Roundabout: Solving PFC Deadlocks with Distributed Detection and Buffer Collaboration

Kai Lv*‡, Heng Pan†, Chengjun Jia§, Jiaxing Zhang*‡, Luyang Li*‡, Jianer Zhou¶, Yanbiao Li†
Zhenyu Li*, Gaogang Xie†
*Institute of Computing Technology, Chinese Academy of Sciences, China
†Computer Network Information Center, Chinese Academy of Sciences, China
‡University of Chinese Academy of Sciences, China §Tsinghua University ¶Peng Cheng Laboratory, ShenZhen, China

***Abstract*—RDMA over Converged Ethernet (RoCEv2) employs Priority-based Flow Control (PFC) for a lossless fabric to maintain high performance. However, PFC can cause *Deadlock*s, which pauses traffic and potentially leads to severe exceptions for applications. Existing solutions solve deadlocks at a considerable cost, resulting in degradation of end-to-end network performance.**

We present Roundabout, a data plane scheme designed to detect and resolve deadlocks with minimal side effects. We first analyze how switches in different states contribute to deadlocks. Based on the analysis, we design an election-based distributed detection scheme that efficiently and robustly identifies deadlocks. By exploiting buffer configuration redundancy, we develop an in-network collaborative packet scheduling scheme that forwards deadlocked packets to their destinations in a lossless manner, facilitating natural deadlock resolution. Additionally, we implement a barrier mechanism to ensure in-order packet delivery to the receiver. Both analysis and experiments demonstrate that Roundabout effectively detects and resolves deadlocks while minimizing side effects to the network, making it an ideal enhancement for PFC switches.

I. Introduction

Remote Direct Memory Access (RDMA) has become the de-facto standard for high-speed networks in modern data centers due to its high throughput, low latency and low CPU overhead. RoCEv2 (RDMA over Converged Ethernet Version 2), a major branch of RDMA[1], has been deployed at scale [1–3] and is actively embraced by a wide range of applications, such as storage [3, 4], computing [5], distributed transactions [6], and large-scale AI training [7].

RoCEv2 maintains a lossless network fabric by employing Priority-based Flow Control (PFC) [1, 8]: when an ingress queue grows and exceeds a threshold X_{off}, a PAUSE frame is sent upstream to stop the packet transmission, and a *headroom* buffer is used to absorb the in-flight packets; after the queue drains below another threshold X_{on}, a RESUME frame is sent upstream to continue packet transmission. Various modern congestion control schemes leverage PFC as the last line of defense against packet loss [1, 9].

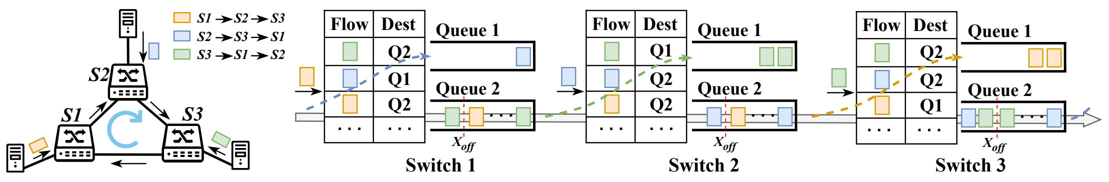
However, PFC inherently introduces *deadlock*s, a phenomenon of cyclic buffer dependency (CBD), where all switches within a cycle pause the packet transmission of their upstream switch port, while their own packet transmission are also paused by their downstream switches [1]. Packets trapped in the deadlock loop are paused constantly, severely impacting network performance and availability [3]. Switches in deadlock can also flood PAUSE frames to other innocent links and devices, potentially affecting the entire network. Modern data centers widely adopt the Clos topology (e.g., fat-tree) [1, 3, 4, 10]. In general, deadlocks are rare in Clos, until some less common events (e.g., link failure or port down/flap) happen and incur loops (see details in §VIII). Meanwhile, some other topologies are more prone to deadlocks [11]. As networks scale rapidly [12], the frequency of deadlock occurrences may increase.

To this end, extensive research efforts have focused on PFC deadlocks [13–20], which can be broadly classified into two categories: proactive deadlock avoidance and reactive deadlock detection/resolution. Deadlock avoidance strategies aim to break the essential conditions of deadlocks to prevent their occurrence in advance. However, they are often tightly coupled with specific network topologies [21], routing algorithms [14], or traffic patterns [18], limiting their flexibility. In contrast, detection and resolution schemes relax these constraints, allowing deadlocks to occur. As a result, they need to detect and solve deadlocks with high accuracy and efficiency to minimize their impact on network performance.

Existing deadlock detection/resolution schemes have their drawbacks (detailed in §II-B): (*i*) For detection, control-plane based methods [19, 22] can not fully meet the performance requirements and may introduce false positives. Recent data-plane solutions [20] may fail when multiple switches trigger one deadlock cooperatively. (*ii*) For resolution, adaptive routing [20] can interfere with other parts of the network; buffer reconfiguration [19, 20] encounters failure when there's no enough free shared buffer. Moreover, since RoCEv2 uses "go-back-N" to retransmit dropped or out-of-order packets [23], schemes that may lead to retransmission (such as switch reset or reconfiguration, packet drop [20] and adaptive routing [19]) will cause performance degradation.

We aim to develop an efficient deadlock detection-resolution scheme with minimal side effects to avoid these issues. The scheme should be independent of network topologies, routing algorithms, or congestion control mechanisms. It should effectively identify deadlocks during their intricate formation and

Chengjun Jia is now at Huawei Technologies.
[1]We use RoCE and RoCEv2 interchangeably in this paper.



(a) Deadlock topology.

(b) An expanded view of the deadlock topology. Ingress is not shown here.

Fig. 1: All packets trapped in the loop have their own exit port (queue). Gray arrow indicates the deadlock loop.

resolve them with minimal effect on innocent flows, ports, or queues. it should also preserve packet order for flows to prevent unnecessary retransmissions in RoCE environments. This is motivated by two key insights.

Our first key insight is that for a packet already stalled in a deadlock loop, it only seeks to traverse along its in-loop forwarding path to find its **exit switch** where it can be forwarded off the loop via the corresponding **exit port (queue)**, just as they are traversing a *Roundabout*. Fig. 1 shows an example. Packets in yellow should leave the loop from switch 3, queue 1. If packets can be scheduled to their *exit ports* in a lossless manner, queue length can be reduced and the deadlock is resolved.

Our second key insight is that, headroom buffer is typically configured larger than its theoretical value, which can be better utilized to resolve deadlocks. Although headroom size can be theoretically calculated [24–26], obtaining the necessary metrics can be more complex (§II-A). For example, metrics like interface delay and high-level delays are vendor and implementation dependent [27]. Therefore, headroom size is typically configured slightly larger to accommodate "plug-and-play" functionality, as is the default settings provided by manufacturers [27]. The redundant buffer space is unused in most cases, which can be used for better deadlock resolution.

While our insights are straightforward, leveraging them to solve deadlocks is still challenging. First, deadlock detection algorithms must contend with the frequent state transitions in switches during deadlock formation (detailed in §III). Second, robust deadlock resolution necessitates efficient management and scheduling of switch buffers, which calls for well-designed buffer collaboration mechanisms. Third, the inherent absence of packet order records in switches must be addressed so that packets can be scheduled to their destinations in the correct order, without retransmission.

In this paper, we propose Roundabout, a novel and efficient **data plane** solution that detects and resolves deadlocks with minimal side-effects. Roundabout is compatible with any topologies, routing schemes and flow control algorithms, and it operates entirely in the data plane at line rate. In deadlock detection phase, (*i*) we classify switches forming a deadlock into **three states**, and design an ***election-based* distributed detection scheme** triggered bu PAUSE events. By continuously monitoring deadlock signals from local queues and downstream switches and maintain the more advantageous results, A switch is able to closely monitor the transient state of in-loop ports, facilitating swift and accurate deadlock detection (§III). In deadlock resolution phase, (*ii*) we **leverage the idea of *buffer collaboration***, both between and within switches, to break the logical boundaries of switch buffers and effectively schedule packets towards their exit ports (§IV). This process only utilizes the links that are already deadlocked, therefore does not affect the rest of the network. Moreover, it does not require buffer reconfiguration, avoiding possible interference with queued packets. (*iii*) We also **design a novel *barrier mechanism*** to isolate packets originated from different deadlocked switches and propose a scheduling scheme to ensure packets exit the loop through the correct port (§IV-D).

We experimentally demonstrate the overall effect of Roundabout and compare it with other state-of-the-art solutions. The results show that Roundabout outperforms other schemes across multiple metrics, and is suitable to serve as a functional building block to enhance PFC-supported switches (§VI).

II. Background

A. Dynamic Thresholds Switches for RoCE

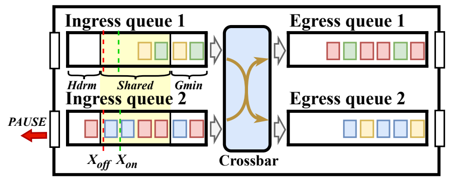


Fig. 2: A Dynamic Threshold (DT) switch supporting RoCE. Buffer in yellow is shared among ports and queues.

As shown in Fig. 2, for PFC-enabled switches, ingress queue typically consists of three parts. First, each queue has a dedicated buffer (Guaranteed Minimum, *Gmin*) to prevent starvation. Then, a *Shared* buffer is dynamically allocated from the global shared buffer with algorithm such as Dynamic Thresholds (DT) [28–31]. Finally, a headroom buffer (*Hdrm*) is reserved to absorb in-flight packets that arrive after PFC PAUSE is asserted. Physically, the "Ingress queue" is essentially a *Virtual Statistics Queue* (VSQ) maintained by a set of counters [29, 32, 33], which record the occupancy for each part of the queue. Buffer management algorithms rely on these counters for their operation.

Upon packets arrival, they attempt to be admitted to Gmin, then to Shared. If they both fail and trigger PFC PAUSE, packets are then absorbed by Hdrm. Headroom should at least be able to absorb: (*i*) maximum packet size (9kB for jumbo frames), (*ii*) bit rate×round-trip latency (2x BDP), and (*iii*) bit rate×internal latencies within the switch. Headroom buffer size is additionally infected by the transmission speed of bits across diverse wires, transceiver latency, internal implementation-dependent response time, cell size, buffer management strategy, etc. [24, 29], making it hard to obtain the headroom size precisely. In practice, headroom buffer is configured to be larger than its theoretical value (e.g., default value) to prevent possible accidental packet losses [25, 26, 29], which results in *waste* of buffer. These buffer should be utilized more efficiently, but few have explored it.

B. Causes of Deadlocks and Limitations of Existing Solutions

Studies have revealed many causes of deadlock, including infrastructure and configuration issues such as hardware failures, software bugs, misconfigurations and network updates [34–36]. Deadlocks can also stem from traffic forming transient loops [17], which may occur more frequently under bursty and concurrent scenarios, such as the "All-to-All" or "ring All-reduce" traffic [37] in distributed machine learning when hosts perform data synchronization. In DT switches, another reason for a queue to exceed its X_{off} threshold (a prerequisite condition of deadlock) is the inability for Shared to obtain sufficient space from the global shared buffer.

A common method for deadlock detection is to periodically probe port states and assert deadlock by checking for Circular Buffer Dependency (CBD) in the control-plane [19, 22]. However, performing quick deadlock detection is challenging due to the inherent latency between the control plane and data plane. Moreover, since CBD is a loose condition for deadlocks, asserting deadlocks by checking for the presence of CBD may incorrectly assert a deadlock that doesn't actually exist (false positive) [17]. Recent work [20] shows the performance advantage of detecting deadlocks in the data-plane. When a switch initiates a pause frame (and becomes an initiator), a probing message is piggybacked and is sent along the loop to check the ingress queue length of each switch it passes through. If they all exceed their X_{off} thresholds, a deadlock is identified. However, as shown in Fig. 3, when multiple initiators jointly trigger a deadlock (which is common), the probing message will be dropped by other initiators, which makes this procedure not robust.

Once a deadlock is detected, it is typically resolved by packet drop [20], adaptive routing or buffer reconfiguration [19, 20]. However, these methods have drawbacks. Resetting links and ports can drop packets in the loop, which is disruptive and inelegant. A recent scheme prioritizes dropping packets from elephant flow, which mitigates the impact of packet loss but does not completely eliminate it. Adaptive routing does not cause packet loss, but it interferes with other innocent traffic by forwarding packets to other links. Dynamic buffer reconfiguration still needs help from control-plane and may lead to packet drops in the chip buffer. It not only occupies buffer space that could be utilized by other queues but often fails to allocate enough space from the shared buffer as well. Furthermore, RoCE uses "Go-Back-N" retransmission, which is sensitive to dropped or out-of-order packets. None of these methods can preserve packet order, leading to degradation in network performance. Although Infiniband [38] and some certain NICs [39] support out-of-order packet delivery, it is impractical to assume such capability in RoCE-based datacenters composed of heterogeneous NICs [40].

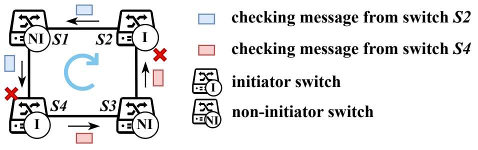

S1
S2
S3
S4
NI
I
I
NI
checking message from switch S2
checking message from switch S4
initiator switch
non-initiator switch

Fig. 3: Simple back-propagation cannot detect deadlock robustly.

III. Distributed Deadlock Detection

As we can see in §II-B and Fig. 3, detection triggered by PAUSE events has to handle transient and complex loop states.

Insight. Switches forming a deadlock loop can be classified into three categories, based on the status of their relevant ports and queues. *Trigger*s are the switches that actively bring new PAUSE signals to the loop. They pause their upstream switches due to incast traffic or PAUSE frames from outside the loop, *etc*. *Spreader*s are the switches paused by other in-loop switches. They may propagate pause signals upstream in the loop as their queues build up. *Non-pause*s are the switches that currently do not receive or generate in-loop pauses. In a loop of N switches forming a deadlock, there could be 3^N possible loop states at any given time.

Relying on simple back propagation to detect deadlocks can be insufficient in many cases. For instance, each switch in the loop may independently trigger PFC and pause its upstream switch during deadlock formation, consequently interrupting the existing detecting process; during the formation of a deadlock, switches may transition between three states due to buffer changes, which might interrupt the propagation of probing messages; when a switch transition to non-pause state, their probing message becomes outdated, *etc*.

Core idea. Deadlock detection in the data plane can be viewed as a consensus problem. Our objective is to identify deadlocks based on the status of switches within a loop and achieve consensus across them. To this end, we propose an *election-based* distributed detection scheme, which performs on-the-fly election at each hop. Probing messages are carried in PFC frames. When a switch generates a new probing message or receives a probing message from another switch, it performs an election to select the more advantageous message and caches it locally. When a switch's status changes, it promptly notifies the affected switches to ensure continuous and accurate tracking of the loop's status. By adhering to the uniform election rule, switches in the loop can ultimately converge on a consistent detection result.

A. Definitions of Data Structure and Packet Format

We define probing message as a triplet $T = \{sid, qid, rand\}$. Here, sid and qid are the global switch ID and queue ID within the switch respectively, uniquely identifying the source of a probing message. The 32-bit $rand$ is randomly generated for comparing the priorities between probing messages. For each ingress queue, we instantiate two triplet registers: T_{init} and T_{curr}, along with two 1-bit flags to indicate their validity. T_{init} records the inherent probing message: when a queue actively triggers PAUSE and generates probing message, T_{init} is initialized and marked valid; when the queue

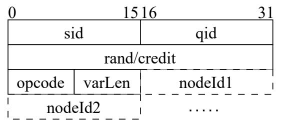


Fig. 4: *ra_hdr* format. $rand$ and $credit$ used in seperated operations shares a 32-bit field to reduce header length.

opcode	packet character / operations	*opcode*	packet character / operations
1	deadlock detection	2	T_{curr} invalidation
3	deadlock announcement	4	accumulate routing info
5	build Roundabout routing table	6	carrying credits
7	barrier packet	8	checking message

TABLE I: *opcode* and the corresponding actions.

Algorithm 1: Functions at Ingress.

```
1  // $T_{init}$ and $T_{curr}$ are registers at ingress queue.
2  // Egress parse the triplet $T$ carried in probing packets and
     synchronize with functions at its casualty ingress queue(s).
3  function Elect($T_1$, $T_2$)
4      if $T_1.\{sid, qid\} = T_2.\{sid, qid\}$ or $T_2$ is not Valid then
5          return $T_1$;
6      else
7          return $T_1.rand < T_2.rand$ ? $T_1$ : $T_2$;
8  function SendPause()
9      InitAndSetValid ($T_{init}$);
10     $T_{curr} \leftarrow$ Elect($T_{init}, T_{curr}$);
11     SetValid ($T_{curr}$);
12     SendPauseWithTuple ($T_{curr}$);
13 function SendResume()
14     SendResumeWithTuple ($T_{curr}$);
15     SetInvalid ($T_{init}$);
16     SetInvalid ($T_{curr}$);
17 function PeerGetPause($T$)
18     if $T_{init}$ is Valid and $T_{init} = T$ then
19         deadlock detected;
20     else
21         if $T_{init}$ is Valid and $T_{curr}$ is not Valid then
22             $T_{curr} \leftarrow$ Elect($T, T_{init}$);
23         else
24             if $T_{curr} = T$ then
25                 return;
26             $T_{curr} \leftarrow$ Elect($T, T_{curr}$);
27         SetValid ($T_{curr}$);
28         if self pauses upstream then
29             SendPauseWithTuple ($T_{curr}$);
30 function PeerGetResume($T$)
31     if self pauses upstream and $T_{curr}.\{sid, qid\}$ = $T.\{sid, qid\}$
         then
32         SetInvalid ($T_{curr}$);
33         SendInvalidWithTuple ($T$);
34         $T_{curr} \leftarrow T_{init}$;
35         SetValid ($T_{curr}$);
36         SendPauseWithTuple ($T_{init}$);
37 function PeerGetInvalid($T$)
38     if $T = T_{curr}$ and $T_{curr}$ is Valid then
39         SetInvalid ($T_{curr}$);
40         SendInvalidWithTuple ($T$);
```

triggers RESUME, T_{init} is invalidated. T_{curr}, on the other hand, is used for tracking loop state changes by caching the most advantageous probing message it observes.

As shown in Fig. 4, Roundabout header (abbreviated as ***ra_hdr***) is designed to convey messages between switches. Switches generate and piggyback *ra_hdr* when triggering PFC pause/resume frames, which are referred to as ***ra_pause*/*ra_resume*** frames. PFC frames with *ra_hdr* used for deadlock detection are called ***probing packets***. Note that *ra_hdr* also contains several other fields in addition to the triplet. These are utilized for different operations throughout the whole deadlock detection and resolution process, which we demonstrate later. Switches parse the *opcode* and act correspondingly. We summarize the value of *opcode* field and their behaviors in table I.

To track deadlock dependency between ingress and egress within a switch, we adopt a data structure similar to the approach in [20]. Specifically, each lossless egress queue maintains a bitmap of length equal to the number of ports N, indicating whether an egress queue contains packets from a specific ingress port. The bitmap is indexed by port ID and updated based on queue statistics and packet size. By querying the bitmap, an egress queue can identify its *causal ingress port*(s) for deadlock detection.

B. Election based Deadlock Detection

We first introduce the *election rule*, the fundamental part of data plane deadlock detection. The core idea is that switches should maintain the most advantageous and up-to-date factors that could lead to deadlocks, which enables distributed deadlock detection to converge on a latest and consistent result.

Election Rule. When a switch observes a probing message (either generated by its own queue exceeding X_{off} threshold, or received from another switch), it performs an election and updates the locally cached message. Algorithm 1 (line $3-7$) demonstrates three election rules. (*i*) Probing message with smaller $rand$ is considered more advantageous (line 7); (*ii*) if a queue triggers multiple probes over time, the latest one is effective (line 4); (*iii*) if no valid probing message was observed previously, the observed message is elected directly.

Detection Scheme. To address detection failures caused by changes in switch states, we design a distributed deadlock detection algorithm. The core idea is that when a switch transitions between the three states, its impact on neighboring switches changes. These changes must be accurately conveyed to other switches in the loop, ensuring that switches in the loop correctly maintain the latest deadlock state. The actions at each switch's ingress queue are summarized in Algorithm 1:

- When a switch becomes a *trigger* or *spreader* and sends a pause frame, it contributes to the deadlock and must inform its upstream of this impact with an *ra_pause frame* with *opcode* $= 1$. Before this, spreaders should run an election to update the most advantageous impact (line 8-12).
- When a switch triggers resume (become a *non-pause*), it no longer contributes to deadlock formation. Therefore, it invalidates its record and removes its impact on its upstream switches hop by hop with an *opcode* $= 2$ *ra*

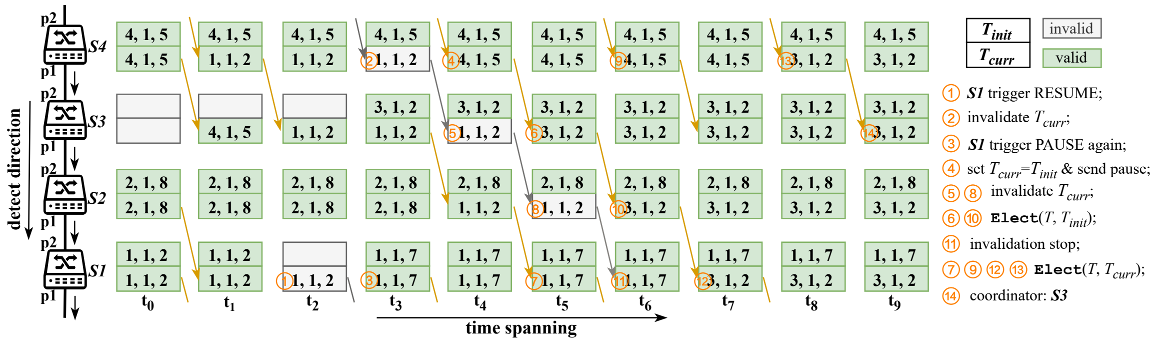


Fig. 5: A concrete example of detecting actions in a 4-switch deadlock. Each cell group represents the register state of corresponding ingress queue (at port p1). The upper and lower half denotes T_{init} and T_{curr} respectively. Arrows indicate the interactions between switches.

frame. (line 13-16, 30-36).

- When a switch observes a new probing message (at "peer" egress), it queries the bitmap and invokes the action at its casualty ingress queues. It first updates the impact it observes with an election. Then, it notices upstream switches of the impact (line 21-29).
- Switch monitors whether it receives a probing message from itself. If a probing message is received with matching triplets, it indicates a deadlock (line 18-19).

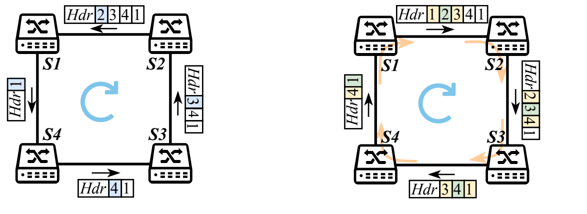
Example. Fig. 5 shows a concrete example of the distributed deadlock detection process. At t_0, switches S1, S2, and S4 become triggers simultaneously, although the deadlock has not yet fully formed. S3 becomes a spreader, and S1 becomes a non-pause at t_2 (possibly due to a decrease in shared buffer occupancy, *etc.*). Elections are performed at different times on different switches. Eventually, switch S3 receives a probing message that originated from itself, indicating a deadlock.

Coordinator. When a switch receives a PAUSE frame with a valid probing message from itself, it becomes a coordinator to act as a controller to lead the subsequent deadlock resolution process. It sends an *announcement packet* upstream, (an *ra_pause* frame with $opcode = 3$ and carries its T_{init}). This will notify other switches in the loop of the deadlock. External packets are not able to enter the loop when deadlock occurs.

IV. Deadlock Resolution

Deadlock resolution is based on a simple observation: Packets trapped in a deadlock loop attempt to exit the loop from their downstream exit switches, following existing routing rules. This is also the expected behavior when no deadlock occurs. If we can enable this behavior even in deadlock scenarios, queue length can be reduced and the deadlock is resolved. The key is to identify available buffer space to resume the flow, allowing stalled packets to continue toward their destination automatically.

Insight. We observed that the headroom is typically configured bigger than its theoretical value, which reveals a new opportunity: by fully leveraging the excess headroom buffer (§II-A) in deadlocked switches, deadlocks can be resolved without buffer reconfiguration. This also brings a significant benefit: we don't need to recalculate new paths for packets during deadlocks, thus saving time and minimizing disruptions to innocent flows on other links.



(a) Accumulate routing info (b) build in-loop routing rules

Fig. 6: Roundabout routing table build-up in the data-plane. The circular arrow represents flow directions.

A. In-deadlock Routing

To resume packet transmission from a deadlocked state, we should first identify the topology of the deadlock loop. Specifically, since an egress queue in the loop may have multiple associated ingress queues, it's crucial to pinpoint the one that is also within the loop. This ensures that the scheduling for deadlock resolution is restricted to the queues within the loop.

Inspired by In-Network Telemetry, we propose a scheme that collects routing information of deadlock paths and notifies the deadlocked switches of their respective ingress and egress ports within the loop. As shown in Fig. 4, variable-length $nodeId$ fields are appended to the end of the *ra_hdr*, with its length recorded in $varLen$ field. There are two consecutive steps for each deadlocked switch to obtain routing information:

Routing information accumulation. As shown in Fig. 6a, once the coordinator (e.g., S1) is elected, it sends a Roundabout packet with $opcode = 4$, reversing the flow direction. Upon receiving this packet, each switch inserts its global ID after the $varLen$ field and relay the packet through its paused causal ingress ports. This process continues until the packet loops back to the coordinator, indicating that the information needed for building the routing table has been collected.

Roundabout routing table buildup. As shown in Fig. 6b, when the coordinator receives the Roundabout packet it originated, it modifies the $opcode$ field to 5, inserts its Global ID again, and forwards the packet along of the deadlock loop with a different priority. Other switches in the loop

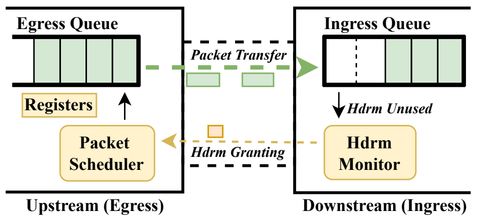


Fig. 7: Buffer collaborates between switches.

identify their in-loop ingress and egress ports by parsing $nodeId_1$ and $nodeId_3$ fields and then querying *SwitchID-Port* table which is pre-configured during network buildup. These switches then removes the $nodeId_1$ field, and forwards the packet downstream through their in-loop egress ports. When the coordinator finally get the packet, it also identifies its in-loop ingress port by parsing the $nodeId_1$ field.

Simplicity. While the packet is variable-length from a global perspective, each switch only need to handle a specific portion of the packet, as the parsing, insertion, and deletion of $nodeId$s are all performed at the fixed position within the Roundabout packet. This conserves Packet Header Vector (PHV) resources and simplifies the implementation.

B. Buffer Collaboration between Switches

We leverage the redundant headroom to resume packet transmission in a lossless manner. To achieve this, the available headroom space of a deadlocked ingress queue need to be granted to its upstream egress queue, allowing packets to be pulled from upstream in a *producer-consumer* pattern.

As shown in Fig. 7, once the Roundabout routing table is built, a *hdrm monitor* at the downstream switch sends a credit packet [41] (a Roundabout packet with $opcode = 6$) to its upstream, carrying *the available headroom size* in its *credit* field. A *credit register* at the upstream is initialized to 0. The *packet scheduler* increments the credit register by the value parsed from credit packets. When a packet is dequeued, the credit register is decremented by the packet size. Packets are sent downstream whenever the credit register value $\geq$ MTU, which will not cause packet drop. Roundabout only requires 1 MTU of extra buffer per port at minimum, which can be easily satisfied under default settings [27]).

For a deadlocked switch, as packets are dequeued and sent downstream, the corresponding ingress queue is drained, increasing its available headroom. This prompts its upstream switch to receive additional credits, which facilitates the transmission of packets in the deadlock.

It is important to note that when a packet is sent downstream, it follows the origin routing table to determine its next hop. The packet escapes the deadlock if its next hop is not a switch in the loop, which leads to a reduction in queue length. All packets will eventually escape the loop in this way. However, this also requires buffer coordination within each switch, as described below.

C. Buffer Collaboration within Switch

Due to the buffer management strategies of dynamic threshold switches, in-loop packet scheduling described in §IV-B can

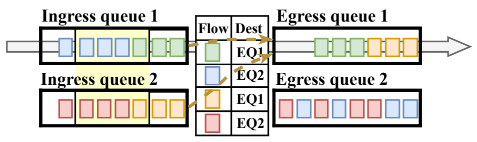


Fig. 8: A case of *false full*. Gray arrow indicates a part of deadlock.

fail. As shown in Fig. 8, ingress and egress queue 1 is in a deadlock loop. However, packets at the head of egress queue 1 come from ingress queue 2. When egress queue 1 dequeues a packet, the headroom occupancy of ingress queue 2 decreases, while ingress queue 1's headroom size remains unchanged, thus cannot make room for further packet scheduling. The root cause is that ingress queue 1 and queue 2 do not share the headroom, so the newly released buffer cannot be utilized by ingress queue 1, which we refer to as *false full*.

Roundabout leverages the property of *virtual statistics queue* (§II-A) to tackle this problem. We adjust the buffer auditing scheme with a slight modification to coordinate buffer usage among queues and ensure that the actual queued packets are unaffected. When an egress queue transmits a packet received from an out-of-loop ingress queue, the packet size is first deducted from the shared buffer and then from the headroom, ensuring that the in-loop ingress queue can actively obtain more space from the shared buffer. This minor adjustment guarantees sufficient buffer space within the loop by efficiently utilizing idle buffers, ensuring continuous buffer collaboration between switches until deadlock is resolved.

Note that this modification is only active during deadlock resolution, which does not impact the behavior of switches when there is no deadlock.

D. In-order Packet Delivery

Roundabout is designed for in-order packet delivery to receivers across various network topologies and routing rules. This relies on a simple ***observation***: a flow's routing path may have one or more segments overlapping with the deadlock. For example, in Fig. 9, the yellow flow has only one segment (S1→S2→S3→S4) while the green flow has two segments (S1→S2 and S3→S4). If packets of a flow on each segment can be forwarded following their original routing rules and exit the loop through their nearest exit switch, packet order can be maintained.

With the methods we proposed above, packets can resume flowing within the loop and exit the deadlock. However, situation can become intricate when a flow enters and leaves the loop multiple times. For instance, in Fig. 9, the green flow has two exit ports on the loop: $Port_1$ and $Port_2$. Packets 3-5 are naturally blocked and cannot enter S3 when the deadlock occurs. If packets from S2 keeps arriving at S5, they will eventually saturate the ingress queue at S5, causing $Port_1$ to be paused as well. Consequently, packet 6 of the green flow cannot be forwarded to $Port_1$ (its nearest exit port). To ensure the scheduling of other flows, Roundabout necessitates rotating packets along the loop following *Roundabout routing table*, so packet 6 will eventually exit the loop through $Port_2$. Packets 3-5 resume transmission only after the deadlock is solved. This

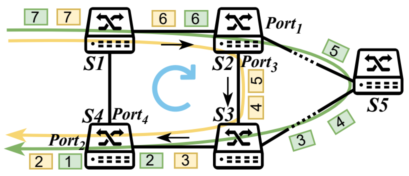


Fig. 9: The green flow has 2 exit ports in the loop: $Port_1$ and $Port_2$.

is acceptable if we only want to break the deadlock, but it can also break packet order.

We design a novel mechanism to keep packets in order. Packets are only permitted to exit the loop through their nearest *exit port*s. If they miss these ports, instead of been forwarded out of loop from other *exit port*s, they are returned to their original positions. In this way, we will not violate the observation and packet order can be maintained.

First, packets that miss their nearest *exit port* are marked by setting the reserved bit in the IP Type of Service (ToS) field to 1 [42, 43], which does not introduce extra bits to inflate the buffer. Packets with ToS=1 will be forwarded to their in-loop downstream switches according to Roundabout routing table and will eventually return to their starting switch.

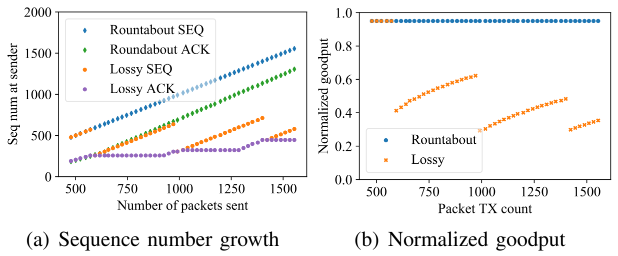
Second, we need to ensure that these packets can stop precisely at the switch where they were located when the deadlock resolution began. We achieve this by introducing *barrier packet*s: After Roundabout routing table is built, instead of transmitting data packets immediately, switch first generates and sends a barrier packet, a Roundabout packet with $opcode = 7$. It is used to isolate the data packets on different switches, and can only be forwarded in the loop following Roundabout routing rules. Barrier packet circulates along the deadlock loop, followed by data packets from the same switch. The packet scheduler module monitors every packet dequeued. Once it identifies a packet as the barrier packet it generated, the dequeue process is halted and the barrier packet is discarded. This mechanism ensures that all packets missing their proper exit ports can return to their originating switches.

When a barrier packet completes its trip in the loop, switch will send a checking message round the loop to confirm whether the deadlock is resolved. The checking message is a Roundabout packet with $opcode = 8$ and the *rand* field set to 1. At each switch in the loop, if its barrier packet has returned and queue length is below the X_{ON} threshold, the *rand* field is set to 0. If the coordinator receives its checking message with *rand*=0, it indicates that the deadlock has been resolved and packets can re-enter the loop. Otherwise, the coordinator can initiate another round of resolution until the deadlock is solved, or fall back to other solutions.

V. Testbed Experiments

A. Implementation

We implemented a Roundabout switch prototype, as well as the senders and receivers, on commodity x86 servers with DPDK 19.08.2 [44]. Both the switches and sender/receiver nodes are equipped with 10Gbps NICs. The testbed is built with a topology and routing scheme identical to Fig. 1.



(a) Sequence number growth (b) Normalized goodput

Fig. 10: Roundabout has minimum bandwidth waste.

The switch's data plane consists of three modules: *Receiving* module gets packets from multiple input ports for serial processing and manages port pause states based on Roundabout packets. *Forwarding* module implements the main function of Roundabout, such as ra_frame generation, local election, deadlock routing table buildup, ToS marking, admission control and queue management. *Transmitting* module dequeues and sends packets downstream through their corresponding egress ports in a round-robin (RR) fashion.

B. Testbed Evaluation

Roundabout solves deadlocks with minimum impact on end-to-end performance. Since deadlock formation is determined by queue length rather than flow size, we let senders transmit long flows that pass through three switches in clockwise order before reaching the receiver. Receivers sends per-packet ACKs back to the senders.

We compare Roundabout with selective packet drop [20], and a sampling of packet sequence number is shown in Fig. 10a. Compared to schemes that cause sequence number gaps like packet drop, Roundabout is able to maintain in-order packet delivery to avoid the large number of unnecessary retransmissions caused by RoCE's go-back-N transmission scheme. Fig. 10b shows the normalized effective throughput (goodput). Lossy deadlock resolution scheme results in a significant decline in effective throughput, whereas Roundabout maintains consistently high end-to-end performance.

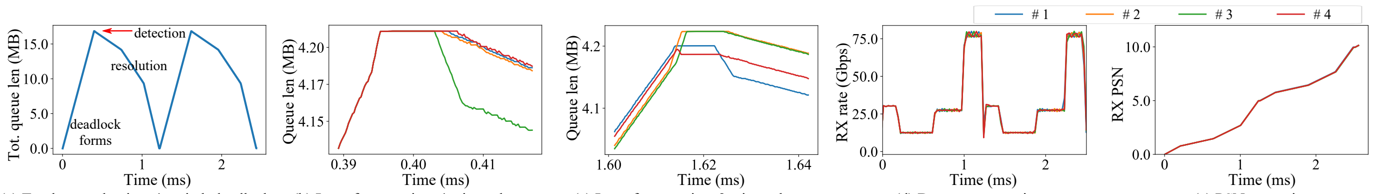
VI. Simulation Evaluation

A. Simulation Setup

We evaluate our design with network simulator NS3 [45].

We first use full mesh topology [46] from §VI-B to §VI-D to demonstrate the micro behavior of Roundabout, with 16 shared buffer switches connected to each other. Each switch has 4 servers connected. All links have a capacity of 100Gbps and a propagation delay of 1us [2]. We disable flow control on the server such that all traffic can be sent at 100Gbps[2]. Unless explicitly specified, we use the following parameters as default: Each switch has an SRAM buffer of 32MB, a redundant headroom of 1 BDP, an in-loop hop count (the number of hops from a packet entering the loop to it reaches its exit port) of 3, a maximum shared portion (the maximum percentage of shared buffer that can be allocated to a single queue)

[2]We use full mesh topology and disable flow control to create more intuitive and varied deadlocks (different deadlock loop length, trigger locations, number of hops from exit ports, *etc.*), which is uncommon in practice.



(a) Total queue len in a 4-switch deadlock. (b) Len of queues in a 4-trigger loop. (c) Len of queues in a 2-trigger loop. (d) Data rate at receivers. (e) PSN at receivers.

Fig. 11: The behavior of Roundabout.

of 1/8. We utilize AI training workload measured in a real-world testbed cluster. The neural network has multiple fully connected layers and employs ring all-reduce to synchronize gradients. Compared to web search [47] or incast [48] traffic, this workload is more bursty and exhibits a higher degree of concurrency [49], thus is more likely to cause deadlocks.

We also evaluate Roundabout under 2D-Torus [50] topologies in §VI-E. The 2D-Torus topology comprises 16 switches organized in a 4×4 grid, with each switch connected to 4 servers. The switch configurations are identical to those in the Full-Mesh topology. We used Ali-Storage workload [2], set the average link load to 80% to simulate heavy traffic conditions, and use DCQCN [51] as the congestion control algorithm.

B. Demonstration of Roundabout

We first present the behavior of Roundabout. We configure the routing table for a loop of 4 switches. Flows enter the loop from different switches, circulate in the same direction, and leave the loop after three hops. Fig. 11a shows the sum of the deadlock queue lengths on in-loop switches during two complete deadlock formation-resolution processes. The queues grows first, then triggers deadlock due to the frequent “on-off” port behavior [52]. Roundabout is able to detect deadlock within microseconds and then start deadlock resolution.

We then dive into the micro-behavior of deadlock detection. Fig. 11b shows a scenario where four switches become triggers nearly simultaneously, which takes only one hop's time to form a deadlock. Roundabout uses the following 3 hop's time for deadlock detection, and another 4 hop's time to synchronize the results with other switches. Fig. 11c shows another scenario: two switches become triggers first, then pause the other two switches as PFC pause frames propagate. In this case, deadlock formation and detection finish almost simultaneously. The “plateau region” in the curve is much shorter, during which the deadlock detection results is synchronized with other switches in the loop. Both cases in Fig. 11b and Fig. 11c detect deadlock within 10 microseconds, demonstrating the effectiveness of our scheme. Notably, for a loop of length N, achieving global consensus and synchronizing with all nodes requires 2N hops. The performance of Roundabout follows a similar pattern.

Once deadlock is confirmed, Roundabout initiates deadlock resolution. Packets are efficiently scheduled to exit the loop in a lossless manner. We observe that one of the queues experiences a rapid decrease in length and then returns to normal. This is because buffer negotiation between switches is triggered hop-by-hop. When the first switch has already sent a significant number of packets downstream, its upstream switch may still be waiting for buffer negotiation to be triggered. Thus, the first switch will temporarily not receive any packets until buffer collaboration starts across all switches, resulting in a faster draining rate of the queue.

Roundabout efficiently supports end-to-end packet transmission. Fig. 11d reveals that since detection can be done quickly, the time packets actually paused in the loop can be negligible. As packets of a certain flow approach their exit port, packets of other flows are gradually scheduled out of loop midway. This increases the proportion of the given flow in the overall traffic, resulting in a faster sending rate (even faster than before the deadlock occurred). Fig. 11e is actually a scatter plot of the sequence number of packets that arrives at the receiver over time. It shows the order-preserving nature of Roundabout and further corroborates the results from the testbed experiments.

C. Performance.

We evaluate the performance of Roundabout across various configurations, including different loop lengths, headroom sizes, in-loop hop counts, and maximum shared portions. We vary one parameter once while keeping the others as default.

Fig. 12 reveal some important properties of Roundabout: a) As the loop length increases, Roundabout spends more time for detection, since it needs to circulate control messages over the loop. However, loop length itself does not significantly impact the speed of deadlock resolution. b) Deadlocks are triggered more quickly as incast becomes more severe with an increasing average in-loop hop count. This is due to the introduction of additional flows on the same link, which dilutes the target flows and reduces their sending rate. c) The size of redundant headroom affects the initial speed of deadlock resolution. However, as packets exit the loop, the available headroom buffer gradually grows to its predefined limit and is no longer a bottleneck, which does not significantly influence the overall resolution time. d) The maximum available queue length (related to the switch's buffer size and memory management policy) is a double-edged sword: a larger queue length can accommodate more packets before triggering a deadlock, but it also prolongs the overall resolution process.

D. Comparison with Other Solutions.

We compare Roundabout with four latest deadlock detection and resolution methods. Loop Breaker (*LB*) and Deadlock Breaker (*DB*) [19] are control-plane approaches that asserts deadlocks by probing port states. (*i*) *LB* periodically sends probe packets with unique identifiers from deadlock-suspected

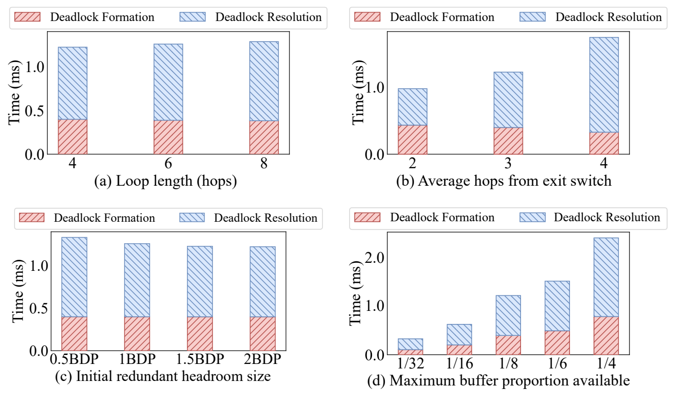


Fig. 12: Roundabout's performance under different metrics.

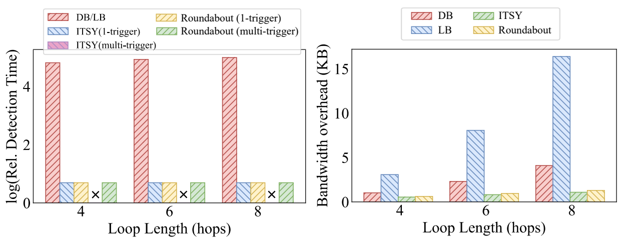
ports along the loop. Each node choose and record the smaller identifier and pass it on, until the identifier returns to its starting port. It resolves deadlocks by dynamically rerouting deadlocked packets to other switches. (*ii*) *DB* detects deadlocks similar to *LB* but carries full-path information in probing packets. Upon detecting a deadlock, the control plane dynamically adjusts the buffer to resume packet flow. (*iii*) ITSY [20] is a dataplane scheme. It first identifies an initial trigger, and then sends a checking message reversing the loop until it returns to the trigger. It proposes packet dropping (lossy) and active buffer adjustment (lossless) to resolve deadlocks.

Requirements	LB	DB	ITSY (lossy)	ITSY (lossless)	Roundabout
Quick Detection	×	×	✓	✓	✓
Adapt to Multiple Triggers	✓	✓	×	×	✓
Small Detection Overhead	×	×	✓	✓	✓
Keep Packets in Order	×	–	×	–	✓
No side-effects on Other Queues	×	×	✓	×	✓
Compatible with RoCEv2	×	×	×	×	✓

TABLE II: Roundabout satisfies all the requirements.

Overall Effect. As shown in Table II, Roundabout outperforms other solutions across various metrics. As for deadlock detection schemes, *LB* and *DB* exhibit inherent performance issues and encounter significant detecting overhead. Although ITSY can detect deadlock quickly, it faces challenge when the loop status is complex and has multiple triggers. As for deadlock resolution schemes, *LB* leverages adaptive routing which may affect other traffic; *DB* and ITSY (lossless) dynamically configure the buffer but may not succeed when shared buffer space is constrained. None of the compared schemes can guarantee in-order packet delivery. Roundabout is the only solution that can satisfy all the listed metrics in table II, and is therefore ideally suited for RoCEv2.

Detecting Speed. As shown in Fig. 13a, Roundabout significantly outperforms *LB* and *DB*, which suffer from latency between the control plane and data plane. Additionally, *LB* and *DB* exhibit more variable detection latency, as deadlocks may occur at any time between two periodical detections. Although Roundabout and ITSY have similar detection speeds, Roundabout is able to effectively detects and resolves deadlocks when multiple switches trigger the same deadlock.



(a) Relative detection time (b) BW overhead of one detection

Fig. 13: Roundabout compared with other schemes.

Bandwidth Overhead. Fig. 13b compares the additional bandwidth consumption during a complete detecting process. Roundabout and ITSY, triggered by PAUSE events, consume minimal bandwidth. Roundabout also reuses header fields to further optimize bandwidth utilization. However, since Roundabout builds in-loop routing table with variable-length $nodeId$ field, this results in a relatively higher (yet still small) bandwidth cost compared to ITSY. Both *LB* and *DB* have to periodically send detection packets at high frequency to ensure timely detection, which consumes more bandwidth.

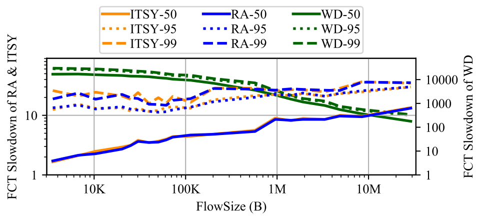


Fig. 14: FCT slowdown with different schemes. Note that WD has a different y-scale.

E. Evaluating Roundabout in Large-scale Networks.

We evaluate Roundabout (RA) with 2D-Torus topology and Ali-Storage workload [2] with 80% link load, as described in §VI-A. To evaluate the impact on flows affected before and after the deadlock, we initiate flows within 0.1 seconds following a Poisson distribution. We compare RA with (*i*) ITSY with packet dropping, and (*ii*) PFC watchdog (WD) deployed in the industry, which monitors for long-duration PFC pauses and take measures such as packet drops or port resets to maintain network stability. The watchdog threshold is set to 200 ms [53].

Fig. 14 shows the flow completion time (FCT) slowdown, flow's actual FCT normalized by the base FCT when the network has no other traffic, at the 50th, 95th, and 99th percentiles under our settings. In the WD scenario, PFC pause spreading caused by deadlock [20] affects the network. Most flows are paused over 100 ms, since the watchdog threshold is significantly higher than their theoretical completion time. As a result, the FCT slowdowns of these flows increase significantly, especially for short flows whose ideal FCT is only tens of microseconds. Roundabout and ITSY detect deadlocks quickly, allowing the network to promptly return to normal and further avoiding the widespread of PFC pause frames. However, ITSY's detection is not always successful. In the experiments where it successfully detects deadlocks, it results in an average of 16 flows experiencing out-of-order transmission and 7,952 packets being received out of order. In contrast, Roundabout effectively detects deadlocks and does not cause packet reordering, thereby minimizing the impact on both senders and receivers.

VII. Discussion

A. Concurrent Deadlocks

Multiple deadlock loops may overlap on links or nodes, resulting in concurrent deadlocks. Roundabout selects and subsequently breaks one of the deadlocks, thus automatically resolving the others. (*i*) If these loops have different coordinators, their probing messages will competing for T_{curr} registers at overlapped switches, and only the dominant message can survive. Therefore, only the "optimal" coordinator can succeed when synchronizing detection results. (*ii*) When multiple loops share the same coordinator, probing message first propagates along overlapping path and is then sent to distinct paths at branching switch in a multicast manner. These probing packets carrying the same messages will reach the convergence switch in sequence. Only the first arrival will pass through and return to the coordinator, thereby selecting a unique loop.

B. Hardware Feasibility

The objective of Roundabout is to operate as a building block to improve the deadlock-handling capabilities of switches, which can be integrated in switch ASICs.

Meanwhile, as switches become increasingly programmable [54, 55], Roundabout also shows promising potential to achieve more flexible deployment. Most features required by Roundabout have been natively supported by commodity programmable hardware [55], including registers for casualty recording, customized packet header parsing and deparsing, user-defined packet processing, programmable packet generation, and dataplane advanced flow control to pause/resume target queue [56]. PFC frames can be passed to the switch's programmable module by manipulating port configurations. Although the length of *ra_hdr* is variable from a global perspective, packet parsing and deparsing occur at fixed position for a single switch (§IV-A). Therefore, the packet header vector (PHV) overhead remains constant. The buffer management scheme of Roundabout involves updating ingress queue statistics by modifying counters, which is easy to achieve line rate in ASICs. Some programmable switch designs, such as Trio [54], also provide a read-modify-write engine to increment or decrement packet/byte counters, which is worth exploring and we will leave it as future work.

VIII. Limitations

To evaluate Roundabout under deadlocks, we used configurations that trigger deadlocks more frequently (e.g., mesh topology, disabled congestion control, and more bursty workloads). In practice, deadlocks may occur less frequently. For example, in Clos networks like fat-tree [2, 57], deadlocks can be rare. Packets are typically routed in a loop-free manner, such as through up/down routing paradigm [58], where a spanning tree is constructed for the network, and packets are forwarded up toward the root direction before being sent down to the destination. This prevents routing loops and is therefore deadlock-free [59]. However, factors such as link/node failures or port flaps [16, 21, 60] can cause packets to deviate from their intended paths, potentially introducing routing loops and leading to deadlock. Some schemes are proposed to avoid such loops, such as [16, 21] discussed in IX.

IX. Related Work

PFC deadlock has been a popular research topic for many years. Many approaches have been proposed that can be classified into two categories: proactively avoid deadlock, and reactively detect/resolve deadlock. We have discussed deadlock detection/resolution schemes in detail in §II-B.

For deadlock avoidance, many approaches rely on routing or topology restrictions [13–16]. However, this can lead to reduced throughput, wasted link bandwidth [17], and may not be suitable for various topologies and routing algorithms [14]. Moreover, such approaches tightly couple routing rules with the network topology, making them inflexible for deployment [15]. Similarly, partitioning priorities into lossy and lossless and scheduling flows across different priorities [21] not only makes the network lossy, but also wastes valuable bandwidth and priorities. Some other approaches adjust buffer configurations of switches based on their locations to avoid deadlocks [17], but this requires a relatively stable traffic pattern, and can not eliminate the probability of deadlock. Eliminating hold-and-wait condition to avoid deadlocks seems promising [18], but it can impact the sending rate and requires fine-grained controlling, which can be challenging in practice. Adjusting flow control algorithms may reduce the probability of deadlocks [2, 9, 51, 61] but are not 100% reliable.

X. Conclusion

We present Roundabout, a deadlock detection and resolution scheme for PFC-enabled data center switches. We analyze switch states within deadlock loops and propose an election-based approach for efficient and robust deadlock detection. We present a novel buffer collaboration scheme between and within switches to resume packet flowing without requiring buffer reconfiguration, which only leverages engineering redundancy in the headroom buffer. Additionally, we analyze the cause of packet reordering during deadlock resolution and design an isolation-scheduling mechanism to maintain packet order. Experimental results demonstrate that Roundabout can effectively detect deadlocks in complex scenarios and efficiently resolve deadlocks with minimal side effects.

Acknowledgments

We thank our shepherd Soudeh Ghorbani and the anonymous reviewers for their insightful feedback. This work is supported in part by National Key R&D Program of China (Grant No. 2022YFB2901300), and in part by Major Key Project of PCL (Grant No. PCL2023AS1-5, PCL2023AS1-3). Heng Pan is the corresponding author.

References

[1] C. Guo, H. Wu, Z. Deng, G. Soni, J. Ye, J. Padhye, and M. Lipshteyn, “Rdma over commodity ethernet at scale,” in *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016, pp. 202–215.

[2] Y. Li, R. Miao, H. H. Liu, Y. Zhuang, F. Feng, L. Tang, Z. Cao, M. Zhang, F. Kelly, M. Alizadeh *et al.*, “Hpcc: High precision congestion control,” in *Proceedings of the ACM Special Interest Group on Data Communication*, 2019, pp. 44–58.

[3] Y. Gao, Q. Li, L. Tang, Y. Xi, P. Zhang, W. Peng, B. Li, Y. Wu, S. Liu, L. Yan *et al.*, “When cloud storage meets rdma,” in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, 2021, pp. 519–533.

[4] W. Bai, S. S. Abdeen, A. Agrawal, K. K. Attre, P. Bahl *et al.*, “Empowering azure storage with rdma,” in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023, pp. 49–67.

[5] A. Mahgoub, L. Wang, K. Shankar, Y. Zhang, H. Tian, S. Mitra, Y. Peng, H. Wang, A. Klimovic, H. Yang *et al.*, “Sonic: Application-aware data passing for chained serverless applications,” in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, 2021, pp. 285–301.

[6] Y. Chen, X. Wei, J. Shi, R. Chen, and H. Chen, “Fast and general distributed transactions using rdma and htm,” in *Proceedings of the Eleventh European Conference on Computer Systems*, 2016, pp. 1–17.

[7] Y. Jiang, Y. Zhu, C. Lan, B. Yi, Y. Cui, and C. Guo, “A unified architecture for accelerating distributed dnn training in heterogeneous gpu/cpu clusters,” in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 463–479.

[8] IEEE, “Priority-based flow control,” https://1.ieee802.org/dcb/802-1qbb/, 2010.

[9] R. Mittal, V. T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats, “Timely: Rtt-based congestion control for the datacenter,” *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 537–550, 2015.

[10] A. Gangidi, R. Miao, S. Zheng, S. J. Bondu, G. Goes, H. Morsy, R. Puri, M. Riftadi, A. J. Shetty, J. Yang *et al.*, “Rdma over ethernet for distributed training at meta scale,” in *Proceedings of the ACM SIGCOMM 2024 Conference*, 2024, pp. 57–70.

[11] Y. Jiang, H. Gu, Y. Lu, and X. Yu, “2d-hra: Two-dimensional hierarchical ring-based all-reduce algorithm in large-scale distributed machine learning,” *IEEE Access*, vol. 8, pp. 183 488–183 494, 2020.

[12] Z. Jiang, H. Lin, Y. Zhong, Q. Huang, Y. Chen, Z. Zhang, Y. Peng, X. Li, C. Xie, S. Nong, Y. Jia, S. He, H. Chen, Z. Bai, Q. Hou, S. Yan, D. Zhou, Y. Sheng, Z. Jiang, H. Xu, H. Wei, Z. Zhang, P. Nie, L. Zou, S. Zhao, L. Xiang, Z. Liu, Z. Li, X. Jia, J. Ye, X. Jin, and X. Liu, “MegaScale: Scaling large language model training to more than 10,000 GPUs,” in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. Santa Clara, CA: USENIX Association, Apr. 2024, pp. 745–760. [Online]. Available: https://www.usenix.org/conference/nsdi24/presentation/jiang-ziheng

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

[14] B. Stephens, A. L. Cox, A. Singla, J. Carter, C. Dixon, and W. Felter, “Practical dcb for improved data center networks,” in *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, 2014, pp. 1824–1832.

[15] B. Stephens and A. L. Cox, “Deadlock-free local fast failover for arbitrary data center networks,” in *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*, 2016, pp. 1–9.

[16] S. Zhao, Q. Zhang, P. Cao, X. Zhang, X. Wang, and C. Zhou, “Flattened clos: Designing high-performance deadlock-free expander data center networks using graph contraction,” in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023, pp. 663–683.

[17] S. Hu, Y. Zhu, P. Cheng, C. Guo, K. Tan, J. Padhye, and K. Chen, “Deadlocks in datacenter networks: Why do they form, and how to avoid them,” in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, 2016, pp. 92–98.

[18] K. Qian, W. Cheng, T. Zhang, and F. Ren, “Gentle flow control: avoiding deadlock in lossless networks,” in *Proceedings of the ACM Special Interest Group on Data Communication*, 2019, pp. 75–89.

[19] A. Shpiner and othersov, “Unlocking credit loop deadlocks,” in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, 2016, pp. 85–91.

[20] X. C. Wu and T. E. Ng, “Detecting and resolving pfc deadlocks with itsy entirely in the data plane,” in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*, 2022, pp. 1928–1937.

[21] S. Hu, Y. Zhu, P. Cheng, C. Guo, K. Tan, J. Padhye, and K. Chen, “Tagger: Practical pfc deadlock prevention in data center networks,” in *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*, 2017, pp. 451–463.

[22] P. Lopez, J.-M. Martínez, and J. Duato, “A very efficient distributed deadlock detection mechanism for wormhole networks,” in *Proceedings 1998 Fourth International Symposium on High-Performance Computer Architecture*. IEEE, 1998, pp. 57–66.

[23] Z. Wang, L. Luo, Q. Ning, C. Zeng, W. Li, X. Wan, P. Xie, T. Feng, K. Cheng, X. Geng *et al.*, “Srnic: A scalable architecture for rdma nics,” in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023, pp. 1–14.

[24] Cisco White Papers, “Priority flow control: build reliable layer-2 infrastructure,” 2010.

[25] “Traffic management user guide (qfx series switches and ex4600 switches),” https://www.juniper.net/documentation/us/en/software/junos/traffic-mgmt-qfx/traffic-mgmt-qfx.pdf, 2023.

[26] “Bcm88480 traffic management architecture,” https://docs.broadcom.com/doc/88480-DG1-PUB, 2021.

[27] L. Lv, “Adaptive pfc headroom,” https://www.ieee802.org/1/files/public/docs2021/new-lv-adaptive-pfc-headroom-0121-v01.pdf, 2021.

[28] A. K. Choudhury and E. L. Hahne, “Dynamic queue length thresholds for shared-memory packet switches,” *IEEE/ACM Transactions On Networking*, vol. 6, no. 2, pp. 130–140, 1998.

[29] V. Addanki, W. Bai, S. Schmid, and M. Apostolaki, “Reverie: Low pass filter-based switch buffer sharing for datacenters with RDMA and tcp traffic,” in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. Santa Clara, CA: USENIX Association, Apr. 2024, pp. 651–668. [Online]. Available: https://www.usenix.org/conference/nsdi24/presentation/addanki-reverie

[30] NVIDIA, “UNDERSTANDING THE ALPHA PARAMETER IN THE BUFFER CONFIGURATION OF MELLANOX SPECTRUM SWITCHES,” https://enterprise-support.nvidia.com/s/article/understanding-the-alpha-parameter-in-the-buffer-configuration-of-mellanox-spectrum-switches, 2022.

[31] NVIDIA Onyx User Manual v3.10.4302 (LTS), “Shared Buffers,” https://docs.nvidia.com/networking/display/onyxv3104302/shared+buffers, 2023.

[32] BROADCOM, “Bcm88480 traffic management architecture,” https://docs.broadcom.com/doc/88480-DG1-PUB.

[33] Intel, “P4$_{16}$ intel$^{®}$ tofino^{TM} switch,” https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series.html.

[34] X. Wu *et al.*, “Netpilot: Automating datacenter network failure mitigation,” in *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, 2012, pp. 419–430.

[35] K. Anjan and T. M. Pinkston, “An efficient, fully adaptive deadlock recovery scheme: Disha,” in *Proceedings of the 22nd annual international symposium on Computer architecture*, 1995, pp. 201–210.

[36] S. K. R. Kakarla *et al.*, “Finding network misconfigurations by automatic template inference,” in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020, pp. 999–1013.

[37] J. Lee, I. Hwang, S. Shah, and M. Cho, “Flexreduce: Flexible all-reduce for distributed deep learning on asymmetric network topology,” in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[38] NVIDIA, “Out-of-order (ooo) data placement,” https://docs.nvidia.com/networking/display/MLNXOFEDv590560107/Out-of-Order+(OOO)+Data+Placement.

[39] NVIDIA, “Connectx-6 dx ethernet smartnic,” https://nvdam.widen.net/s/qpszhmhpzt/networking-overal-dpu-datasheet-connectx-6-dx-smartnic-1991450.

[40] Q. Li *et al.*, “Flor: An open high performance rdma framework over heterogeneous rnics,” in *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, 2023, pp. 931–948.

[41] S. Hu, W. Bai, G. Zeng, Z. Wang, B. Qiao, K. Chen, K. Tan, and Y. Wang, “Aeolus: A building block for proactive transport in datacenters,” in *Proceedings of the Annual conference of the ACM*

Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication, 2020, pp. 422–434.

[42] Y. Li, I.-J. Liu, Y. Yuan, D. Chen, A. Schwing, and J. Huang, "Accelerating distributed reinforcement learning with in-switch computing," in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019, pp. 279–291.

[43] P. Zhang, H. Pan, Z. Li, P. Cui, R. Jia, P. He, Z. Zhang, G. Tyson, and G. Xie, "Netsha: In-network acceleration of lsh-based distributed search," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 9, pp. 2213–2229, 2021.

[44] Linux Foundation, "Data plane development kit (dpdk)," http://www.dpdk.org, 2019.

[45] "NS3 network simulator," https://www.nsnam.org, 2023.

[46] Z. Li and P. Mohapaira, "The impact of topology on overlay routing service," in *IEEE INFOCOM 2004*, vol. 1. IEEE, 2004.

[47] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," in *Proceedings of the ACM SIGCOMM 2010 Conference*, 2010, pp. 63–74.

[48] V. Addanki, O. Michel, and S. Schmid, "Powertcp: Pushing the performance limits of datacenter networks," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022, pp. 51–70.

[49] Cisco, "Evolve your ai/ml network with cisco silicon one," https://www.cisco.com/c/en/us/solutions/collateral/silicon-one/evolve-ai-ml-network-silicon-one.html, 2023.

[50] H. Mikami, H. Suganuma, Y. Tanaka, Y. Kageyama *et al.*, "Massively distributed sgd: Imagenet/resnet-50 training in a flash," *arXiv preprint arXiv:1811.05233*, 2018.

[51] Y. Zhu *et al.*, "Congestion control for large-scale rdma deployments," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 523–536, 2015.

[52] Y. Zhang, Y. Liu, Q. Meng, and F. Ren, "Congestion detection in lossless networks," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, 2021, pp. 370–383.

[53] A. Gangidi, R. Miao, S. Zheng, S. J. Bondu, G. Goes, H. Morsy, R. Puri, M. Riftadi, A. J. Shetty, J. Yang, S. Zhang, M. J. Fernandez, S. Gandham, and H. Zeng, "Rdma over ethernet for distributed training at meta scale," in *Proceedings of the ACM SIGCOMM 2024 Conference*, ser. ACM SIGCOMM '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 57–70. [Online]. Available: https://doi.org/10.1145/3651890.3672233

[54] M. Yang, A. Baban, V. Kugel, J. Libby, S. Mackie, S. S. R. Kananda, C.-H. Wu, and M. Ghobadi, "Using trio: juniper networks' programmable chipset-for emerging in-network applications," in *Proceedings of the ACM SIGCOMM 2022 Conference*, 2022, pp. 633–648.

[55] "Intel tofino 2." https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-2-series.html, 2023.

[56] J. Lee, "Advanced Congestion & Flow Control with Programmable Switches," https://opennetworking.org/wp-content/uploads/2020/04/JK-Lee-Slide-Deck.pdf, 2020.

[57] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM computer communication review*, vol. 38, no. 4, pp. 63–74, 2008.

[58] S. Y. Qiu, P. D. McDaniel, and F. Monrose, "Toward valley-free interdomain routing," in *2007 IEEE International Conference on Communications*, 2007, pp. 2009–2016.

[59] X. Yuan, W. Nienaber, Z. Duan, and R. Melhem, "Oblivious routing in fat-tree based system area networks with uncertain traffic demands," *IEEE/ACM Transactions on Networking*, vol. 17, no. 5, pp. 1439–1452, 2009.

[60] V. Liu, D. Halperin, A. Krishnamurthy, and T. Anderson, "F10: A Fault-Tolerant engineered network," in *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. Lombard, IL: USENIX Association, Apr. 2013, pp. 399–412. [Online]. Available: https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/liu_vincent

[61] W. Cheng, K. Qian, W. Jiang, T. Zhang, and F. Ren, "Re-architecting congestion management in lossless ethernet," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020, pp. 19–36.