packet-loss due to congestion. This introduces dependencies between input and output channels, in case of cyclic dependencies the network can deadlock. Deadlocks can be resolved by splitting a physical channel into multiple virtual channels with independent buffers and credit systems. Currently available routing engines for InfiniBand assign entire paths from source to destination nodes to different virtual channels. However, InfiniBand allows changing the virtual channel at every switch. We developed fast routing engines which make use of that fact and map individual hops to virtual channels. Our algorithm imposes a total order on virtual channels and increments the virtual channel at every hop, thus the diameter of the network is an upper bound for the required number of virtual channels. We integrated this algorithm into the InfiniBand software stack. Our algorithms provide deadlock free routing on state-of-the-art low-diameter topologies, using fewer virtual channels than currently available practical approaches, while being faster by a factor of four on large networks. Since low-diameter topologies are common among the largest supercomputers in the world, to provide deadlock-free routing for such systems is very important.**

I. Introduction

InfiniBand (IB) [1] is an interconnection network widely used in high-performance computing (HPC) and datacenter networks[1]. It uses point-to-point links between switches and host channel adapters (HCAs). HCAs, which are identified by a local ID (LID) connect compute nodes to the network. Traffic between HCAs is routed using destination-based routing. Each switch has access to a forwarding table, which maps each possible combination of input port and destination LID to an output port.

IB uses a credit-based flow control scheme to avoid packet loss at congested switches. An output port can only send packets if it has credits at the input port of the next switch. Cyclic dependencies between buffers can deadlock the network. An example of such a deadlock is shown in Figure 1. Every HCA (H1, H2, H3, H4 in Figure 1) wants to send data to an HCA two hops away (clockwise). In the pictured scenario, none of the packets are moving, because, e.g., packets injected via channel 1 cannot move to the output port at the switch S1 because its buffer is already filled with other packets, which in turn cannot leave S1 via channel 10 because the corresponding input port at S2 is filled up (and so on). There is a dependency e.g., between channel 1 and channel 10 because when sending from H1 to H3 packets that use channel 1 need to use channel 10 next.

[1]In Nov. 2015, 47.4% of the 500 most powerful supercomputers used IB, see http://www.top500.org/

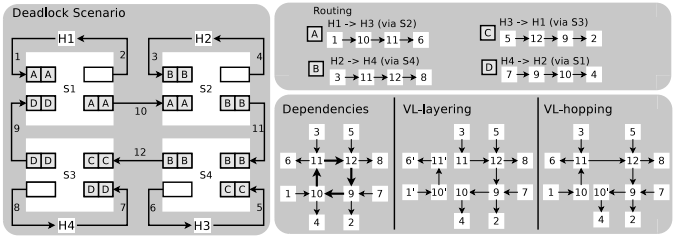

Fig. 1. Deadlock in an InfiniBand network. The dependencies between channels form a cycle, which can be broken either with VL-layering or VL-hopping. VL-hopping requires fewer resources.

Credit-induced deadlocks in IB can be dealt with in multiple ways: The problem can be ignored. Switches can employ time-outs and drop packets in case a deadlock is detected, higher level protocols need to ensure reliability. Another option is to restrict routing, such that no cyclic channel dependencies can be formed [2]. Both options are not discussed here, since they have a negative performance impact: Ignoring the problem means we forfeit the advantages of a lossless network, restricted routing means we cannot fully use the (expensive) physical network. Thus we focus on different techniques:

Layered Routing: Current IB routing engines break cyclic channel dependencies by assigning some paths to different virtual channels. Virtual channels, or Virtual Lanes (VLs) in IB terms, have separate buffer resources and credit systems. If two of the four paths in Figure 1 are assigned to VL 1 and the remaining two to VL 2, no deadlock can occur. The number of available VLs is small (each VL requires costly buffers). Determining an optimal (minimal number of VLs) deadlock-free layering is NP-complete [3].

VL-hopping: IB allows a packet to change its VL inside a switch. The algorithms presented here assign different parts of a single path to different VLs. In the example shown in Figure 1, it is enough to put a single channel into a different VL (c' indicates c was moved to another VL), while layered routing moves at least four channels, thus increasing chances for a deadlock in the new layer.

TABLE I
NOTATION

SL	Set of available SLs, in IB $s \in SL$ is a number between zero and $maxSL$ (fifteen in IB)
VL	Set of available VLs, in IB $v \in VL$ is a number between zero and $maxVL$ (up to fifteen in IB).
$I = (N, C)$	InfiniBand network graph, vertices N are HCAs and switches, edges C are physical channels.
$D = (C', E)$	Channel dependency graph, vertices C' are virtual channels. We denote elements of C' using superscripts to denote the VL, e.g., a^p is the virtual channel formed by using a with VL p.
$V(i, o, s) = v$	SL-to-VL mapping relation, takes input and output channel $i, o \in C$ and an SL value $s \in SL$ as its argument and returns the output VL $v \in VL$. Every switch contains a table that implements V for the input ports of that switch.
$R(i, d) = o$	Routing relation, maps an input channel $i \in C$ and a destination HCA $d \in N$ to the next output channel $o \in C$. Each switch implements R for the local switch ports using a forwarding table.
$P(s, d) = (c_0 \dots c_n)$	The list of physical channels $c_i \in C$ traversed by a packet injected by $s \in N$ with destination d. It can be obtained by successively applying R.
$R'(i^p, d) = o^q$	To route a packet which has the destination $d \in N$ and SL $s \in SL$ encoded in its header and arrived through the virtual input channel $i^p \in C'$, we first apply $R(i, d) = o$ then use $V(i, o, s) = q$ to obtain q, thus the packet is sent via the virtual channel o^q. To simplify notation we introduce the virtual routing relation R' to express this. The value of p is irrelevant for routing: InfiniBand switches cannot access the input VL.
$S(i, d) = s$	If an HCA uses channel $i \in C$ to inject a a packet with destination $d \in N$ it must use SL $s \in SL$, this models PathRecords.

II. BACKGROUND

An InfiniBand network can be modeled as a directed graph $I = (N, C)$ where N is the set of HCAs and switches, while C is the set of links (often referred to as physical channels) connecting them. Each directed physical channel $i = (t, h) \in N \times N$ is connected to two endpoints, which we refer to as $t = tail(i)$ and $h = head(i)$. IB links are full-duplex, i.e., $(t, h) \in C \leftrightarrow (h, t) \in C$. InfiniBand uses destination based routing, every packet contains its destination HCA in the header. Every switch contains a table that determines the output port for every possible destination. We model routing as a relation $R(i, d) = o$ which assigns an output channel $o \in C$ to a combination of input channel $i \in C$ and destination node $d \in N$. Routing happens only at switches; HCAs are connected to exactly one input and one output channel and do not forward packets, they only inject or consume them.

Routing tables in IB switches are programmed by the subnet manager. The most widely used subnet manager is OpenSM. When OpenSM is started, it detects the network topology and programs the routing tables in switches according to an algorithm selected by the user. After the network is fully configured, the subnet manager keeps running. If a link fails, it can update routing tables. Often the subnet manager is run on a switch itself, instead of using a dedicated node.

IB supports up to 16 virtual lanes per physical channel[2]. VL 15 is used only by management traffic. Each VL has separate buffers and its own credit system, thus, VLs implement virtual channels. The output VL is determined based on the combination of the physical channel a packet used to enter a switch (input channel), the physical channel assigned to the packet by the routing relation R (the output channel), and the Service Level (SL). The SL of a packet is an integer between 0 and 15, it cannot be changed while the packet traverses the network, and is encoded in the packet header by the sender of the packet. Every switch contains an SL-to-VL mapping table which implements the relation $V : (i, o, s) = v$ where $s, v \in [0, 15]$ and $i, o \in C$. To forward a packet with destination d and service level s, the switch first determines $R(i, d) = o$, then applies $V(i, o, s) = v$ to forward the packet on the virtual channel o^v. There are two types of VL buffer semantics: either there is one big buffer per VL where input and output buffer share the same memory region [3]–[5], or two disjoint buffers, one for sending and one for receiving [6]. The DF-SSSP algorithm in OpenSM constructs the CDG according to the two-buffer model. In this work, we use the two-buffer model, since the one-buffer model predicts deadlocks even for two switches connected with a single cable, which does not match our experience with IB hardware.

The Service Level s is an important part of the routing, as it determines which VL a packet will use in any switch, together with the SL-to-VL mapping table. The SL-to-VL mapping table is programmed into the switch by the subnet manager. However, the SL has to be set in the packet before it arrives at the first switch, thus it needs to be done by the host, even though it is part of routing, which should be transparent for the host. This dilemma is solved using PathRecord queries: Before a host sends a packet to a destination, it should query the subnet manager, which SL value should be used for that source/destination pair. We model this with the relation $S(i, d) = s$ which assigns an SL value $s \in SL$ to the combination of input channel $i \in C$ to which the sending HCA is connected, and the destination HCA $d \in N$. Since the subnet manager is a single endpoint, often running on embedded hardware, this can lead to considerable startup overhead [7]. Thus many IB applications do not perform such queries, i.e., Open MPI [8] has to be compiled and run with special flags in order to perform PathRecord queries. While previously known VL-based solutions for deadlock-free routing with InfiniBand, such as DF-SSSP and LASH require PathRecord queries (unless a single VL is sufficient to ensure deadlock freedom), we propose an algorithm for low-diameter networks which uses multiple VLs to ensure deadlock-freedom but relies solely on SL-to-VL mapping. All packets can be injected into the network with the same SL.

Now that we have the notation defined to model an Infini-

[2]Currently available switches support only up to eight VLs, but all datastructures are wide enough for 16.

Band network, we can formalize the dependencies between channels: The nodes C' of the channel dependency graph $D = (C', E)$ are the virtual channels. Edges in the CDG are formed by dependent virtual channels. Two virtual channels a^m, b^q are dependent if there exists a source/destination pair $s, d \in N$ and packets that travel from s to d first use channel a^m then use b^q immediately after that. To express which virtual channels a packet uses we define the virtual routing relation R' (in contrast to the routing relation R which is defined on physical channels):

$$\begin{aligned} R'(a^m, d) = b^q \leftrightarrow \exists\ s, d \in N : R(a, d) = b \wedge \\ P(s, d) = c_0 \dots c_{a-1}, c_a \dots \wedge \\ \exists k \in SL : V(c_{a-1}, c_a, k) = m \wedge \\ V(a, b, k) = q \ \wedge\ S(c_0, d) = s \end{aligned}$$

Using R' expressing E becomes straightforward:

$$(a^m, b^q) \in E \leftrightarrow \exists d \in N, a^m \in C' : R'(a^m, d) = b^q$$

If D is acyclic, there can be no credit-induced deadlocks, which has been proven by Dally et al. [5]. The intuition of the proof is that in an acyclic CDG we can define a total order on the channels by sorting them topologically (only acyclic graphs have a topological order), thus if $(i, j) \in C'$ then $i > j$. Consider the least channel in this order with a full queue, l. Every channel, that l sends packets to is less than l, and thus does not have a full queue. Therefore, l is not blocked, and there is no deadlock.

III. Incrementing VLs to Avoid Deadlocks

Current routing algorithms for InfiniBand, i.e., DF-SSSP and LASH achieve deadlock-free routing with VL layering: VLs are set for entire paths. No packet will be forwarded from one VL to another in any switch, thus there exists no $(a^m, b^q) \in E$ with $m \neq q$ and we can create independent layers of channel dependencies for each VL. If none of the layers contains a cycle, D does not contain a cycle, and the routing is deadlock-free. The problem with this approach is that paths which contribute to a circle in the CDG need to be moved into another layer in its entirety. Thus, chances are that we get another cycle in that layer and have to move some paths to the next layer. Deciding which paths should be moved to which layer in order to minimize the number of used layers is an NP-complete problem [3].

However, in this work we use a different approach, to our knowledge first mentioned by Gopal [9]: If we impose an ordering on the set of available VLs, i.e., $VL = [0, 15]$ and use $\leq$ as ordering constraint, we can increase the VL at every hop, such that the following condition holds:

$$\forall d \in N, \forall i^p \in C' : R'(i^p, d) = o^q \rightarrow q = p + 1 \quad \text{(VLInc)}$$

For every destination node d and every input channel i^p the output VL q is bigger (by one) than the input VL.

Theorem 1. *The CDG is acyclic if the VL is incremented by one in every hop, i.e., Equation VLInc holds.*

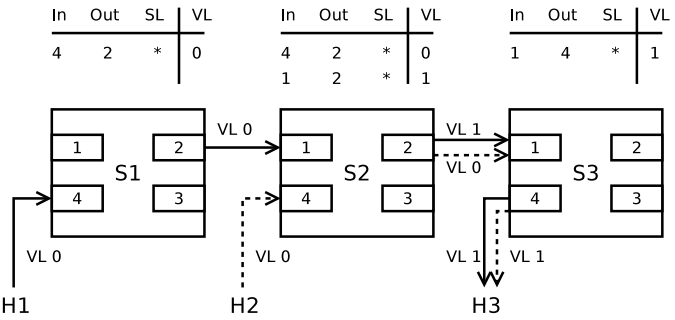

Fig. 2. Example SL-to-VL based deadlock free minimal path routing in diameter two networks.

Proof. Equation VLInc imposes a partial order on channel nodes in D, for any $(i^p, o^q) \in E$, $q = p + 1$. Thus D can be topologically sorted because there are no channels from high VLs to low VLs. This implies that D is a DAG. □

Note that we can generalize Equation VLInc: We do not require $q = p + 1$. From the proof of Theorem 1 follows that $q > p$ (or any other total order among VLs) is sufficient.

Unfortunately there is no straightforward way to implement routing which satisfies Equation VLInc (or a generalization thereof) in InfiniBand: The output VL is decided only by taking into account the physical input port, i.e., i, while we would require i^p, the combination of input port and input VL. However, the input VL is not available in the function V which decides the output VL. Instead we have only the SL (which remains constant while data is forwarded). Changing that has been proposed [10] but to the best of our knowledge that feature is not available in IB.

However, we show that it is possible to construct V such that the resulting CDG satisfies Equation VLInc. The diameter of the network influences how this is done, we describe the special case where all paths have at most length two (not counting HCA-to-switch links) first. We name the resulting algorithm DF-D2. We generalize it later to networks with an arbitrary diameter, and name the general algorithm DF-DN. Our algorithm does not provide routing itself, it takes an existing routing R and ensures deadlock-freedom.

IV. Incrementing VLs in Diameter-Two InfiniBand Topologies

In diameter-two networks we can make the CDG D acyclic by leveraging the SL-to-VL mapping V, given that the routing uses only minimal paths. A network has a diameter of two, iff all shortest paths between any two HCAs contains at most three switches (this implies the diameter of the network of switches is two).

The main idea is to use VL 0 between the injecting HCA and the second switch on each path, and VL 1 afterwards. Figure 2 gives an example. Since each switch port can be connected only to one vertex $\in N$, we can decide which output VL should be used based on the type of the connected device.

The algorithm shown in Figure 3 generates the SL-to-VL tables V for deadlock-free diameter-two (DF-D2) routing. For

Data: $I = (N, C)$: Diameter-two InfiniBand network
Result: S, V : SL-to-VL mapping, guarantees DL-freedom
forall $(i, o) \in C^2$ **do**
  $s \leftarrow head(i)$
  if $s = tail(o) \wedge isSwitch(s)$ **then**
    $h \leftarrow head(i)$
    if $isSwitch(h)$ **then**
      $V(i, o, *) \leftarrow 1$
    else
      $V(i, o, *) \leftarrow 0$
$S(*, *) \leftarrow 0$

Fig. 3. The DF-D2 Algorithm fills the SL-to-VL mapping table in such a way that the VL is incremented after the first hop.

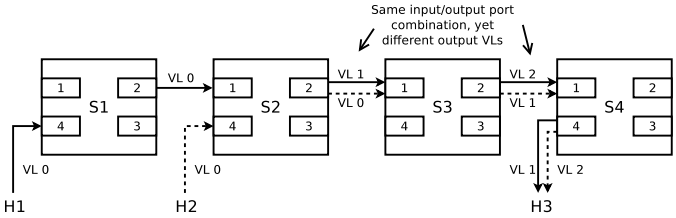

Fig. 4. Switch S3 cannot decide the output VL solely based on input and output port, as in DF-D2, the SL needs to be taken into account.

each combination of input and output port (i, o) in any switch we check if i is connected to an HCA or a switch. If i is connected to an HCA, we add $V(i, o, *) = 0$ to the SL-to-VL table, where $*$ means "for any SL" (in practice that means we add 16 entries, one for each SL). If i is connected to a switch, we use VL 1 instead. Every HCA should use VL 0 when injecting packets, regardless of the destination, which is expressed with $S(*, *) \leftarrow 0$ This algorithms runtime complexity is linear in the total number of switch ports, since OpenSM directly provides the list of input and output channels for each switch.

V. Incrementing VLs in Arbitrary Low-Diameter InfiniBand Topologies

If the network I (without HCAs) has a diameter higher than two, The algorithm shown in Figure 3 fails to provide deadlock free routing. An example is given in Figure 4. Input port 3 of switch S3 is connected to another switch, thus it will use VL 1 as output VL for any data arriving on port 3. However, this will violate Equation VLInc: Data which originates from H2 should use VL 1, but data from H1 should use VL 2. However, the switch cannot differentiate between both flows (with the SL-to-VL mapping constructed by the in Figure 3), since the input and output port are identical. Thus we need to leverage the SL — H1 and H2 need to set different SL values when sending data.

A possible solution to differentiate between different paths (determined by their source destination pair) would be to give every $(s, d) \in N^2$ a different SL. Assume a specific pair (s, d) is assigned SL z. Now we can walk the path (defined by R) that messages from s to d will traverse, we denote that path as list of used physical channels $P(s, d) = (c_0, c_1, ...c_n)$.

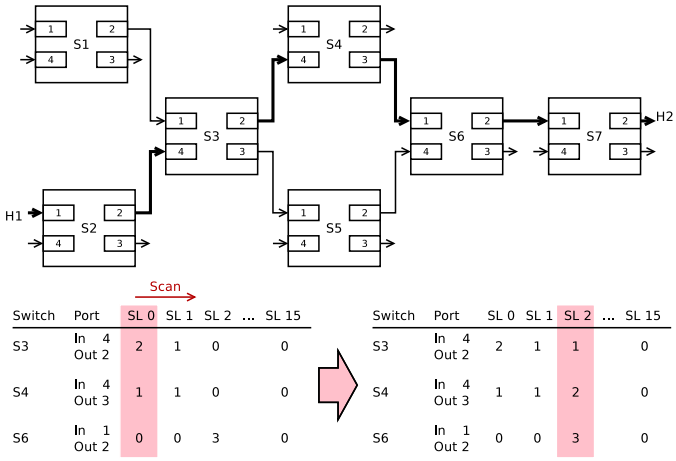

Fig. 5. DF-DN: We traverse each route, collect the relevant SL-to-VL table entries in traversal order, and stack them from top to bottom. Then we scan from left to right until we find an SL which increments the VL at every hop or has unused entries (set to zero) so that we can set them such that the VL is incremented at every hop.

We know that for every switch the position along the path determines which input and output VL must be used, if we start with VL 0 and increment the VL by one at every hop: A switch which is characterized by the input/output port combination (c_l, c_{l+1}) needs to assign the output VL $l + 1$ to any packet which arrives with an SL of z. Thus we can generate an entry $V(c_l, c_{l+1}, z) = l + 1$ to the SL-to-VL table. While a SL-to-VL table built in that way will satisfy Equation VLInc, it is not practical for InfiniBand networks, since InfiniBand only offers 16 different SLs, however, the number of SLs required by the approach described above is quadratic in the number of HCAs, it allows for a maximum of four HCAs. The number of required VLs is equal to the diameter of the network graph[3].

To make incrementing VLs practical for InfiniBand networks, we have to make better use of available SLs than the algorithm outlined above. Instead of giving each source/destination pair its own SL, we collect which input/output port combinations uses which output VL. This is done along all possible paths. Figure 5 gives an example: In the network shown, assume we scan the highlighted path from source H1 to destination H2. We remember all the switches on the path (except the first and last one), so our path for this example is S3, S4, S6. We collect all the relevant parts (where the input/output port matches that of the currently scanned path) of the SL-to-VL tables in switches along that path in traversal order, as indicated in Figure 5. The highlighted path is not necessarily the first one which is examined, thus there are already some non-zero entries in our example (initially we populate all SL-to-VL tables with zero). Now we scan all columns from left to right, until we find a column where either the VL increases by one in every line, or it can be made into that by changing zero entries to non-zeros. If such a column

[3]minus two, if we count the first and last hop (from/to HCAs) as well

Data: $I = (N, C)$: InfiniBand network, R : Routing function
Result: H, V : SL-to-VL mapping, guarantees DL-freedom
$cache[0 \dots maxVL][0 \dots maxSL] \leftarrow 0$
forall $(u, v) \in N^2 : (u \neq v) \wedge isHCA(u) \wedge isHCA(v)$ **do**
  $c_{out} \leftarrow (u, x) \in C$
  $s \leftarrow head(c_{out})$
  for $i \leftarrow 0 \dots maxVL$ **do**
    $c_{in} \leftarrow c_{out}$
    $c_{out} \leftarrow R(c_{in}, v)$
    forall $k \in SL$ **do**
      $cache[i][k] \leftarrow V(c_{in}, c_{out}, k)$
    $s \leftarrow head(c_{out})$
    if $s = v$ **then**
      $found \leftarrow false$
      for $sl \leftarrow 0 \dots maxSL$ **do**
        if $found = true$ **then**
          break
        $found \leftarrow true$
        for $j \leftarrow 0 \dots i$ **do**
          $vl \leftarrow cache[j][sl]$
          $found \leftarrow found \wedge (vl \in \{0, j+1\})$
      if $found = true$ **then**
        for $j \leftarrow 0 \dots i$ **do**
          $cache[j][sl] \leftarrow j + 1$
          $V(c_{in}, c_{out}, sl) \leftarrow j + 1$
      else
        Error Not enough SLs!
      $S(c_{out}, v) \leftarrow sl$
      break
  if $i = maxVL$ **then**
    Error Not enough VLs!

Fig. 6. The DF-DN Algorithm fills the SL-to-VL mapping table in such a way that the VL is incremented after the first hop.

is found, the position of the column gives us the SL we will use.

The DF-DN algorithm, shown in Figure 6, provides details: The outer loop iterates over all combinations of sources and destinations. We traverse $P(u, v)$ in the loop over i, if the path is longer than $maxVL$ hops, our algorithm will fail due to an exhaustion of VLs. The $cache$ structure holds the relevant subset of V, which we modify once we find an appropriate SL, for which all entries are monotonically increasing, or we can make them monotonically increasing by setting unused (zero) entries. The variable $found$ keeps track of this.

VI. Evaluation

To evaluate performance and test correctness of our routing engine, we implemented the algorithms described in Sections IV and V in OpenSM. Our algorithms are implemented as an extension of non-deadlock-free routing engines: First we perform SSSP or MinHop routing (and only use one VL). After that we assign VLs to hops, and program SL-to-VL tables.

In order to evaluate the implementation with multiple different networks, we use the ibsim (version 0.6) InfiniBand simulator. The ibsim utility simulates an IB network, thus we can use tools such as OpenSM (version 3.3.19) on that simulated network, furthermore we can obtain the network configuration using ibnetdiscover (version 1.6.1) and ibdiagnet (version 1.5.7). We wrote a tool to convert the output of these tools into input files for the OMNet++ (version 4.6) based Flit-level InfiniBand simulator ib-flit-sim (version 28.07.13). The original version of ib-flit-sim does not support virtual lanes. We added support for virtual lanes in the context of this work. The OMNet++ simulator allows us to test correctness of our routing engine.[4]

All experiments are performed using Centos 6.7, kernel version 2.6.32. The host system is an Intel i5-4590 CPU @ 3.30GHz and 8 GB RAM.

A. Slim Fly Topologies

We generated InfiniBand networks for different Slim Fly [11] topologies[5] shown in Figure 7. To verify that deadlocks are indeed a problem on this topology, we simulated a network with the given topologies in ib-flit-sim with the following parameters: 800 credits per port, an MTU of 2KiB, 10 GiB full duplex links, and a message size of 128 KiB. The routing tables are created with the SSSP routing engine from OpenSM. The traffic pattern is generated such that every host sends a message to a different receiver. If we find cycles in the CDG, we select sender/receiver such that they utilize paths which contribute to a cycle in the CDG if possible (otherwise the selection is random). Routing is generated with OpenSM, using the SSSP routing engine. All communication uses VL zero. To lower memory usage and runtime of the ib-flit-sim simulation we simulated only two (instead of p) HCAs per switch for the deadlock simulations.

To resolve deadlocks we applied existing deadlock-free routing engines LASH and DF-SSSP as well as our proposed DF-D2 heuristic together with MinHop and SSSP. The UPDN routing engine fails to work with most evaluated topologies because it is unable to identify a set of root switches. We can see in Figure 9 that LASH is able to find deadlock free configurations with two VLs for all networks. Note that in practice LASH and DF-SSSP utilize all available VLs, by balancing routes across them. The numbers reported here are the results before the balancing step is performed. For the topologies with more than 50 switches (350 hosts), the DF-SSSP heuristic is suboptimal, it requires three instead of two VLs, while our heuristic gives the optimal result.

By itself that would not be a surprising result. After all, with LASH we already have a routing engine capable of delivering deadlock free routing using the minimal number of VLs. However, our heuristic has two advantages over LASH: it does not require path queries and it is much faster. In Figure 8 we show that our heuristic is capable of routing the largest evaluated network two orders of magnitude faster than LASH, while outperforming DF-SSSP by a factor of three. Figure 8 shows a boxplot [12] of eleven measurements per combination of network and routing engine. The thicker line inside the boxes marks the median value, the boxes extend to lowest and

[4]Our toolchain can be downloaded from https://spcl.inf.ethz.ch/Research/Scalable_Networking/DFDN/.

[5]All topologies were obtained from https://spcl.inf.ethz.ch/Research/Scalable_Networking/SlimFly/

p	Switches	Hosts	Time to DL
5	18	90	no DL
7	50	350	40.5 us
11	98	1078	9.2 us
17	242	4114	22.3 us
19	337	6403	8.1 us

Fig. 7. Balanced Slim Fly topologies. The parameter p determines how many HCAs are attached per switch, the network is then built such that it provides full global bandwidth.

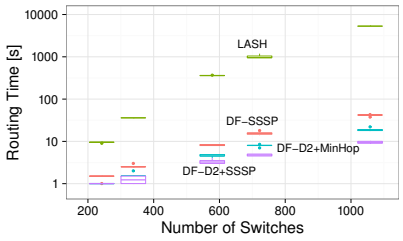

Fig. 8. Performance of different OpenSM routing engines for Slim Fly topologies. Our heuristics outperforms LASH by orders of magnitude and outperforms DF-SSSP by a factor of three.

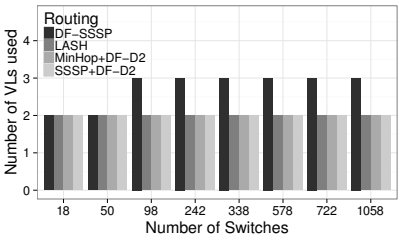

Fig. 9. Required VLs for Slim Fly topologies, for different routing engines. DF-SSSP is suboptimal for larger networks, LASH and DF-D2 deliver the optimal result.

highest value within the 1.5 interquartile range. In Figure 9 the low dispersion of the measurements and the logarithmic scale make most boxes appear as a single line. For the two smallest networks for which data is plotted the measurements for DF-D2+SSSP and DF-D2+MinHop are overlapping, thus only three boxes are visible.

B. Dragonfly Topologies

We evaluated our heuristic on different Dragonfly [13] topologies (obtained from the topology collection which accompanies the Slim Fly technical report), they vary in size between 18 and 1058 switches. The results are shown in Figure 10. All used Dragonfly topologies are balanced configurations with full global bandwidth. A Dragonfly topology consists of g groups, the a routers within a group are fully connected, if groups are contracted into a single vertex the resulting graph is again fully connected. To obtain such a network, each router is connected to p HCAs, $a-1$ routers in its own group and h routers in other groups. To obtain full global bandwidth we chose $a = 2p = 2h$ and $g = ah + 1$ The number of HCAs, denoted as n, can be calculated with $n = gap = (ah+1)2p^2 = (2p^2+1)2p^2 = 4p^4 + 2p^2$. For our experiments we varied the number of HCAs per switch from $p = 2$ to $p = 8$.

For Dragonfly topologies LASH and DF-DN (combined with MinHop and SSSP) both deliver the optimal result of 3 VLs used to provide deadlock-freedom, however DF-DN was able to do so 68 times faster than LASH, and more than three times faster than DF-SSSP for the largest network with $p = 8$, which implies 16512 HCAs.

Figure 13 shows a boxplot of the runtimes of LASH, DF-SSSP, and DF-DN for Dragonfly topologies of different sizes. The runtimes for DF-DN+MinHop and DF-DN+SSSP are very similar, thus we plot both in one set of boxes, labeled DF-DN. LASH is roughly two orders of magnitude slower than DF-DN, while DF-DN requires 28 seconds to route the largest network, LASH requires 3.5 hours. Note that the routing engine not only has to be run when the network is initially deployed, but also in the case of link failures. Thus we consider LASH impractical for large networks. DF-SSSP is four times slower than DF-DN, it requires 121 seconds to perform deadlock-free routing.

C. Orthogonal Fat-Tree Topologies

The Orthogonal Fat-Tree [14] (OFT) topology is based on the idea of stacking two Single-Path Trees (SPT) on top of each other. An SPT consists of two layers of routers. The R_1 routers in the first layer have a router-to-router radix of r_1 and the R_2 routers in the second layer r_2. HCAs are be connected only to the first layer of routers and router-to-router links have an endpoint in both layers, thus r_1 HCAs are attached to each router in the first layer. Connections are formed such that there is only a single path between any two HCAs, and the number of used switches in the second layer is minimized. How to construct such networks for arbitrary values of r_1 and r_2 is not straightforward. When constructing networks, it is a common requirement that switches have a certain number of ports. This can be reached by combining multiple SPTs into a Stacked SPT (SSPT). Building an SSPT with $r_1 = r_2 = k$ requires stacking two 2 SPTs, the resulting structure is called Two-Level k-OFT [15]. We generated OFTs for $k = 3, 4, 6, 8, 14, 16, 18$. As shown in Figure 11, DF-DN outperforms DF-SSSP on all networks of medium and large size by a large margin. LASH saves one and sometimes two VLs compared to DF-DN, however, for the largest network they are equal. This is a positive result for DF-DN, given LASHs running time is orders of magnitude higher.

D. Real-World Topologies

We evaluated our heuristic on several real-world networks. The results are shown in Figure 12. None of those was constructed as a low-diameter topology, yet our heuristic is capable of providing deadlock-free routing for all of them. In the case of Juropa our heuristic requires only six VLs, while DF-SSSP requires nine. LASH on the other hand failed to provide deadlock-free routing on that topology.

VII. Related Work

In this work, we discuss credit-induced deadlocks in InfiniBand networks. Our notation is based on that of Dally and Seitz [5], who showed the connection between deadlocks and the channel dependency graph. The key idea for the heuristics presented here is to increment the VL at every hop. This scheme has been introduced by Gopal [9], and was used recently by Besta and Hoefler [11] in the context of diameter-two networks. To the best of our knowledge, VL hopping has

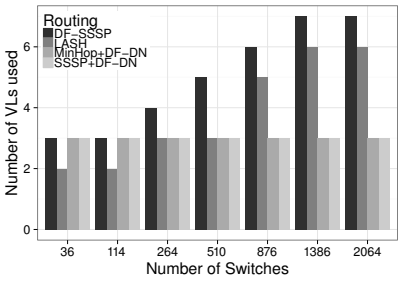

Fig. 10. Required VLs for Dragonfly topologies and different routing engines. DF-SSSP and LASH require twice as many VLs as DF-DN for large networks.

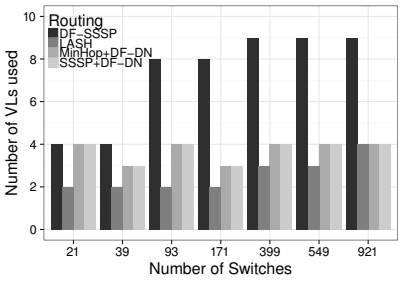

Fig. 11. Required number of VLs for different Orthogonal Fat-Trees, with radixes between 3 and 18. They provide full global bandwidth. DF-DN saves up to 5 VLs compared to DF-SSSP, and performs almost equal to LASH (but is much faster).

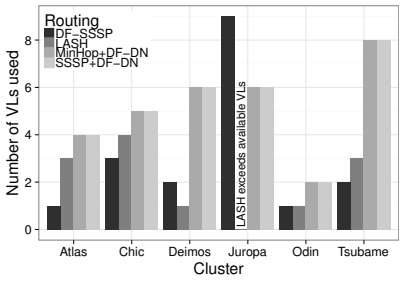

Fig. 12. Required number of VLs for different Real-World Topologies. None of them is a low-diameter topology, but the DF-DN heuristic can deliver deadlock free routing for all of them and outperforms DF-SSSP on Juropa, while LASH is unable to provide a valid routing.

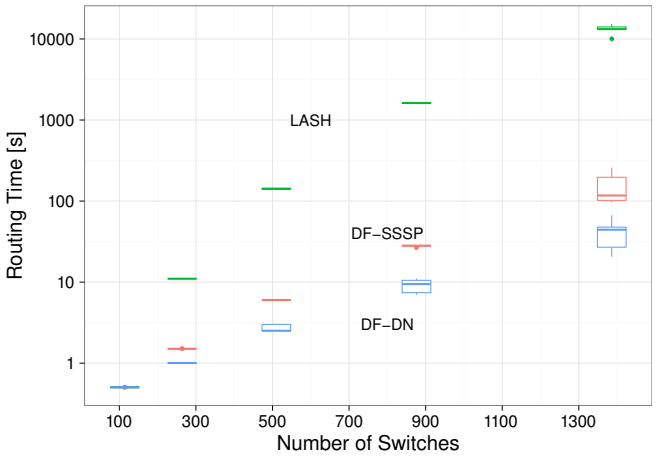

Fig. 13. Performance of different OpenSM routing engines for Dragonfly topologies. Our heuristics outperforms LASH by orders of magnitude (while delivering equal results) and outperforms DF-SSSP by a factor of four, while delivering better results.

not been used in InfiniBand networks before, our heuristic to program the SL-to-VL mapping tables is novel. Our heuristics also target low-diameter networks [13]–[15], which are an area of active research, since they offer high-performance (small number of hops) at relatively low cost.

Some existing routing algorithms for InfiniBand, namely DOR and Fat-Tree provide topology-aware, deadlock-free routing for specific networks topologies. They are deadlock-free as long as there is no change in network structure. Often performance decreases even in case of single link failures and these routing engines will fall-back to a general-purpose, non-deadlock-free algorithm. Domke et al. [16] propose fail-in-place network design and suggest network-agnostic routing in order to keep up the high throughput, even in presence of link or switch failures. The Up/Down routing scheme for InfiniBand [17] offers deadlock-free routing for non-structured networks. It uses a tree structure in order to disallow paths which create channel dependency cycles. However, Up/Down routing does not offer shortest-path routing. LASH [18] is another topology agnostic, deadlock-free routing engine for InfiniBand. It computes shortest-paths between nodes and assigns paths to VL layers, such that a minimal number of VLs are utilized. Since this is an NP-hard problem, LASHs network initialization time exceeds reasonable values, e.g., 3.5h for a Dragonfly topology with 9702 HCAs. Daryin et al. [10] proposed a VL-incrementing scheme to provide deadlock-free InfiniBand routing, however, they did not design any implementation using SL-to-VL tables. Instead they suggest to replace the SL-to-VL table with a VL-to-VL table that takes the input VL into account to determine the output VL for the next hop. This would require fundamental changes to IB, which are not necessary, because we have shown that DF-DN and DF-D2 can increment VLs using only SL-to-VL tables. Hoefler et al. [4] introduced the SSSP routing algorithm. SSSP provides balanced shortest-path routing and minimizes the number of paths per edge globally. However, SSSP routing is not deadlock-free, which is why Domke et al. introduced DF-SSSP [3]. DF-SSSP performs SSSP routing and subsequently, assigns each path to a VL layer in order to break channel dependency cycles. It uses a heuristic to minimize the number of VLs for path layering in polynomial time. We show that our heuristic, which takes SL-to-VL mapping into account is faster and requires less VLs in many practical networks. Nue-Routing [19] is similar to DF-SSSP in that it does not make use of SL-to-VL mapping for InfiniBand. Nue-Routing combines the layering technique with restricted routing: it performs layering with a given number of VLs, while using an “escape path” based on restricted routing. If the number of available VLs is smaller than what would be required to obtain deadlock-freedom when solely relying on layering, any paths still containing cycles are changed such that they utilize the escape path. We view Nue-Routing as complimentary to DF-DN: While DF-DN is well suited for low-diameter networks, Nue-routing has advantages when routing high-diameter networks.

To provide deadlock-free routing, DF-SSSP and LASH require expensive PathRecord queries: A host needs to query the Subnet Manager for the path SL before it can send any

messages. To avoid the SM to be a bottleneck, Tasoulas et al. [7] propose a query caching scheme, distributed SMs have been proposed as well [20]. In contrast, our DF-D2 algorithm provides deadlock-free routing in diameter-two networks without path queries.

VIII. Conclusion

The algorithms presented in this work make arbitrary minimal-path based routing modules for InfiniBand deadlock-free. They outperform existing approaches, such as DF-SSSP and LASH on most low-diameter topologies both in terms of used VLs and runtime. The number of used VLs is bounded by the diameter of the graph. In our experiments with low-diameter topologies, we did not find a case were the number of available SLs was the limiting factor. Layering based approaches cannot provide such bounds. We expect that future networks, i.e., for Exascale machines, will make use of low-diameter topologies, such as Dragonfly, Slim Fly or Orthogonal Fat-Trees, for which our heuristic is very well suited. Contrary to previous approaches [10], we do not require any hardware changes.

Apart from providing deadlock-free routing, the DF-DN algorithm for InfiniBand introduced in this work also opens new avenues for future work on performance of InfiniBand networks: Virtual channels in InfiniBand serve two purposes, they provide deadlock freedom as well as Quality of Service (QoS). However, in contemporary InfiniBand HPC networks the capability to use VLs for QoS is often ignored, regardless of strong evidence [21], [22] that it can increase performance. Presumably because guaranteeing deadlock-freedom takes precedence and consumes most of the available VLs. Our proposed heuristic in conjunction with low-diameter topologies is able to change that: Not only do we require less VLs to provide deadlock-freedom and thus can offer more VLs to segregate different classes of traffic, but also the DF-DN heuristic increments the VL at every hop. The arbiter in InfiniBand switches is able to prioritize packets according to their VL, thus DF-DN can be used to implement age-based arbitration [23], which has been shown to improve the performance by up to 37% [24], but has not been evaluated yet on InfiniBand networks.

References

[1] InfiniBand Trade Association, *InfiniBand Architecture Specification: Release 1.0*. InfiniBand Trade Association, 2000.

[2] C. J. Glass and L. M. Ni, "The turn model for adaptive routing," *ACM SIGARCH Computer Architecture News*, vol. 20, no. 2, pp. 278–287, 1992.

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

[4] T. Hoefler, T. Schneider, and A. Lumsdaine, "Optimized routing for large-scale InfiniBand networks," in *High Performance Interconnects, 2009. HOTI 2009. 17th IEEE Symposium on*. IEEE, 2009, pp. 103–111.

[5] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *Computers, IEEE Transactions on*, vol. 100, no. 5, pp. 547–553, 1987.

[6] J. Duato, S. Yalamanchili, and L. M. Ni, *Interconnection networks: an engineering approach*. Morgan Kaufmann, 2003.

[7] E. Tasoulas, E. G. Gran, B. D. Johnsen, and T. Skeie, "A novel query caching scheme for dynamic InfiniBand subnets," in *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*. IEEE, 2015, pp. 199–210.

[8] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine *et al.*, "Open MPI: Goals, concept, and design of a next generation MPI implementation," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface*. Springer, 2004, pp. 97–104.

[9] I. S. Gopal, *Interconnection Networks for High-performance Parallel Computers*, I. D. Scherson and A. S. Youssef, Eds. Los Alamitos, CA, USA: IEEE Computer Society Press, 1994.

[10] A. Daryin and A. Korzh, "Early evaluation of direct large-scale InfiniBand networks with adaptive routing," *Supercomputing Frontiers and Innovations*, vol. 1, no. 3, pp. 56–69, 2014.

[11] M. Besta and T. Hoefler, "Slim Fly: A cost effective low-diameter network topology," Nov. 2014, Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC14).

[12] D. F. Williamson, R. A. Parker, and J. S. Kendrick, "The box plot: a simple visual method to interpret data," *Annals of internal medicine*, vol. 110, no. 11, pp. 916–921, 1989.

[13] J. Kim, W. J. Dally, S. Scott, and D. Abts, "Technology-driven, highly-scalable dragonfly topology," in *ACM SIGARCH Computer Architecture News*, vol. 36, no. 3. IEEE Computer Society, 2008, pp. 77–88.

[14] G. Kathareios, C. Minkenberg, B. Prisacari, G. Rodriguez, and T. Hoefler, "Cost-effective diameter-two topologies: Analysis and evaluation," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '15. New York, NY, USA: ACM, 2015, pp. 36:1–36:11. [Online]. Available: http://doi.acm.org/10.1145/2807591.2807652

[15] M. Valerio, L. Moser, and P. Melliar-Smith, "Using fat-trees to maximize the number of processors in a massively parallel computer," in *Proceedings of the 1993 International Conference on Parallel and Distributed Systems*. Citeseer, 1993, pp. 128–134.

[16] J. Domke, T. Hoefler, and S. Matsuoka, "Fail-in-place network design: Interaction between topology, routing algorithm and failures," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 597–608. [Online]. Available: http://dx.doi.org/10.1109/SC.2014.54

[17] J. C. Sancho, A. Robles, and J. Duato, "Effective strategy to compute forwarding tables for InfiniBand networks," in *Parallel Processing, 2001. International Conference on*, Sept 2001, pp. 48–57.

[18] T. Skeie, O. Lysne, and I. Theiss, "Layered shortest path (LASH) routing in irregular system area networks," in *IPDPS*. IEEE, 2002, p. 162.

[19] J. Domke, T. Hoefler, and S. Matsuoka, "Routing on the dependency graph: A new approach to deadlock-free high-performance routing," in *Proceedings of the 25th Symposium on High-Performance Parallel and Distributed Computing (HPDC'16)*, Jun. 2016.

[20] H. Rosenstock, "Update on scalable sa project," in *10th Annual OpenFabrics International Developer Workshop*, 2014. [Online]. Available: https://www.openfabrics.org/images/eventpresos/workshops2014/DevWorkshop/presos/Wednesday/pdf/04_ssa.pdf

[21] F. J. Alfaro, J. L. Sánchez, and J. Duato, "QoS in InfiniBand subnetworks," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 15, no. 9, pp. 810–823, 2004.

[22] F. J. Alfaro, J. L. Sanchez, M. Menduia, and J. Duato, "A formal model to manage the InfiniBand arbitration tables providing QoS," *Computers, IEEE Transactions on*, vol. 56, no. 8, pp. 1024–1039, 2007.

[23] R. S. Passint, G. M. Thorson, and T. Stremcha, "Age-based network arbitration system and method," 2004, US Patent 6,674,720.

[24] D. Abts and D. Weisser, "Age-based packet arbitration in large-radix k-ary n-cubes," in *Supercomputing, 2007. SC'07. Proceedings of the 2007 ACM/IEEE Conference on*. IEEE, 2007, pp. 1–11.