Fast Deadlock-free Routing Reconfiguration for Arbitrary Datacenter Networks

July 14, 2016 2:50 AM

ABSTRACT

RDMA is being deployed in DCNs for the benefit of ultra-low latency, high throughput and low CPU overhead in recent years. Current practice of RDMA deployment introduces the deadlock problem as it requires PFC to provide a lossless L2 network. While deadlock can be avoided by using a routing function that includes no cyclic buffer dependency, in this paper we demonstrate that for both tree based and non-tree based DCNs, reconfiguration-induced deadlock could still occur during the routing reconfiguration process even if the routing functions are deadlock-free.

Deadlock-free routing reconfiguration can be ensured by simply diving the reconfiguration process into multiple static stages. However, it could lead to a very slow routing reconfiguration as many unnecessary constraints on the ordering of update actions are introduced. Motivated by this, in this paper, we develop an approach for achieving fast deadlock-free routing reconfiguration which introduces much less constraints on the ordering and can significantly speed up the routing reconfiguration process.

1. INTRODUCTION

The growing demand for online services and cloud computing has driven today's datacenter networks (DCNs) to a large scale with hundreds of thousands of servers and thousands of switches. With this enormous number of network devices, network failure and device upgrade become the norm rather than the exception.

Network reconfiguration will be needed when there is failure or upgrade of links/nodes, new switch onboarding, load balancer reconfiguration, etc. To support this, the network's routing function, which includes all the paths packets can take in the network, are often needed to be reconfigured for the purpose of either maintaining the connectivity of the network or better serving the current network traffic.

On the other hand, as DCNs enter the 40/100Gbps era, RDMA is currently being deployed for achieving ultra-low latency, high throughput and low CPU overhead. To enable efficient operation, RDMA usually runs over a lossless L2 network. The using of a lossless L2 network introduces the deadlock problem into the DCNs, which refers to a standstill situation where a set of switch buffers form a permanent cyclic waiting dependency and no packet can get drained at any of these buffers. Once deadlock occurs, no packet can be delivered through a part of or even the whole DCN.

Under static circumstances (i.e., when both of the network topology and the routing function are fixed), deadlock can be avoided by using a routing function that contains no cycle in the corresponding buffer dependency graph.

Under dynamic circumstances, however, deadlock may occur during reconfiguration process when transitioning from an old deadlock-free routing function R_s to a new deadlock-free routing function R_t. This is because during the routing reconfiguration process, due to the asynchronous updates of switch rules, any paths included in $R_s \cup R_t$ may take effect at the same time. When $R_s \cup R_t$ contains a cycle in the corresponding buffer dependency graph, deadlock may occur if the routing reconfiguration process is not well planed. We refer to this kind of deadlock as *reconfiguration-induced deadlock*.

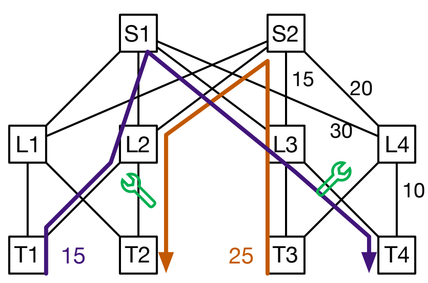
Reconfiguration-induced deadlock can be avoided by imposing some constraints on the ordering of configuration actions during the reconfiguration process. For example, deadlock-free can be guaranteed by removing all the paths included in R_s first before adding any new path included in R_t. Alternatively, we can remove some paths in R_s to reduce the routing function into $R_s \cap R_t$ at first, and then add the new paths included in R_t to finish the reconfiguration procss.

The speed of routing reconfiguration is important as it determines the response time to a network failure. Although both of the above approaches can ensure deadlock-free, they will lead to a slow routing reconfiguration process as multiple staitc intermediate stages are needed.

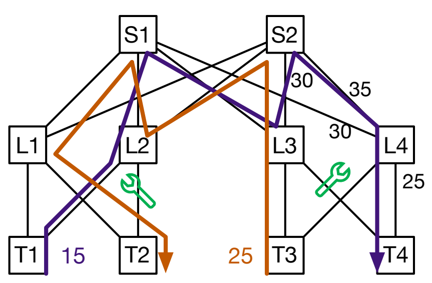
In this paper, we develop an approach for achieving fast deadlock-free routing reconfiguration. It is based on two observations: 1) there exist multiple valid orderings that is deadlock-free; and 2) choosing an ordering with minimum order dependencies among configuration actions can lead to fast reconfiguration. Our approach is general and can be applied to arbitrary DCNs, including Fat-tree, VL2, HyperX, Jellyfish, etc.

2. BACKGROUND AND MOTIVATION

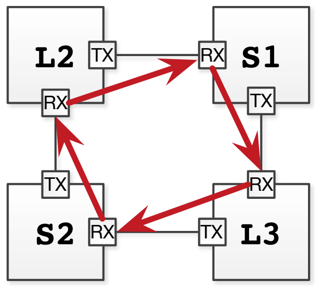
2.1 PFC Deadlock Problem



(a) Routing state before reconfiguration.



(b) Routing state after reconfiguration.



(c) Buffer dependencies among ingress queues.

Figure 1: Reconfiguration-induced deadlock case for leaf-spine topology.

Priority-based Flow Control (PFC): The deployment of RDMA over Ethernet requires PFC to provide a lossless L2 network. PFC is a mechanism for ensuring zero packet loss under congestion in data center bridging (DCB) networks. PFC allows an overwhelmed network device to send a PAUSE frame to its immediate upstream device, which halts the transmission of the sender for a specified period of time.

PFC works in a per ingress queue fashion. When PFC is enabled, the switch will maintain a counter to track the virtual queue length of each ingress queue. Once the queue length exceeds a pre-configured PFC threshold, a PAUSE frame will be generated.

PFC deadlock problem: The using of PFC can cause deadlock problem. Deadlock may arise when there is cyclic buffer dependency in the network. Once a PFC deadlock is created, a set of ingress queues form a permenant pause cycle, and no packet is allowed to be transmitted inside the cycle.

2.2 Reconfiguration-induced Deadlock

PFC deadlock can be avoided by leveraging a routing function that introduces no cycle in the buffer dependency graph. However, this approach cannot eliminate the cyclic buffer dependency during routing reconfiguration.

In this part, we use examples to show 1) cyclic buffer dependency can be generated for both tree based and non-tree based DCNs when the routing reconfiguration is not well planed; 2) a bad deadlock-free reconfiguration plan will lead to a slow reconfiguration process.

2.2.1 *Deadlock Under Tree Based DCNs*

Fig. 1(a) shows a small Leaf-Spine DCNs. The capacity of all the links are 40Gbps. Due to maintenance issue, the network operator now wants to replace two links L2-T2 and L3-T3 in the topology. To avoid long-term packet loss during link replacement, the network traffic passing through these two links needs to be migrated to some other paths.

In this example, we consider the migration of the traffic from switch T1 to switch T4 following the path T1-L2-S1-L3-T4, and the traffic from switch T3 to switch T2 following the path T3-L3-S2-L2-T2.

There are three alternative paths that the traffic from T1 to T4 can migrate to: 1) *path-1* along T1-L1-S1-L4-T4; 2) *path-2* along T1-L2-S1-L4-T4; 3) non-shortest *path-3* along T1-L2-S1-L3-S2-L4-T4. As we can find in Fig. 1(a), the traffic load from T1 to T4 is 15Gbps. *Path-1* and *path-2* are not congestion-free choices as the load on link S1-L4 are 30Gbps, larger than 25Gbps. *Path-3* is a congestion-free choice as the load on links L3-S2, S2-L4 and L4-T4 are all smaller than 25Gbps.

While not explicitly drawn in the Fig. 1(a), non-shortest path T3-L3-S2-L2-S1-L1-T2 is also the only congestion-free choice that the traffic from T3 to T2 can migrate to. Fig. 1(b) shows the routing state after migrating the traffic to the two non-shortest paths.

In Fig. 1(c), we focus on the buffer dependency among four switches L2, L3, S1 and S2. We reposit the locations of these four switches and draw both ingress queues (RX) and egress queues (TX) for the purpose of better explanation. The buffer dependency introduced by the two non-shortest paths in Fig. 1(b) are drawn with directed lines. As we can see, there is a cyclic buffer dependency among the ingress queues. This indicates that the network is now exposed to the danger of PFC deadlock.

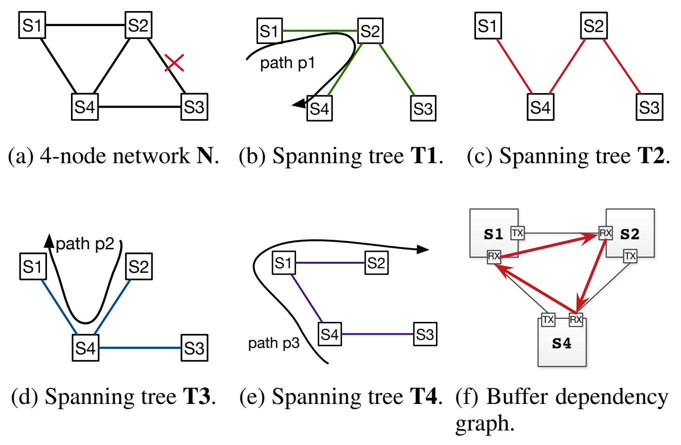
To avoid possible PFC deadlock problem, one safe congestion-free reconfiguration plan is to replace the two links one by one. Only using one non-shortest path will not introduce cyclic buffer dependency into the network.

2.2.2 *Deadlock Under Non-tree Based DCNs*

As shown in Fig. 2(a), in this example we consider a 4-node network **N**. This topology can be a subgraph of many non-tree based DCNs, like HyperX, Jellyfish and BCube.

Fig. 2(b)-(e) are four spanning trees **T1**-**T4** which specify the routing paths that can be used in **N**. For example, path p1 is a legal routing path specified in **T1**.

Let $\mathbf{R}_i$ be the set of paths specified in tree **Ti**. Let $\mathbf{R}_s = \mathbf{R}_1 \cup \mathbf{R}_2$, and $\mathbf{R}_t = \mathbf{R}_3 \cup \mathbf{R}_4$. It is easy to check both $\mathbf{R}_s$ and



(a) 4-node network **N**. (b) Spanning tree **T1**. (c) Spanning tree **T2**. (d) Spanning tree **T3**. (e) Spanning tree **T4**. (f) Buffer dependency graph.

Figure 2: Reconfiguration-induced deadlock case for non-tree topology.

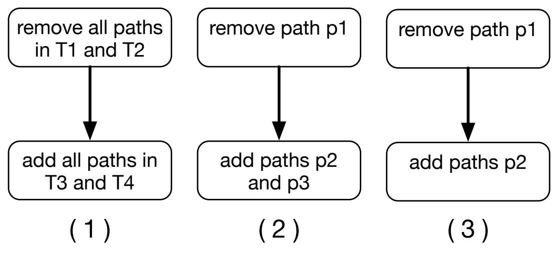


Figure 3: Three deadlock-free reconfiguration schemes.

$\mathbf{R}_t$ are deadlock-free routing functions. Initially, $\mathbf{R}_s$ are used as the routing function of **N**. Due to the failure of link S2-S3, switch S3 becomes unreachable. To maintain the connectivity of **N**, we can perform a routing reconfiguration to transition from $\mathbf{R}_s$ to $\mathbf{R}_t$.

During the reconfiguration process, if path p2 in **T3** and path p3 in **T4** are added to the routing function before path p1 in **T1** is removed, a cyclic buffer dependency will be generated, as shown in Fig. 2(f). This may cause a PFC deadlock as we explained in Sec. 2.1.

In Fig. 3, we present three possible deadlock-free reconfiguration schmes. The first scheme is to remove all the paths in **T1** and **T2** before adding any new paths in **T3** and **T4**. This scheme will lead to a slow reconfiguration process as all the operations of adding new paths are delayed by the operations of removing old paths.

The second scheme only requires path p1 is removed before paths p2 and p3 are added. All the other paths not mentioned can be updated freely without any order constraint. Hence the speed of routing reconfiguration can be improved. The third scheme is an optimized reconfiguration scheme in terms of imposing minimum order constraints on the update actions. The intuition here is that as long as paths p1, p2 and p3 do not take effect at the same, deadlock-free can be well guaranteed.

$G(V,E)$	The DCN, where V is the set of all nodes and E is the set of all links.
C	$C \subset G(V,E)$ is a cycle in $G(V,E)$.
P_s	The set of paths in the old configuration.
P_t	The set of paths in the new configuration.
R_p	The set of rules corresponding to path p.
$G_c(V_c,E_c)$	A configuration dependency graph, where V_c is a set of configuration operations, and E_c is a set of order constraints.
P_c	The set of configuration paths in G_c.
t_o	The time to finish an operation $o \in V_c$.
$t(P,G_c)$	The time to configure all paths in P obeying the dependencies in G_c.
$ts(G_c)$	A topological sorting of G_c, which is a list of configuration operations.
$TS(G_c)$	The set of all possible $ts(G_c)$.
$P^{(i)}(ts)$	The set of active paths after finishing first i-th operations in $ts(G_c)$.
$d^P_{lx,ly}$	The buffer dependency edge from link l1 to link l2 introduced by the paths in P.
$P^d_{lx,ly}$	The set of all paths in P contributed to $d^P_{lx,ly}$.

Table 1: The key notations used in the problem formulation.

While for this example it may seem easy to find a deadlock-free reconfiguration scheme that requires minimum order constraints, in general it is difficult as there are combinatorial such schemes to be checked.

2.3 Measurement of Rule Update Time

In this part, we demonstrate that adding order constraints to the update of switch rules will significantly prolong the reconfiguration process.

3. SOLUTION

In this part, we present our preliminary solution for achieving fast deadlock-free routing reconfiguration.

3.1 Problem Formulation

In Table 1, we list the key notations used in our problem formulation. $G(V,E)$ is the DCN. C is a cycle in $G(V,E)$. P_s is the set of old routing paths, while P_t is the set of new routing paths. Here we assume $P_s \cap P_t = \emptyset$, which can always be achieved by removing all the paths in $P_s \cap P_t$ in advance.

$G_c(V_c,E_c)$ is a configuration dependency graph (CDG), where V_c is a set of configuration operations, and E_c is a set of order constraints. Fig. 4 shows an example of CDG. In the graph, each node represents a configuration operation. For example, node $o1$ represents the operation to remove path $p1$, while node $o3$ represetns the operation to add path $p3$. Each directed edge in the graph represents an order con-

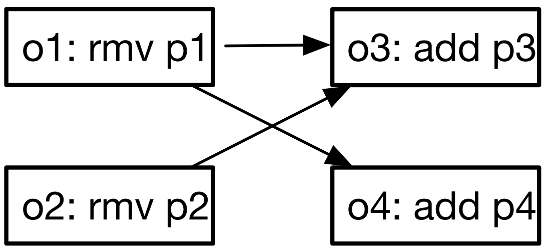


Figure 4: An example of configuration dependency graph.

straint on the operations. For example, $o1$ must be finished before we start the operation $o4$

P_c is the set of configuration paths in G_c. In Fig. 4, there are three legal configuration paths: 1) o1-o3; 2) o1-o4; 3) o2-o3. We use t_o to denote the time to finish an operation o in V_c. The time to finish an configuration path is the sum of the time to finish any single operation on the path. $t(P, G_c)$ is the time to configure all routing paths in P with respect to the dependency contrsints of G_c. The value of $t(P, G_c)$ is determined by the bottleneck configuraton path in G_c which requires longest time to finish.

We use $ts(G_c)$ to denote a topological sorting of G_c. $ts(G_c)$ represents a possible order of configuration operations in terms of the finish time. $TS(G_c)$ is the set of all possible topological sortings in G_c. In Fig. 4, there are five possible topological sortings: (o1, o2, o3, o4), (o1, o2, o4, o3), (o1, o4, o2, o3), (o2, o1, o4, o3) and (o2, o1, o3, o4). $P^{(i)}(ts)$ is the set of active routing paths after first i-th operations in $ts(G_c)$ is finished.

We use $d^P_{lx,ly}$ to denote the buffer dependency from link lx to link ly introduced by the paths in P. Note that each link in a DCN is exactly corresponding to an ingress queue. For simplicity, we use a pair of links to denote the buffer dependency among a pair of ingress queues. We define

$$d^P_{lx,ly} = \begin{cases} 1, & \text{links } lx \text{ and } ly \text{ are adjacent, and } \exists p \in P \\ & \text{that goes over } lx \text{ and } ly \text{ in sequence.} \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

Given a DCN topology $G(V, E)$, an old path set P_s, a new path set P_t and a CDG $G_c(V_c, E_c)$, we say $G_c(V_c, E_c)$ is a deadlock-free CDG for the reconfiguration from P_s to P_t when the following condition is met: for any legal topological sorting $ts(G_c)$, at any reconfiguration state $P^{(i)}(ts)$, there is no cyclic buffer dependency for any cycle C in $G(V, E)$. This condition can be formally described as

$$\forall ts \in TS(G_c), \forall P^{(i)}(ts), \forall C \subset G(V, E), \prod_{\forall lx,ly \in V(C)} d^{P^{(i)}(ts)}_{lx,ly} = 0 \tag{2}$$

For an input ($G(V, E)$, P_s, P_t), The goal of our solution is to find a deadlock-free CDG $G_c(V_c, E_c)$ with minimal reconfiguration time $t(P, G_c)$.

3.2 Fast Deadlock-free Reconfiguration For a Single Cycle

In this part, we present our solution for constructing a deadlock-free CDG $G_c(V_c, E_c)$ for a single topology cycle C in $G(V, E)$.

The naive approach to find an optimal deadlock-free CDG is to enumerate all the possible CDGs, and choose a deadlock-free CDG with minimum configuration time. However, it would be computationally impossible as there are combinatorial such CDGs.

Our solution is designed based on the observation that *For a single topology cycle with cyclic buffer dependency, as long as we guarantee that one old buffer dependency edge is removed from the cycle before one different new buffer dependency edge is added to the cycle, the reconfiguration process will be deadlock-free.*

In the next, we describe how our solution works. For each pair of adjacent links lx and ly in cycle C, let $P^{lx,ly}_s$ be the set of paths in P_s contributed to the buffer dependency edge $d^{P_s}_{lx,ly}$, and $P^{lx,ly}_t$ be the set of paths in P_t contributed to the buffer dependency edge $d^{P_t}_{lx,ly}$. Removing all paths in $P^{lx,ly}_s$ will delete buffer dependency edge $d^{P_s}_{lx,ly}$ from the network, while adding paths in $P^{lx,ly}_t$ will create buffer dependency edge $d^{P_t}_{lx,ly}$.

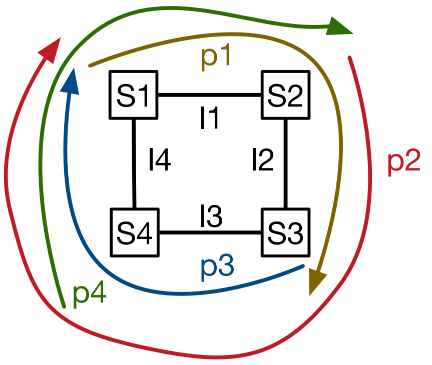
For any non-empty set $P^{lx,ly}_t$, as P_t is deadlock-free, there exists at least one

Given two non-empty sets $P^{lx,ly}_s$ and $P^{lm,ln}_t$ that satisfy $(lx, ly) \neq (lm, ln)$, to guarantee one old buffer dependency edge is removed before one different new buffer dependency edge is added, we can let all the paths in $P^{lx,ly}_s$ be removed from the network before adding any path in $P^{lm,ln}_t$. In other word, we can construct a deadlock-free CDG by adding a configuration dependency edge from any path in $P^{lx,ly}_s$ to any path in $P^{lm,ln}_t$.

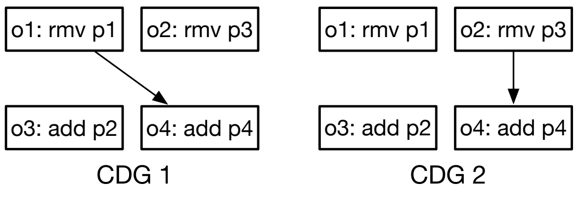
In the next, we present our solution for searching deadlock-free CDGs with minimum time cost. Assuming that we know the time cost to remove or add a single path. Then we can calculate the time cost of path configurations for any $P^{lx,ly}_s$ and any $P^{lm,ln}_t$. To find the optimal CDG(s), our solution will calculate the time cost of path configurations for any legal pair ($P^{lx,ly}_s$, $P^{lm,ln}_t$), and then choose the one with minimum time cost.

Let k be the number of links in C, and n be the number of paths contributed to the buffer dependency in C. The time complexity of the above calculation is within $O(nk + k^2)$. Hence our solution is scalable.

In Fig. 5, we use a simple example to illustrate how our solution works. In this example, $P_s = \{p1, p3\}$ and $P_t = \{p2, p4\}$. As we can see in Fig. 5(a), paths in $P_s \cup P_t$ introduce a cyclic buffer dependency to the network. It is easy



(a) Topology and paths.



(b) Two optimal CDGs.

Figure 5: An example to illustrate the solution. The network topology is a 4-node cycle. $P_s = \{p1, p3\}$ and $P_t = \{p2, p4\}$. In this example, we assume the time cost to configuring a path is equal to the number of switches this path traverses in the cycle.

to know $P_s^{l1,l2} = \{p1\}$, $P_s^{l3,l4} = \{p3\}$, $P_t^{l2,l3} = \{p2\}$, $P_t^{l3,l4} = \{p2\}$ and $P_t^{l4,l1} = \{p4\}$.

Let $t_{add}(p)$ and $t_{rmv}(p)$ be the time cost to add and remove a path p, respectively. In this example, we assume the time cost to configuring a path is equal to the number of switches this path traverses in the cycle. So we have $t_{rmv}(p1) = 3$, $t_{add}(p2) = 4$, $t_{rmv}(p3) = 3$ and $t_{add}(p4) = 4$.

Our solution will calculate the time cost for all the path sets contributed to some buffer dependency edge. The results are as follows $t(P_s^{l1,l2}) = 3$, $t(P_s^{l3,l4}) = 3$, $t(P_t^{l2,l3}) = 4$, $t(P_t^{l3,l4}) = 4$, $t(P_t^{l4,l1}) = 3$.

Our solution will then calculate the time cost of path configurations for any pair ($P_s^{lx,ly}$, $P_t^{lm,ln}$), and then choose one optimal pair with minimum time cost. In this example, both ($P_s^{l1,l2}$, $P_t^{l4,l1}$) and ($P_s^{l3,l4}$, $P_t^{l4,l1}$) have minimum time cost, so two optimal CDGs can be constructed, as shown in Fig. 5(b).

4. EVALUATION

to be added. In this part, we evaluate the performance of our solution via simulations.

Topology: 4-level Fat-tree, HyperX, Jellyfish, etc.

Model of switch rule update: parallel update, sequential update, etc. We also need to model the delay of control messages in our simulator.

5. RELATED WORKS

to be added.

6. REFERENCES