Gentle Flow Control: Avoiding Deadlock in Lossless Networks

Kun Qian, Wenxue Cheng, Tong Zhang, Fengyuan Ren
Tsinghua University
Beijing National Research Center for Information Science and Technology (BNRist)

ABSTRACT

Many applications in distributed systems rely on underlying lossless networks to achieve required performance. Existing lossless network solutions propose different hop-by-hop flow controls to guarantee zero packet loss. However, another crucial problem called network deadlock occurs concomitantly. Once the system traps in a deadlock, a large part of network would be disabled. Existing deadlock avoidance solutions focus all their attentions on breaking the cyclic buffer dependency to eliminate *circular wait* (one necessary condition of deadlock). These solutions, however, impose many restrictions on network configurations and side-effects on performance.

In this work, we explore a brand-new perspective to solve network deadlock: avoiding *hold and wait* situation (another necessary condition). Experimental observations tell that frequent pause on upstream ports driven by existing flow control schemes is the root cause of *hold and wait*. We propose Gentle Flow Control (GFC) to manipulate the port rate at a fine granularity, so all ports can keep packets flowing even cyclic buffer dependency exists, and prove GFC can eliminate deadlock theoretically. We also present how to implement GFC in mainstream lossless networks (Converged Enhanced Ethernet and InfiniBand) with moderate modifications. Furthermore, testbed experiments and packet-level simulations validate GFC can efficiently avoid deadlock and introduce less than 0.5% of bandwidth occupation.

CCS CONCEPTS

• **Networks → Link-layer protocols**; **Data path algorithms**;

KEYWORDS

Lossless Networks, Deadlock Prevention, Flow Control

ACM Reference Format:
Kun Qian, Wenxue Cheng, Tong Zhang, Fengyuan Ren. 2019. Gentle Flow Control: Avoiding Deadlock in Lossless Networks. In *SIGCOMM '19: 2019 Conference of the ACM Special Interest Group on Data Communication, August 19–23, 2019, Beijing, China.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3341302.3342065

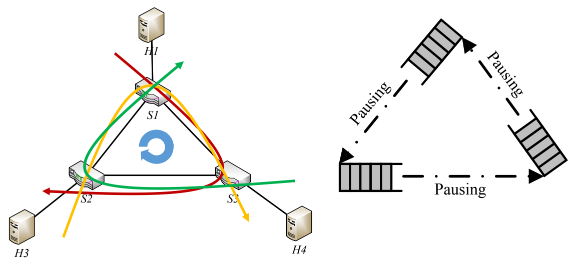
1 INTRODUCTION

Lossless network is a crucial infrastructure in many application scenarios. In data center networks, packet loss causes great damage to application performance and revenue loss [23, 37, 59, 60]. In HPC system, InterProcess Communication (IPC) requires extremely low communication latency. Packet retransmissions caused by loss would vastly increase the transmission delay and further lead to missing communication deadlines. In block-based storage, lossless delivery of network packets is a critical requirement for providing satisfied quality of service [9].

In order to provide a lossless network switching fabric, Converged Enhanced Ethernet (CEE) employs Priority Flow Control (PFC) [27], and InfiniBand develops Credit Based Flow Control (CBFC) [4]. These flow controls guarantee zero loss by ceasing the upstream port before buffer overflows and resuming packet sending after queue decreasing. However, a crucial phenomenon called network deadlock appears concomitantly. Network deadlock refers to a standstill situation: a Cyclic Buffer Dependency (CBD) exists in the network, while all switches in this cycle pause the upstream switch port and wait for the downstream switch port to resume packet transmission at the same time [24]. Figure 1 illustrates a simple example of deadlock. When it happens, all ports in the CBD cease forever. Furthermore, these ports will pressure congestion back to sources and then cease all ports along the path. Finally, the whole (or part of) network would be disabled [24]. Deadlock is a silent killer. Once it occurs, the network cannot resume to normal state autonomously.

The issue of network deadlock has been discovered in ages [14]. In the past, the scale of network is comparatively small so operators can manipulate network configurations well to avoid deadlock. Thus it does not raise considerable attentions. However, with the fast growth of data centers and clusters, the scale of networks and traffic increase drastically. It becomes impractical to avoid deadlock manually. Many large data center operators have confirmed the occurrence of deadlock in practical scenarios [25]. Since network deadlock would bring unbearable side-effects (e.g., entire network paralysis), both cloud providers [22, 24, 25] and device manufacturers [19, 52] devote great efforts to solving it.

The existing solutions to network deadlock problem fall into two main categories: recovery and avoidance. Recovery schemes first detect deadlock and then break it [2, 3, 36, 38, 52]. However, it is difficult to detect deadlock in distributed networks, and the corresponding breaking approaches (e.g., resetting devices, dropping packets and heuristic rerouting) are blunt and rigid [24]. Moreover, they do not eradicate the root cause of deadlock. On the other hand, avoidance schemes intend to break the necessary conditions of deadlock to eliminate it. Existing avoidance solutions mainly focus on breaking the *circular wait* condition [6, 7, 13–18, 20, 21, 35, 46, 50, 54–56, 58]. However, totally eliminating CBD is intractable under intricate traffic in large-scale networks [24]. Such solutions impose restrictions on routing configurations or flows' priorities. As a result, the network capabilities of multi-path and performance isolation are razed dramatically.



(a) Flow directions (b) CBD

Figure 1: Simple example of deadlock.

In this work, we explore a brand-new solution to break another necessary condition, that is *hold and wait*. The experimental observations enlighten us that employing existing flow control schemes would inevitably cause *hold and wait*. When buffer is almost full, the flow control mechanisms directly pause the upstream port, then packets in the upstream port fall into the state of "holding" in current buffer and "waiting" for available buffer space in the downstream port. Especially in congestion scenarios, these flow controls will pause upstream ports frequently, which makes the network prone to *hold and wait*.

In light of above understanding, we propose Gentle Flow Control (GFC) to avoid the *hold and wait* condition. First, we propose a conceptual design to manipulate input rate at a fine granularity: as the queue length increases, the flow control decreases the upstream rate accordingly. The port will eventually enter into a status where the input rate matches the draining rate and the queue length keeps steady. Therefore, all flows can keep passing through the network continuously even CBD exists, and no deadlock appears. Then, considering the actual deployment constraints, the practical GFC is proposed, which introduces negligible side-effect on available bandwidth. Moreover, we discuss how to implement GFC in mainstream lossless networks with moderate modifications. We use testbed experiments as well as large-scale simulations to verify the performance of GFC. The results confirm that GFC can avoid deadlock efficiently and eliminate accompanying side-effects as well.

The contributions of this paper are as follows:

(1) Understanding and solving network deadlock through avoiding *hold and wait*.
(2) Designing GFC, and demonstrating its effectiveness in avoiding *hold and wait* theoretically.
(3) Presenting how to implement GFC in mainstream lossless switching fabrics (e.g., CEE and InfiniBand) with moderate modifications.

The research in this paper poses no ethical issues. The rest of the paper is organized as follows: Section 2 introduces all necessary conditions for network deadlock and mainstream lossless flow control mechanisms in detail. In Section 3, design decisions are discussed. Both conceptual and practical GFC are elaborated in Section 4. Implementation details of GFC are presented in Section 5. Section 6 shows the evaluation results. Section 7 gives some discussions about GFC. Section 8 introduces related work about solving deadlocks and finally the paper is concluded in Section 9.

2 BACKGROUND

2.1 Necessary conditions for network deadlock

Deadlock is a fundamental issue in multi-task systems, and there have been numerous investigations focusing on it. In [53], four necessary conditions for deadlock occurrence are summarized: (1) mutual exclusion, (2) no preemption, (3) hold and wait and (4) circular wait. We elaborate how these necessary conditions map to the network deadlock issue.

Mutual exclusion: Mutual exclusion means the resource is non-sharable. In lossless networks, this non-sharable resource is buffer. Once a packet is received, it occupies a part of buffer space. If other packets require the occupied buffer, they need to wait until it is released, namely, stored packets are forwarded to the next hop.

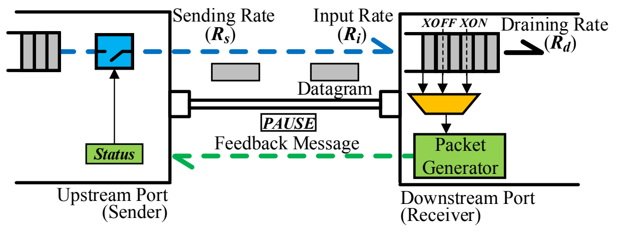
No preemption: Preemption means that the switch drops stored packets for receiving new ones. In lossless network, no packet has the right to expropriate buffer occupied by other packets. Therefore, this condition always holds.

Hold and wait: Hold and wait means that packets are occupying the buffer in current switch and waiting for the available buffer space in the downstream switch. For example, in Figure 1(b), when the downstream input buffer is full, packets in the upstream switch must stay in the queue and wait. It is a common situation in existing lossless networks when the downstream switch is congested.

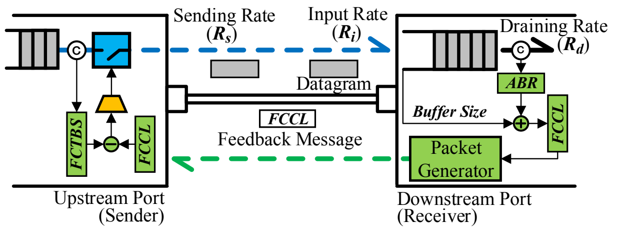
Circular wait: Circular wait refers to Cyclic Buffer Dependency (CBD) existing in the network. CBD is defined as a cyclic sequence of switch buffers. Each switch in the sequence sends packets to the next switch in the sequence [52]. Although the self-loop can be completely avoided in advance, many other conditions (e.g., link failures [24] and network updates [32]) may force multiple flows to form a CBD.

2.2 Flow controls for lossless networks

Two mainstream communication networks, namely CEE and InfiniBand, are deployed in current data centers and clusters to provide lossless packet transmission. Ethernet is the dominant communication network used in data center scenarios. Data Center Bridging (DCB) task group [28] proposes CEE to meet advanced application requirements. InfiniBand, on the other hand, is widely-used in HPC field [49], which supports more than half of HPC systems in the world [41]. Both CEE and InfiniBand leverage hop-by-hop flow control mechanisms to guarantee zero packet loss [28, 29, 40], as shown in Figure 2. Their fundamental principles are similar. The sender (upstream port) sends packets at line rate, and the receiver (downstream port) generates feedback messages carrying the buffer information back to the sender, so the sender knows when to stop transmission to prevent overflow in the receiver's input buffer. Furthermore, if the flow control message notifies more available buffer, the sender will resume sending packets. We elaborate existing flow control mechanisms for zero packet loss as follows:



(a) PFC



(b) CBFC

Figure 2: Flow controls for zero packet loss.

2.2.1 Priority flow control. In order to enhance traditional Ethernet for losslessness, DCB task group proposes the PFC mechanism [27]. As shown in Figure 2(a), the receiver pauses the sender when the ingress queue length exceeds a preset threshold ($XOFF$). When the ingress queue length decreases below another threshold (XON), the receiver will resume the sender. As long as enough headroom beyond $XOFF$ is reserved for absorbing in-flight packets before PAUSE takes effect, buffer would never overflow. In PFC standard [27], 8 different priorities are defined. Each priority holds a separate queue, and different types of traffic enter into different priority queues. Each PFC message (PAUSE/RESUME) carries the priority information, and can only act on the corresponding priority queue.

PFC alternates between pausing and resuming to make the sending rate of the upstream port match the draining rate of the downstream port in the long term. However, when a port pauses the upstream port, all packets in the upstream port fall into the *hold and wait* situation.

2.2.2 Credit based flow control. InfiniBand deploys CBFC to eliminate packet loss [4]. In CBFC, the buffer space is divided into credits, and the receiver controls the pausing/resuming of sender by offering credits. As shown in Figure 2(b), the receiver maintains an Adjusted Blocks Received (ABR) register, which counts all blocks received since link initialization. By adding up the ABR value and the allocated buffer size, the receiver can calculate out Flow Control Credit Limit ($FCCL$). The receiver updates the $FCCL$ value to sender by generating feedback messages periodically. The sender maintains a Flow Control Total Blocks Sent ($FCTBS$) register to track the number of total blocks sent since link initialization. So the difference between $FCCL$ and $FCTBS$ represents the available credits. When the number of available credits is larger than the packet size, the sender is allowed to send a packet. Otherwise, the sender will cease until the $FCCL$ value is updated. Strictly keeping the number of sent blocks less than the received $FCCL$, zero packet loss can be achieved. Once the sender runs out of credits, it enters into the *hold and wait* state. By updating the $FCCL$ value periodically, the sending rate is adjusted to the draining rate in the long term.

2.2.3 Brief summary. Both PFC and CBFC are proposed to satisfy the lossless requirement, and they follow the similar rationale: the receiver feedbacks the ingress queue status to the sender, so the sender can stop timely to avoid buffer overflow and resume transmitting after queue length decreasing. There are mainly two differences: (1) PFC generates messages according to whether the queue length exceeds the threshold or not (buffer-based), while CBFC generates flow control messages periodically (time-based). (2) In PFC, when the downstream port generates the feedback message, it directly decides the upstream port status (PAUSE/RESUME). In CBFC, feedback messages only carry the credit information. The sender needs further calculation to determine the port status. This difference only concerns where the decision is made. In brief, both distinctions are not radical.

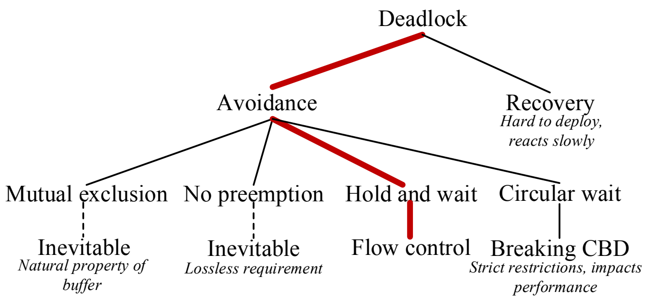


Figure 3: Possible approaches for solving network deadlock.

In both flow control schemes, the sender alternates between sending at line rate (ON) and pausing (OFF). With these alternations, the sending rate of upstream port is adjusted to match the draining rate of downstream port on a long time scale. On one hand, it can avoid packet loss. On the other hand, the pausing status unavoidably leads to the *hold and wait* situation (one necessary condition for deadlock).

3 DESIGN DECISIONS

As shown in Figure 3, deadlock can be handled through either avoidance or recovery. Furthermore, breaking any one of aforementioned four necessary conditions can avoid deadlock. This section presents why we choose the solution path of designing GFC to solve network deadlock.

3.1 Why deadlock avoidance?

The first question to be answered is how to deal with the network deadlock problem: to avoid it before its emergence or to recover from it after its occurrence?

For recovery, the first challenge is detecting deadlocks in distributed networks. It is non-trivial since detecting deadlock requires network devices to generate probing packets and support special protocols processing them. In addition, after detecting the existing loop, dropping packets and heuristic rerouting are main recovering solutions [2, 3, 36, 38, 52], which are very disruptive [24].

Compared with recovery, avoidance is a much more proactive and efficient approach since it can eradicate the root cause of deadlock.

3.2 Why avoiding hold and wait?

To avoid deadlock, one of its necessary conditions needs to be broken. As aforementioned, there are totally four necessary conditions. Both *mutual exclusion* and *no preemption* are unbreakable in lossless networks: buffer resource is inherently mutual exclusive and the lossless requirement enforces the *no preemption* condition must be satisfied.

Existing deadlock avoidance solutions mainly focus on breaking the *circular wait* condition. Many CBD-free routing solutions have been proposed [7, 13–15, 17, 18, 21, 50, 54–56, 58]. The basic idea is restricting routings to guarantee no combination of flows could form CBD. However, these solutions introduce great limitations to network configurations and disable some advantages (e.g., balancing load through intrinsic multiple paths). Another approach is to isolate CBD-prone flows into different virtual channels and employ independent queues to serve them [6, 20, 35]. However, the required number of priority queues increases with network scale, which greatly affects the scalability. Tagger [25] follows the similar idea but requires less priority queues. Nevertheless, its dynamic management of packet priority is complex and packets would be dropped when priority queues are not enough. To decrease the required resources, other dynamic deadlock avoidance solutions are also developed [16, 46]. They have no restrictions on network when running normally. However, when the buffer is full, strict restrictions are still imposed for avoiding the occurrence of CBD. Once system enters the deadlock avoidance status, the performance decreases drastically and it takes a long time to recover back.

We attempt to work out a brand-new solution to break the *hold and wait* condition, which, to the best of our knowledge, is an unexplored approach to overcome network deadlock.

Feasibility: Primarily, is eliminating *hold and wait* condition feasible? The answer is positive. Existing flow control mechanisms manage port rate in a coarse way (ON/OFF) to achieve the matching between input rate and draining rate. It is the root cause of frequent occurrences of *hold and wait*. For each port, as long as the input rate is elaborately controlled to match its draining rate, all flows can continuously pass through the network even if CBD exists. In this situation, all flows maintain proper sending rates. The queue length in each port will keep steady and never trigger *hold and wait*. For example, we consider again the deadlock scenario in Figure 1. Assume the line rate is 10Gbps, as long as all hosts keep sending packets at 5Gbps, every flow can continuously pass through the network. The queue length in each port is steady and no *hold and wait* occurs. Therefore, the key to avoiding *hold and wait* is designing a novel flow control mechanism, which can *gently* match the sending rate of upstream port with the draining rate of downstream port.

4 DESIGN

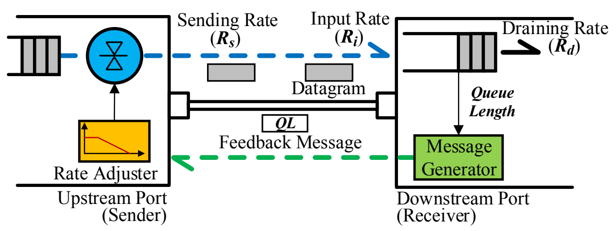
The basic idea is to design a flow control scheme which adjusts the sending rate at a fine granularity. Thus packets can consecutively flow and further *hold and wait* is eliminated.

This section first presents the conceptual design of GFC, and demonstrates it can completely avoid the occurrence of *hold and wait*. However, the conceptual scheme assumes feedback messages can be generated continuously, which is impractical. We further propose the practical design of GFC that occupies only a little bandwidth resource while eliminating *hold and wait*.

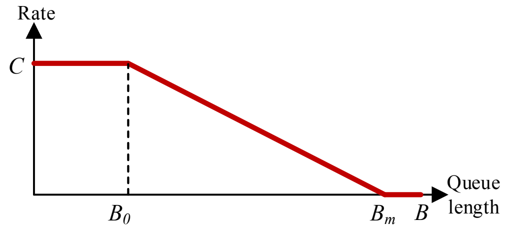
4.1 Conceptual Design

Figure 4(a) presents our conceptual scheme, which contains two main functional components: Message Generator and Rate Adjuster. Message Generator monitors the instantaneous ingress queue length and generates feedback messages carrying the related information back to upstream ports. We first assume that flow control messages can be generated continuously (this constraint will be removed in the subsequent practical design). Rate Adjuster in the upstream port receives feedback messages, parses queue length information, and calculates the sending rate according to the given mapping function. Then the upstream port will send packets in line with this rate. An illustrative mapping function is depicted in Figure 4(b), where B is the input queue buffer size and $B_m \leq B$. Initially, the sender is allowed to transmit packets at line rate. When the ingress queue length in the downstream port exceeds the threshold B_0, the upstream port begins to decrease its sending rate. The rate decrement is proportional to the downstream port queue length. So with queue length increasing, the gap between sending rate and draining rate becomes smaller. Eventually, the system enters a steady state where the sending rate matches the draining rate and the ingress queue length keeps unchanged. Leveraging this fine-grained rate adjustment, switches can prevent packet loss without triggering *hold and wait*. Essentially, the basic operation in GFC is still matching the sending rate with the draining rate, which is the same as existing hop-by-hop flow controls (PFC and CBFC). Thus, in the long time scale, GFC would not waste forward available bandwidth.

Firstly, we use a simple example to demonstrate how conceptual GFC works to avoid *hold and wait*. Consider a 2-to-1 case where two flows from different input ports compete for the same output port, a general congestion situation where sending rate > draining rate. The link capacity is 10Gbps so the draining rate of each ingress queue is 5Gbps. The feedback latency τ (the time interval between generating a feedback message and receiver perceiving the input rate changing accordingly) is 25μs. The maximum buffer size B_m and B_0 are 100KB and 50KB, respectively. We employ PFC as the baseline comparison (results of CBFC are similar). $XOFF$ is 80KB, and XON is 77KB (the recommended interval between $XOFF$ and XON is $2MTU$ [59]). Figure 5(a) shows the simulation results. Under PFC, the ingress queue length first directly exceeds $XOFF$ and then fluctuates near XON and $XOFF$. The sending rate alternates between zero and line rate accordingly. After the system enters the steady state, the intervals between ceasing and full-rate sending are identical, so PFC regulates the sending rate to the draining rate in the long term. Obviously, the upstream port goes into the cease status frequently. Thus, packets in the upstream egress queue

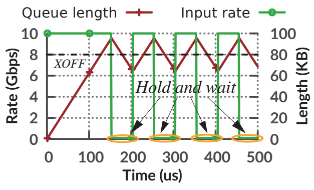


(a) Conceptual GFC

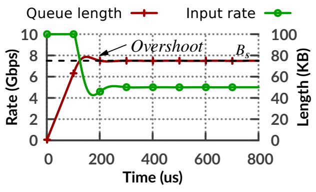


(b) Mapping function

Figure 4: Conceptual design.



(a) PFC



(b) GFC

Figure 5: Evolutions of input rate and queue length.

experience the *hold and wait* situation. If increasing the gap between *XOFF* and *XON*, the fluctuation period will increase and so will the duration of each *hold and wait* phase, which is more prone to deadlock. The results of GFC are shown in Figure 5(b). When the ingress queue length exceeds B_0, the sender is notified to decrease rate accordingly. The gap between sending rate and draining rate decreases so the ingress queue length increases slower. Finally, the ingress queue length stays at the steady-state value B_s (B_s = 75KB in this case), where the sending rate equals the draining rate.

Ideally, if the feedback delay $\tau = 0$, the input rate changes strictly in line with the mapping function. Then the ingress queue length will directly converge to the stable point where the sending rate equals to the draining rate. So packets can continuously transmitted without triggering *hold and wait* and there is further no deadlock. However, τ is nonnegligible in practice. It postpones the reactions to feedback messages, which may lead to queue length overshoot and introduces the overflow risk. In order to totally avoid *hold and wait*, the mapping function should be carefully settled.

If $B_m - B_0$ is large, abundant buffer room is reserved for absorbing overshoot. However, it also means more buffer space is required. In the practical network, the switch buffer is shallow and precious. Therefore, it is an important task to determine the proper value of B_0. Next, we will theoretically find out the bound of B_0, and demonstrate it can completely avoid *hold and wait* in the flow control procedures.

Eliminating hold and wait: We first give an intuitive explanation why our proposed conceptual design can eliminate *hold and wait*. Then the strict condition for eliminating hold and wait is provided. Generally, overshoot starts when ingress queue length increases to B_s. The Message Generator sends a message containing B_s to the upstream port. However, it needs τ to make input rate R_i decrease to R_d. So overshoot accumulates during this interval, and its maximum value $\triangle B = \tau(\bar{R_i} - \bar{R_d}) \leq \tau\bar{R_i}$, where $\bar{R_i}$ and $\bar{R_d}$ denote the average input rate and draining rate in the interval τ, respectively. According to the mapping function, if remaining buffer ($B_m - B_s$) is small, then $\bar{R_i}$ would be small, and vice versa (i.e., $\bar{R_i} \propto B_m - B_s$). So the worst-case value of ΔB is positively correlated with $\tau(B_m - B_s)$ under the linear mapping function. In hence, as long as we can set the slope of mapping function (i.e., $\frac{C}{B_m - B_0}$) small enough, then overshoot ($\triangle B$) can always be absorbed by the remaining buffer ($B_m - B_s$). It means that queue length never reaches B_m and input rate never goes to zero. In this way, *hold and wait* is completely eliminated.

Employing some mathematical methods, we give Theorem 4.1 to determine the strict constraint for eliminating *hold and wait*. The detailed deduction can be found in Appendix A.

Theorem 4.1. *In conceptual GFC, if* $B_0 \leq B_m - 4C\tau$, *hold and wait can be avoided.*

Setting mapping function strictly following Theorem 4.1, sending rate would never reach zero and the ingress queue length never reach B_m. So B_m can be directly set equal to the buffer size B.

This conceptual design is impractical because sending feedback messages too frequently will greatly waste backward bandwidth. In the following part, we propose the practical GFC, which only occupies a small portion of bandwidth.

4.2 Practical Design

In order to suppress excessive feedback messages, we change the continuous mapping function to the step function, as shown in Figure 6. Thus the receiver only sends feedback messages when the queue length changes from one stage to another. Message Generator monitors the ingress queue length. When the queue length steps into a new stage (e.g., the i^{th} stage), it generates a new feedback message carrying the stage ID i back to the sender. On the sender side, when Rate Adjuster receives a feedback message, it parses out i, maps it to the target rate, and changes the sending rate correspondingly.

Intuitively, decreasing the frequency of message generation leads to coarse-grained feedback and may further aggravate the buffer overshoot. So the design of stage mapping function is crucial for avoiding *hold and wait*. Next, we discuss how to fix the stage mapping function. There are three key parameters: the queue length at the start of the n^{th} stage B_n, the corresponding sending rate of the n^{th} stage R_n, and the total number of stages N.

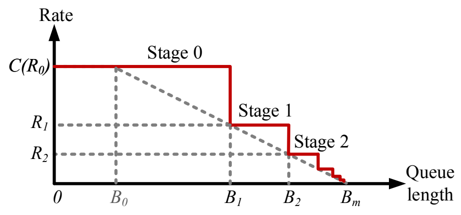


Figure 6: Multi-stage mapping function.

B_n **and** R_n**:** The length of each stage plays an important role in avoiding *hold and wait*. If the lengths of stages are too small, the input buffer may be quickly exhausted before triggering the upstream port decreasing the sending rate. On the other hand, limited buffer resource disallows setting oversize stages.

In practical GFC, we propose a safe way to determine the length of each stage: for any $k \geq 1$, before the ingress queue length exceeds the k^{th} stage, the input rate should have already changed to R_k. To achieve this goal, the length of a stage should be large enough for the feedback message to take effect. Therefore, the relationship between B_k and R_k should be

$$B_{k+1} - B_k \geq R_{k-1}\tau \tag{1}$$

To keep accordance with the linear decreasing, the stage function needs to satisfy

$$\frac{B_k - B_{k-1}}{R_{k-1} - R_k} = \frac{B_m - B_0}{C} \tag{2}$$

Combining (1) and (2), we have:

$$R_k \leq \left(1 - \frac{C\tau}{B_m - B_0}\right) R_{k-1} \tag{3}$$

According to Theorem 4.1, the constraint for eliminating *hold and wait* is $B_0 \leq B_m - 4C\tau$. Substituting it into (3), we have $R_k \leq \frac{3}{4}R_{k-1}$.

We select

$$R_k = \frac{1}{2}R_{k-1} = \frac{1}{2^k}C \tag{4}$$

Combining (2), we also have

$$B_m - B_k = \frac{1}{2^k}(B_m - B_0) \geq 2^{2-k}C\tau \tag{5}$$

Note that the stage 0 $[B_0, B_1)$ has the same function as when the queue length is below B_0, thus we remove the stage 0. In other words, the buffer length should be larger than $B_m - B_1$. According to (5), $B_m - B_1 \geq 2C\tau$, thus the buffer size just needs to be larger than $2C\tau$. Notice that the required *headroom* in PFC is at least $C\tau$ [27], so $2C\tau$ of buffer length is an attainable constraint in practice.

N**:** According to Equation (4), R_n will infinitely approach but never reach zero. So the number of stages should be infinite in this step function. However, in real design, it is unnecessary and impractical to set very small rates. Each time when R_n halves, the remaining buffer halves too. In practice, the buffer is consumed in unit of 8-bit. So once $B_n - B_{n-1} \leq 8b$, the following stages are omissible.

Occupied bandwidth: By changing into multi-stage mapping function, one may ask how much bandwidth expenditure is introduced. Here we give a brief analysis. In the worst case, as discussed in deducing Equation (1), feedback messages are generated every τ time. So the occupied bandwidth is $\frac{m}{\tau}$, where m is the size of feedback message. However, this worst-case situation is only a transient. In steady status, the worst-case occupied bandwidth is only $\frac{m}{8\tau}$. The calculating details are omitted due to limited space. Taking 10GbE as an example, $m = 64B$ and $\tau = 7.4\mu s$ (detailed calculation of τ can be found in Section 5.4), then the occupied bandwidth is 69Mbps (0.69%) in the worst case and 8.6Mbps (0.086%) in the steady case, which is very small.

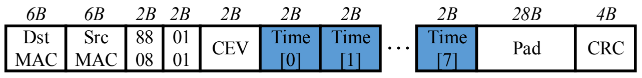


Figure 7: PFC message type.

5 IMPLEMENTATION

As introduced in Section 2.2, mainstream lossless networks have implemented their own flow control frameworks. This section presents how to implement GFC on the basis of these frameworks with moderate modifications. Primarily, existing flow controls employ different mechanisms to trigger the generation of feedback messages. (1) PFC generates a feedback message when the queue length reaches the predefined threshold (buffer-based). (2) CBFC generates messages periodically (time-based). We will present how to implement GFC's Message Generator and Rate Adjuster under these two interaction modes, respectively. Next, compared with existing lossless flow controls, GFC requires a new component: Rate Limiter. We show how to implement it with low overhead. The whole processes of GFC can be implemented through updating the firmware in commercial switches. Finally, the parameter settings of buffer-based/time-based GFC are explicitly presented.

5.1 Buffer-based GFC

Message Generator. PFC has implemented the Message Generator module to generate PAUSE/RESUME messages. We can reutilize this module and only change (1) the trigger for generating a new feedback message and (2) the information carried in the message. In buffer-based GFC, we add multiple thresholds (rather than two in PFC) to implement the multi-stage rate control. The threshold values are determined according to Equation (5). Once the queue length changes across a threshold, Message Generator is triggered to generate a feedback message.

The feedback message should be modified to carry the stage information. The original PFC message type is given in Figure 7. The Class-Enable Vector (CEV) represents on which priority this message takes effect. Time[0] ~ Time[7] are used to store the duration of PAUSE. In our implementation, we modify Time[0] ~ Time[7] to encode the stage information. Since we only need to carry the stage information of queue length, so a two-byte field is enough.

Rate Adjuster. When receiving a feedback message, Rate Adjuster extracts the queue length information and calculates the sending rate accordingly. Feedback message generated by buffer-based Message Generator contains the queue length stage information, which can be directly mapped to the sending rate. Since the mapping function is a step function, Rate Adjuster can directly use an array to store the mapping relationship. This implementation consumes

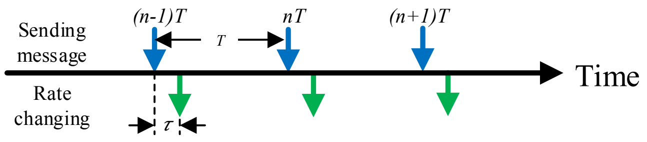


Figure 8: The event sequence in the receiver side of time-based GFC.

few storage resources and eliminates rate calculations. Then Rate Adjuster updates Rate Limiter in the corresponding egress queue to conduct rate modification. The detailed implementation of Rate Limiter is presented in Section 5.3.

5.2 Time-based GFC

Message Generator. Notice that in CBFC, downstream switch port generates feedback messages periodically. This mechanism can also achieve the goal of limiting feedback frequency. So it is reserved in time-based GFC. Furthermore, in CBFC, feedback messages already carry the remaining buffer information (credit), which is enough for GFC. So we can directly reuse this Message Generator without any modification.

Rate Adjuster. Messages sent from Message Generator are triggered by time rather than stage boundaries, which means the deduced multi-stage mapping function is no longer appropriate in this scenario. We work out a new mapping function in time-based Rate Adjuster based on the conceptual design (Section 4.1).

Assume the feedback period is T, the time sequences of sending messages and updating input rate are shown in Figure 8. At the end of the n^{th} round, Message Generator sends back a message carrying current credit information. At present, the input port receives packets at the rate calculated in the $(n-1)^{th}$ round. It takes τ for the new feedback message to take effect. Then the input rate changes to the new rate calculated in the n^{th} round, and this rate will remain unchanged in the next T interval. The introduction of T will blunt the control of sending rate. In order to avoid *hold and wait* in this scenario, we give Theorem 5.1 to determine the bound of B_0. Appendix B gives the detailed proof.

Theorem 5.1. *In time-based GFC, if* $B_0 \leq B_m - (\sqrt{\frac{\tau}{T}} + 1)^2 CT$, *hold and wait can be avoided.*

Under the guidance of Theorem 5.1, the mapping function can be determined. Once Rate Adjuster receives a feedback message, it extracts $FCCL$ and calculates out the remaining buffer size ($FCCL - FCTBS$). Then the corresponding sending rate is calculated through the mapping function and Rate Limiter is updated accordingly.

5.3 Rate Limiter

In GFC, each queue needs an independent Rate Limiter to determine when to send the next packet. Actually, many network equipment manufactures (e.g., Cisco, Juniper Networks and Mellanox) already support this feature in their commercial off-the-shelf switches [11, 34, 39]. Even implementing GFC Rate Limiters from scratch, it only introduces small overhead.

(1) Implementing one GFC Rate Limiter occupies few resources. Rate Limiter can be implemented by employing three registers (R_l, R_r, and R_c) and adding some auxiliary processes. R_l records the packet transmission time. R_r works for storing the assigned queue rate. R_c runs as a countdown timer. Each time a packet is completely sent, the corresponding transmission time is recorded in R_l. R_c is set to $\frac{(C-R_r)}{R_r}R_l$ and starts to count down. When $R_c > 0$, this queue cannot send packets. After R_c counts down to zero, this queue is allowed to send a new packet. This processing logic can be implemented with small overhead. Furthermore, since 10Gbps ports usually run at a clock rate of 833MHz [8], the counting down of R_c can be implemented at the granularity of 1.2ns, which provides sufficient accuracy for our rate limiting.

(2) The number of required Rate Limiters is limited. In commodity switches, each port usually contains 8 (or less) priority queues [22, 25, 28, 40]. Considering a 24-port switch, it requires 192 Rate Limiters at most. Furthermore, less priorities are employed for lossless traffic in actual deployments. For example, in RoCE, only two or three priorities are supported by commodity switches [22, 25], which makes the number of required Rate Limiters decrease to 72. On the other hand, in SENIC [47], 1000 high-accuracy 10Gbps rate limiters are implemented on programmable hardware by just consuming 30KB SRAM resources (0.1%). So the overhead introduced by implementing rate limiters is affordable.

5.4 Parameter Settings

We first give the detailed calculation of τ, which is the foundation of determining parameters in mapping functions. Then the parameter settings in buffer-based/time-based GFC are illustrated, respectively.

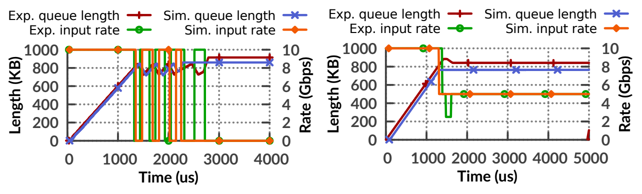
The elements of τ. Both Theorem 4.1 and 5.1 tell that τ plays an important role in determining parameters in GFC mapping function. The following part presents all constituent parts of τ in practice.

(1) When the receiver is ready to emit a feedback message, this message cannot interrupt on-going packet transmission on the port. In the worst case, this message will be delayed for $\frac{MTU}{C}$, where MTU is the maximum transmission unit.
(2) Transmitting packets through the wire needs the time t_w. On the wire, data travels at the speed of about 2×10^8m/s. So transmitting a packet for 100m takes about 512ns.
(3) Then this feedback message arrives at the sender side. It takes t_r for the sender to process this message. The length of t_r depends on the implementation of processing logic and the upper limit is 3μs [10].
(4) After processing the feedback message, the sender changes the output rate accordingly. However, this modification can only be done when no packet is transmitting on the port. In the worst case, it needs to wait for another $\frac{MTU}{C}$.
(5) Finally, the opposite transmission latency t_w is needed to make this change reflecting on the input rate.

By adding all these elements together, the value of τ can be upper bounded by Equation (6).

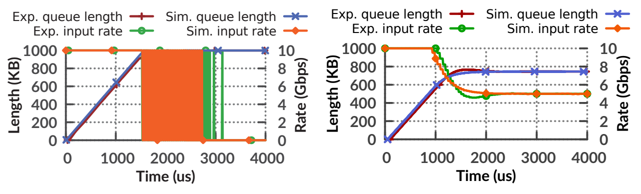
$$\tau \leq \frac{2MTU}{C} + 2t_w + t_r \tag{6}$$

Assume that $C = 10/40/100$Gbps and $t_w = 1\mu$s. In CEE, the $MTU = 1.5KB$, so the worst-case τ is 7.4/5.6/5.2μs, respectively.



(a) PFC (b) Buffer-based GFC

Figure 9: The experiment results of PFC and buffer-based GFC.



(a) CBFC (b) Time-based GFC

Figure 10: The experiment results of CBFC and time-based GFC.

The MTU in InfiniBand is $4KB$, so the worst-case τ is $11.4/6.6/5.6\mu s$, respectively.

Parameters in buffer-based GFC. In buffer-based GFC, two parameters (B_m and B_1) need to be determined. All other parameters can be uniquely deduced. Since the buffer space (B_m, B) would never be used in practice, B_m can be set to B for fully utilizing buffer resource. We directly set B_1 (rather than B_0) here because in multi-stage mapping function the mapped rate of stage 0 is equal to the link capacity C. The constraint of B_1 is that $B_1 \leq B_m - 2C\tau$. Once the physical network is built up completely, the values of B_m, C and τ are determined (in commodity 10/40/100Gbps network, $2C\tau \leq 18.5/56/130$KB, respectively). Corresponding parameters of stage k ($1 \leq k \leq N$) can be calculated as follows: $R_k = \frac{1}{2^k}C$ and $B_k = B_m - \frac{1}{2^{k-1}}(B_m - B_1)$. The total stage number N should make $B_N - B_{N-1} \leq 8$b. Considering $C = 10/40/100$Gbps, $N = 16/18/20$, respectively.

Parameters in time-based GFC. In time-based GFC, three parameters (T, B_m and B_0) need to be configured. Similarly, B_m is set equal to B. The setting of feedback period T concerns a trade-off between buffer and bandwidth. A small T decreases the required buffer size, but increases the overhead on bandwidth, while a large T leads to the opposite case. In CBFC, the recommended value of T is the time required to trasmit 65535B data quantity [40]. This recommended value can be followed in time-based GFC. According to Theorem 5.1, Equation (2) and (4), $B_0 \leq B_m - (\sqrt{\frac{\tau}{T}} + 1)^2 CT$ (in commodity 10/40/100Gbps network, $(\sqrt{\frac{\tau}{T}} + 1)^2 CT \leq 140.8/191.4/271$KB, respectively).

6 EVALUATION

In this section, we evaluate GFC with testbed experiments as well as large-scale simulations. Our testbed experiments verify that buffer-based/time-based GFC can control the sending rate elaborately and eventually avoid deadlock. The detailed packet-level simulations confirm that GFC schemes can take good effect and introduce negligible side-effects in practical network scenarios.

6.1 Experiments

We build up the experiment topology as shown in Figure 1(a). Employing GFC schemes in this typical deadlock-prone network, we can verify the effectiveness of GFC.

6.1.1 Testbed implementation. In the testbed, three servers, each equipped with single Intel i7-4700 CPU (4 cores, 3.4GHz), act as hosts. Since existing commodity switches do not provide interface to modify inbuilt flow control mechanisms, we leverage software switch to customize flow control mechanisms. Three servers, each equipped with dual Intel Xeon E5-2620 v3 CPUs (6 cores, 2.4GHz), are used to build software switches. Each server is plugged with 2 dual-port Intel 82599 10G NICs. So each server can work as a four-port switch. DPDK [30] is employed to deliver received data directly to user space. The basic L2 switch is built based on the reference test-pipeline project [31]. To achieve line-rate sending/receiving at 10Gbps, each RX/TX module is implemented on an individual core. Forwarding module runs on a separate core and a hash table in address-port mapping is used to reduce the lookup delay. We implement buffer-based/time-based GFC based on this basic software switch platform. For comparison, we implement PFC according to IEEE 802.1Qbb [27] and CBFC according to InfiniBand Specification [4]. Although the testbed is built based on Ethernet, the physical layer media hardly affects the process and performance of layer-2 flow control. Therefore this testbed is capable of evaluating above flow control mechanisms.

Parameter configurations: The input buffer size is 1MB. Through our measurement, the worst-case value of τ is $90\mu s$. Here τ is a few times larger than the value in the commodity switch because of software implementation. In PFC, $XOFF$ is set to 800KB and XON is 797KB. In buffer-based GFC, B_1 is set to 750KB. In CBFC, the feedback period is $52.4\mu s$. In time-based GFC, the period remains unchanged and B_0 is 492KB. In experiments, all servers start together and generate packets at line rate.

6.1.2 Results. Both PFC and CBFC trap into deadlock. Tracing the evolutions of input rate and queue length of the switch port connecting to $H1$ as a representative (the states of other ports are similar), the results are shown in Figure 9(a) and Figure 10(a). In both experiments, it takes some time to fill up the ingress queue. However, owing to the intrinsic pausing/resuming behavior pattern, network falls into deadlock finally.

When employing GFC schemes, network deadlock is avoided. The corresponding results are shown in Figure 9(b) and Figure 10(b). In buffer-based GFC, the queue length first climbs up to 884KB, which triggers $H1$ to transiently decrease the output rate to 2.5Gbps. Then the queue length drops back to 840KB. The overshoot is noticeable. However, under the adjustment of mapping function, the queue length comes back to the expected status quickly and then keeps steady.

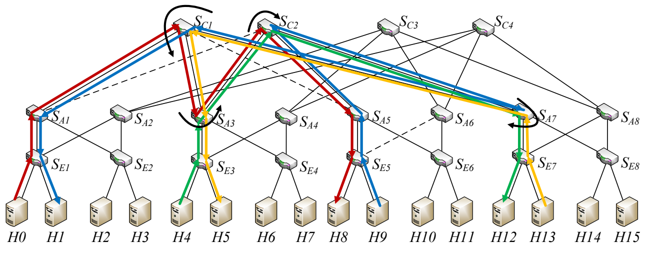
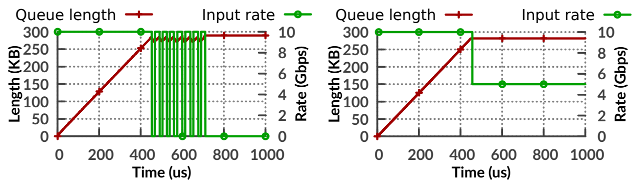
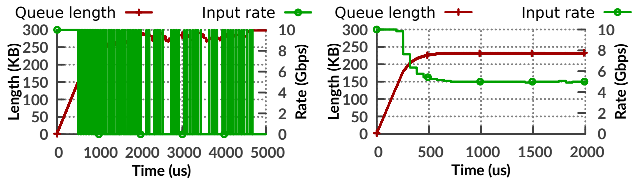


Figure 11: Fat-tree topology ($k = 4$) with three failed links (represented with dash lines).



(a) PFC (b) Buffer-based GFC

Figure 12: The simulation results of PFC and buffer-based GFC.



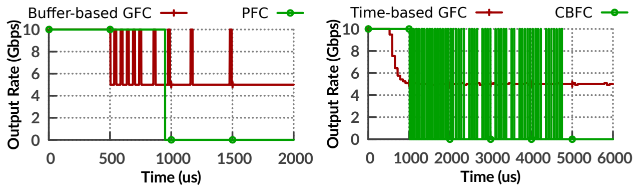
(a) CBFC (b) Time-based GFC

Figure 13: The simulation results of CBFC and time-based GFC.

As shown in Figure 10(b), in time-based GFC, after the initial fluctuation, the queue length stabilizes at 745KB and the input rate maintains at 5Gbps as well. In time-based GFC, the sending rate changes after each feedback period and gradually converges to the steady state. So the evolution of sending rate is smoother compared with that in buffer-based GFC, which can only switch rate among predefined stages. Notice that there is slight overshoot in time-based GFC experiment. There are two main reasons: (1) The τ value considered in parameter configuration is a worst-case value, which can be rarely reached in practice. (2) The deduced bound of B_0 in time-based GFC is relatively slack, thus abundant buffer space is reserved to make the rate adjusting process much more smoothly. Small overshoot is actually a good property in practical deployment.

6.2 Simulations

We leverage OMNET++ [45], a packet-level simulation platform, to build our simulator. Both PFC and CBFC are implemented according to corresponding standards [4, 27], respectively. First the same network scenario as in testbed experiments is used to validate our simulator. Then, a practical 3-layer network deadlock case is built to evaluate the performance of GFC schemes. Finally, we randomly generate large-scale network scenarios that are prone to deadlock to test the comprehensive performance of GFC.



(a) Buffer-based GFC (b) Time-based GFC

Figure 14: Victim flow.

6.2.1 Simulator verification. To validate our simulator, we first build up the same 3-switch deadlock scenario shown in Figure 1(a), and use the same parameter settings in above experiments. The results are also presented in Figure 9 and Figure 10. In all four simulations, the main conclusions remain consistent: both PFC and CBFC fall into deadlock and GFC schemes eliminate it. Since switches in testbed are implemented based on DPDK, the software-based processing would introduce uncertain latency and further affect the evolution of sending rate and queue length. So, there are some transient divergences between experiments and simulations. But the overall evolutions keep good consistence. These results validate our simulator, therefore we use it for further evaluations.

6.2.2 Deadlock case study. As shown in Figure 11, we use a 3-layer fat-tree [1] with $k = 4$ as our case-study topology, and employ the shortest-path-first routing algorithm. Three link failures, marked with dashed lines, make the network prone to deadlock. Four flows are introduced into the network: $F_1 : H0 \to H8$, $F_2 : H4 \to H12$, $F_3 : H9 \to H1$ and $F_4 : H13 \to H5$. Then there is a four-hop CBD in the network: $S_{C1} \to S_{A3} \to S_{C2} \to S_{A7} \to S_{C1}$. In simulations, the input buffer size is 300KB, the link capacity is 10Gbps and the propagation delay is $1\mu s$. In PFC, $XOFF$ is 280KB and XON is 277KB. In buffer-based GFC, B_1 is set to 281KB, and $B_{n+1} - B_n = (\frac{1}{2})^n \times 19$KB. In time-based GFC, B_0 is set to 159KB.

The results are presented in Figure 12 and Figure 13. In both PFC and CBFC, the network falls into deadlock. While employing GFC, deadlock is avoided and each flow shares 5Gbps of bandwidth normally.

Victim flow: As aforementioned, when deadlock occurs, it would not only stop all flows associated with the CBD but also prevent other CBD-irrelevant flows. In other words, a local deadlock may disable a large part of network. Here we add a new flow (F_5) into the network. This flow travels from $H4$ to $H5$ with the forwarding path $H4 \to S_{E3} \to H5$, which does not pass through the CBD. We conduct above simulations again. The throughput of F_5 are shown in Figure 14(a) and Figure 14(b). In both CBFC and PFC, deadlock occurs and the throughput of F_5 goes to zero. The root cause is that deadlock pauses all flows entering the CBD, and such pausing behavior will propagate hop-by-hop back to the sources of these flows. Then irrelevant flows sharing part of this path are victimized.

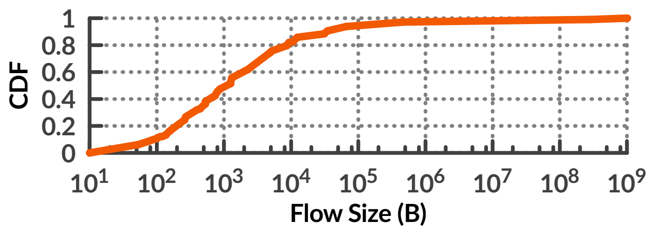


Figure 15: Empirical traffic pattern.

Table 1: Statistical results of deadlock cases

Scale	PFC	Buffer-based GFC	CBFC	Time-based GFC
$k = 4$	32	0	32	0
$k = 8$	12	0	12	0
$k = 16$	2	0	2	0

By employing GFC, deadlock is eliminated and F_5 can share its deserving 5Gbps throughput normally.

6.2.3 Large-scale simulations. In this part we evaluate the overall performance of GFC in large-scale networks. The fat-tree topology with $k = 4$, 8 and 16 is used. The settings of buffer size and link capacity remain unchanged, so do the parameters in different flow control mechanisms. In this set of simulations, the failure probability of each link is 5%. We randomly generate 10000 networks under each scale and employ the shortest-path-first routing algorithm. As shown in Figure 15, the input workload is based on empirically observed enterprise traffic patterns [57]. Each host randomly chooses a destination in different racks to start a new flow. Once this flow is finished, the host repeats the above process. In each selected network, we run different flow control mechanisms, respectively.

Since the topologies and routing algorithm are determined, we can filter out cases which are prone to generate CBD in advance. In order to judge whether deadlock occurs or not under these scenarios, we repeat each simulation for 100 times. For a certain flow control in a specific topology, if deadlock occurs in any of these 100 simulations, we consider this scenario as a deadlock case. Results for all flow control mechanisms under each network scale are listed in Table 1. Results show that deadlock only occurs under PFC and CBFC. With the increase of network scale, less deadlock cases come out. Since large-scale topology possesses more optional forwarding paths, random link failures are harder to generate a CBD. It is worth noticing that PFC and CBFC cause the same number of deadlock cases under all network scales. Actually, through our careful check, they fall into deadlock in the same topologies. By employing GFC schemes, deadlock is avoided in all network scales.

Overall performance: Operators may have interest in whether deploying GFC would introduce side-effect to the overall network performance (e.g., wasting available bandwidth, increasing latency), especially when there is no CBD in the network. To answer this question, 100 CBD-free cases under each network scale are randomly selected to conduct further simulations. We record the average throughput of servers and the average *slowdown*, which is the actual flow completion time (FCT) divided by the shortest possible time for a same-sized flow to finish in an unloaded network. The throughput values are worked out through counting sent bytes every 100μs. When there is no CBD in network, those failed links only constrain forwarding selections. Under these scenarios, different kinds of flow control all work for port-level rate adjustment only, which would introduce small difference on overall performance. In Figure 16(a) and Figure 17(a), the statistical values are similar among different flow control mechanisms (buffer-based/time-based GFC and PFC/CBFC). It confirms that GFC does not introduce extra bandwidth waste or FCT increase. The standard deviation of throughput is smaller in buffer-based/time-based GFC than in PFC/CBFC because GFC adjusts input rate at a much finer granularity. When a deadlock is generated by employing PFC and CBFC, the entire network is further affected and disabled, which leads to zero average bandwidth and infinite FCT as shown in Figure 16(b) and Figure 17(b). By employing GFC, deadlock is eliminated. Furthermore, compared with results under deadlock-free cases, the differences in the macroscopic performance are small. Although specific combination of flows still generate CBD when deploying GFC, these flows can continuously pass through the CBD. Once any flow in this combination is finished, the CBD is naturally broken and there is no further side-effect on overall network utilization. A more detailed case study is given in the following part to conduct an in-depth investigation.

Case study: We select one of deadlock-prone simulations in fat-tree network ($k = 16$) as an example. The evolutions of average throughput under buffer-based GFC and PFC are depicted in Figure 18. After initialization, all servers send traffic at line rate. Quickly the switch queues build up and flow control mechanisms take effect to restrict sending rates. Since the destinations of flows

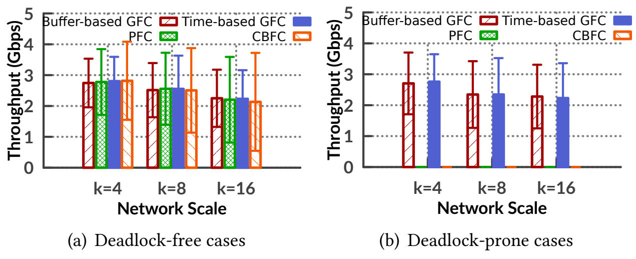


Figure 16: Average available bandwidth.

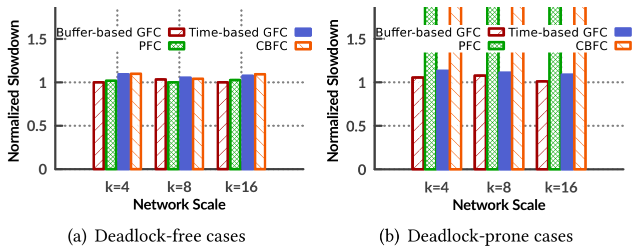


Figure 17: Average slowdown (normalized to minimum in each scale).

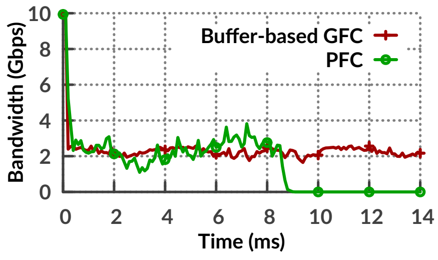


Figure 18: Throughput evolution.

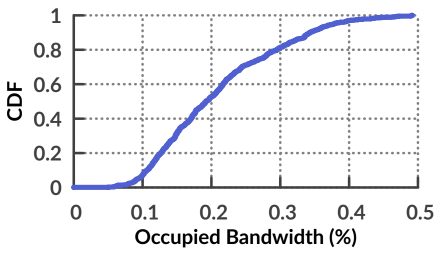


Figure 19: Occupied bandwidth.

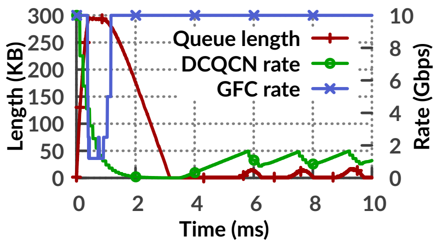


Figure 20: Interaction with DCQCN.

are randomly selected, it takes time to generate the specific flow combinations to fill up CBD under PFC. During 0 ∼ 8.5ms, both flow control mechanisms work well. However, at 8.5ms, deadlock occurs under PFC. Then the throughput of the entire network rapidly goes to zero. Three main reasons lead to this scenario: (1) The appearance of deadlock prohibits the forwarding of all flows associated with the CBD. (2) Deadlock pressures congestion back to sources, further making many victim flows cease. (3) Although many irrelevant servers can still send packets when deadlock occurs, they will randomly select new destinations after current flows finishing. Once the new forwarding path passes through any "dead" links, this server will also be paused and more links will become "dead". Through a transient evolution (from 8.5ms to 8.7ms), the entire network traps into deadlock ultimately. By employing buffer-based GFC, the sending rate can always be controlled well and no deadlock occurs. The results of other network scales and of time-based GFC are similar.

Occupied bandwidth: We provide further evaluations about the consumed bandwidth by employing GFC. In time-based GFC, Message Generator remains unchanged, so the occupied bandwidth keeps the same as in CBFC. In buffer-based GFC, Message Generator would generate more feedback messages, which is suspicious to damage available bandwidth. So this part mainly evaluates the introduced bandwidth consumption of buffer-based GFC. We add a counting module on each port to record the number of received feedback messages every 500μs. The network topology is fat-tree ($k = 16$) and traffic pattern is randomly generated as in the former set of simulations. We conduct 100 simulations and record occupied bandwidth of all ports. The CDF of results is presented in Figure 19. The average bandwidth consumption is 0.21%, and in 99% cases the occupied bandwidth is less than 0.4%. Frequent generations of feedback messages occur only when the sending rate changes dramatically (e.g., 9Gbps → 1Gbps), which happens rarely. In our simulations, only up to 0.49% of bandwidth occupation is observed.

7 DISCUSSION

Cooperation with congestion control. In practical lossless networks, both flow control and congestion control are deployed to manage network traffic. Existing congestion control mechanisms tend to explicitly (e.g., QCN [26] and DCQCN [59]) or implicitly (e.g., TIMELY [42]) control the queue length to a low level, so low latency and high throughput can be achieved at the same time. Since GFC only decreases the sending rate when queue length is very large, the introduction of GFC would not change the steady-state results of congestion control. The duty of GFC is avoiding packet loss and deadlock in critical situations. In common scenarios, congestion control is taking charge of managing proper network sharing and keeping the queue occupation below the active threshold of GFC.

To better understand the interaction between GFC and congestion control, we build a small simulation using the dumbbell topology, which contains 8 senders ($H1 \sim H8$) and 1 receiver ($H9$). All settings of buffer-based GFC are consistent with aforementioned simulations. DCQCN is implemented as a representative congestion control. The ECN threshold is 40KB. $\alpha = 0.5$, $g = 1/256$, $N = 50\mu s$ and $K = 55\mu s$ in DCQCN. All senders start a long flow to the receiver at the same time. The results are shown in Figure 20. We monitor three metrics: (1) the ingress queue length of switch port connected with $H1$, (2) the flow sending rate in $H1$, which is controlled by DCQCN (denoted by "DCQCN rate"), and (3) the rate of output queue in $H1$, which is controlled by buffer-based GFC (denoted by "GFC rate"). Notice that, in this scenario, the minimum value of (2) and (3) will determine the final sending rate of $H1$. At the beginning, the input queue length increases quickly owing to the incast traffic. Although DCQCN detects the congestion, it needs several turns to decrease the flow rate. Therefore, queue length keeps rising and triggers GFC. GFC rapidly limits the output rate of $H1$ to 1.25Gbps, so the queue length stops increasing further. Then DCQCN continuously decreases the sending rate to remove backlog. Once the DCQCN rate is smaller than GFC rate, the sending rate of $H1$ is fully controlled by DCQCN and GFC is actually disabled. Then DCQCN processes the normal evolution to find the appropriate flow rate. This simulation shows that GFC only works as a safeguard and introduces small influence on the overall evolution of end-to-end congestion control.

Multiple priority queues. In CEE, different priorities usually represent the precedence for occupying bandwidth. In order to prevent low priority queues from long-time starvation, which may introduce the risk of buffer exhausting, the output queue scheduling should be enabled to assign minimal output bandwidth to each priority. The weighted output queue scheduling is available on most commodity switches [11, 34]. As long as the queue obtains a positive output rate, GFC can adjust the input rate to match it. In InfiniBand, different priority queues fairly share the output bandwidth, so GFC can work as expected without additional configurations.

Routing loop. Since GFC keeps all routing configurations unmodified, routing loop, if exists, would trap packets in itself and make the sending rate gradually decreases to zero. However, eliminating self-loop is the basic requirement of network routing and there are many efficient mechanisms deployed in data centers to achieve it. In layer-2 network, spanning tree protocol is widely deployed to efficiently avoid the generation of self-loop. Furthermore, novel mechanisms are proposed when designing new data center architectures [44] and efficient rerouting mechanism is designed to guarantee loop-freedom during rule update [5]. Therefore, although routing loop can formulate deadlock, it is, to the best of our knowledge, not observed in real-world deployment. Practical deadlocks are formulated by a combination of flows each passing through part of the CBD [22, 52]. This scenario can be well handled by GFC.

The granularity of rate limiting. Owing to the implementation constraint, the granularity of rate limiting cannot be infinitely small. When the target sending rate is smaller than the minimum rate unit, GFC needs to alternate sending rate between zero and the minimum rate unit to match it, which may still introduce *hold and wait*. However, in commodity switches, the minimum rate unit is 8Kbps [12, 33], which is small enough in general scenarios. For example, in 10/40/100Gbps network, it requires at least 6 cascaded 16:1 incast in the same priority to force the sending rate to be less than 8Kbps (0.59/2.38/5.96 Kbps), which hardly happens in practice.

Lossless Ethernet vs lossy Ethernet. Considering many side-effects (e.g., head-of-line blocking, victim flow and deadlock) caused by guaranteeing zero packet loss (employing PFC), there is a trend to build RMDA over lossy network. IRN is the representative of this work [43]. By implementing a smarter retransmission mechanism, the performance of long flows would not be penalized a lot by packet loss. However, for short flows, which are widespread in data centers, any packet loss would double their FCT. Furthermore, in some services (e.g., IPC), zero packet loss is a critical requirement. Any packet retransmission would make a flow miss its deadline. As mentioned in [43], IRN experiences significant packet loss (8.5%) without PFC, thus cannot support these services well. Therefore, zero packet loss on layer-2 network is still an attractive property for optimizing network performance.

8 RELATED WORK

CBD-free routing. To avoid network deadlock, previous work focuses on designing central routing algorithms to avoid CBD occurrence under any traffic pattern [7, 13–15, 17, 18, 21, 50, 51, 54–56, 58]. Autonet [51] proposes Up*/Down* routing, which can eliminate CBD in tree-like topologies. Then, more methodologies are designed to guarantee CBD-free in other generic topologies. However, these solutions introduce great limitations to network configurations and disable some intrinsic advantages (e.g., balancing load among multiple paths). Furthermore, completely obeying the centralized routing decisions at all time is impractical. When link failure occurs in network, distributed rerouting would be activated to maintain network connectivity, which still introduces the risk of generating CBD.

Other work presents dynamic-routing-based mechanisms to avoid CBD [16]. In normal situations, packets can be forwarded through the optimal paths. When buffers are full, preset escape paths are activated to make sure no CBD exists. The implementation requires switches to reserve an independent buffer for each escape path, which would consume a large amount of precious storage resource. This defence mechanism is conservative since it will activate escape paths even when the normal congestion fills up buffer. However, forwarding packets through escape (non-optimal) paths may greatly deteriorate the network performance.

Queue management. It has been proved that by arranging multiple independent priority queues in each switch and increasing packet priority hop by hop, deadlock can be fully eliminated [6, 20, 35]. The basic idea is to guarantee no CBD exists in the same priority class, and similar queue management strategy [40] is also employed in InfiniBand. However, the number of required priority queues increases with the network scale. These solutions face great challenge in scalability since commodity switches can only support several priorities.

Tagger [25] shares the same principle and further leverages the structure feature of topologies. It increases a packet's priority only when the routing rule is risky of generating CBD. So it can decrease the number of required priority queues. However, when the number of priorities is not enough, Tagger will directly drop corresponding packets, which damages the lossless property.

Deadlock recovery. Deadlock recovery solutions [2, 3, 36, 38, 48, 52] mainly contain two components: (1) Interacting status information between switches to heuristically discover the occurrence of deadlock. (2) Dropping or temporarily rerouting some packets involved in CBD to recover deadlock. SPIN recently proposes the idea of synchronously draining packets on the CBD to recover from deadlock faster [48]. However, recovery solutions all work reactively, thus cannot solve the root cause of deadlock and cannot prevent its reappearance.

9 CONCLUSION

This paper understands and solves the network deadlock from a brand-new perspective: avoiding *hold and wait* condition. We propose GFC to control the sending rate at a fine granularity. Therefore the sending and draining rates could match well without triggering *hold and wait* and further no deadlock. We theoretically demonstrate that our design can completely avoid deadlock, and then elaborate how to implement GFC on the basis of mainstream lossless networks with moderate modifications. Both testbed experiments and packet-level simulations are conducted to evaluate the performance. Results confirm that GFC can effectively avoid deadlock with negligible side-effects.

ACKNOWLEDGMENTS

The authors gratefully acknowledge the shepherd Kai Chen and the anonymous reviewers for their constructive comments. This work is supported in part by the National Key Research and Development Program of China (No.2018YFB1700103), and by National Natural Science Foundation of China (NSFC) under Grant 61872208.

REFERENCES

[1] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. 2008. A Scalable, Commodity Data Center Network Architecture. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication (SIGCOMM '08)*. ACM, New York, NY, USA, 63–74.

[2] K. V. Anjan and Timothy Mark Pinkston. 1995. An Efficient, Fully Adaptive Deadlock Recovery Scheme: DISHA. In *Proceedings of the 22Nd Annual International Symposium on Computer Architecture (ISCA '95)*. ACM, New York, NY, USA, 201–210.

[3] K. V. Anjan, Timothy Mark Pinkston, and José Duato. 1996. Generalized theory for deadlock-free adaptive wormhole routing and its application to DISHA concurrent. In *Proceedings of International Conference on Parallel Processing (ICPP '96)*. 815–821.

[4] InfiniBand Trade Association. 2015. InfiniBand architecture specification: release 1.3. (2015). https://www.infinibandta.org/ibta-specifications-download/

[5] Arsany Basta, Andreas Blenk, Szymon Dudycz, Arne Ludwig, Stefan Schmid, Stefan Schmid, Szymon Dudycz, Andreas Blenk, Arne Ludwig, and Arsany Basta. 2018. Efficient loop-free rerouting of multiple SDN flows. *IEEE/ACM Transactions on Networking (TON)* 26, 2 (2018), 948–961.

[6] Dimitri P Bertsekas, Robert G Gallager, and Pierre Humblet. 1992. *Data networks*. Vol. 2. Prentice-Hall International New Jersey.

[7] Jacek Blazewicz, Daniel P. Bovet, Jerzy Brzezinski, Giorgio Gambosi, and Maurizio Talamo. 1994. Optimal centralized algorithms for store-and-forward deadlock avoidance. *IEEE Trans. Comput.* 43, 11 (1994), 1333–1338.

[8] BroadCom. 2004. 10GBASE-T Coding and Modulation: 128-DSQ + LDPC. (2004). http://www.ieee802.org/3/an/public/sep04/ungerboeck_2_0904.pdf

[9] Adrian M. Caulfield and Steven Swanson. 2013. QuickSAN: A Storage Area Network for Fast, Distributed, Solid State Disks. In *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA '13)*. ACM, New York, NY, USA, 464–474.

[10] Cisco. 2015. Priority Flow Control: Build Reliable Layer 2 Infrastructure. (2015). https://www.cisco.com/c/en/us/products/collateral/switches/nexus-7000-series-switches/white_paper_c11-542809.pdf

[11] Cisco. 2016. Cisco Nexus 3000 Series Switches. (2016). https://www.cisco.com/c/en/us/products/switches/nexus-3000-series-switches/index.html

[12] Cisco. 2019. rate-limit-interface. (2019). https://www.oreilly.com/library/view/cisco-ios-in/0596008694/re712.html

[13] William J. Dally and Hiromichi Aoki. 1993. Deadlock-free adaptive routing in multicomputer networks using virtual channels. *IEEE Transactions on Parallel and Distributed Systems* 4, 4 (1993), 466–475.

[14] William J. Dally and Charles L. Seitz. 1988. Deadlock-free message routing in multiprocessor interconnection networks. (1988).

[15] Jens Domke, Torsten Hoefler, and Wolfgang E Nagel. 2011. Deadlock-free oblivious routing for arbitrary topologies. In *2011 IEEE International Parallel Distributed Processing Symposium (IPDPS '11)*. 616–627.

[16] Jose Duato. 1993. A new theory of deadlock-free adaptive routing in wormhole networks. *IEEE Transactions on Parallel and Distributed Systems* 4, 12 (1993), 1320–1331.

[17] Jose Duato and Timothy Mark Pinkston. 2001. A general theory for deadlock-free adaptive routing using a mixed set of resources. *IEEE Transactions on Parallel and Distributed Systems* 12, 12 (2001), 1219–1235.

[18] Jose Flich, Tor Skeie, Andres Mejia, Olav Lysne, Pedro Lopez, Antonio Robles, Jose Duato, Michihiro Koibuchi, Tomas Rokicki, and Jose Carlos Sancho. 2012. A survey and evaluation of topology-agnostic deterministic routing algorithms. *IEEE Transactions on Parallel and Distributed Systems* 23, 3 (2012), 405–425.

[19] Marina Garcia, Enrique Vallejo, Ramon Beivide, Miguel Odriozola, Cristobal Camarero, Mateo Valero, Jesús Labarta, and Cyriel Minkenberg. 2012. On-the-fly adaptive routing in high-radix hierarchical networks. In *2012 41st International Conference on Parallel Processing (ICPP '12)*. IEEE, 279–288.

[20] Mario Gerla and Leonard Kleinrock. 1980. Flow control: A comparative survey. *IEEE Transactions on Communications* 28, 4 (1980), 553–574.

[21] Christopher J. Glass and Lionel M. Ni. 1992. The Turn Model for Adaptive Routing. In *Proceedings of the 19th Annual International Symposium on Computer Architecture (ISCA '92)*. ACM, New York, NY, USA, 278–287.

[22] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. 2016. RDMA over Commodity Ethernet at Scale. In *Proceedings of the 2016 ACM SIGCOMM Conference (SIGCOMM '16)*. ACM, New York, NY, USA, 202–215.

[23] Peter Hoose. 2015. Managing, monitoring and troubleshooting large scale networks. (2015). http://ces-nanog64.blogspot.com/2015/06/monitoring-managing-and-troubleshooting.html

[24] Shuihai Hu, Yibo Zhu, Peng Cheng, Chuanxiong Guo, Kun Tan, Jitendra Padhye, and Kai Chen. 2016. 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 (HotNets '16)*. ACM, New York, NY, USA, 92–98.

[25] Shuihai Hu, Yibo Zhu, Peng Cheng, Chuanxiong Guo, Kun Tan, Jitendra Padhye, and Kai Chen. 2017. Tagger: Practical PFC Deadlock Prevention in Data Center Networks. In *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT '17)*. ACM, New York, NY, USA, 451–463.

[26] IEEE. 2010. 802.1Qau - Congestion Notification. (2010). https://1.ieee802.org/dcb/802-1qau/

[27] IEEE. 2010. IEEE802.1 Qbb. (2010). https://1.ieee802.org/dcb/802-1qbb/

[28] IEEE. 2013. Data Center Bridging Task Group. (2013). http://www.ieee802.org/1/pages/dcbridges.html

[29] INCITS. 2019. T11 Home Page. (2019). http://www.t11.org/index.html

[30] Intel. 2019. DPDK. (2019). http://dpdk.org/

[31] Intel. 2019. Test-pipeline. (2019). https://github.com/DPDK/dpdk/tree/master/app/test-pipeline

[32] Xin Jin, Hongqiang Harry Liu, Rohan Gandhi, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Jennifer Rexford, and Roger Wattenhofer. 2014. Dynamic Scheduling of Network Updates. In *Proceedings of the 2014 ACM Conference on SIGCOMM (SIGCOMM '14)*. ACM, New York, NY, USA, 539–550.

[33] Juniper. 2019. Bandwidth-limit (Policer). (2019). https://www.juniper.net/documentation/en_US/junos/topics/reference/configuration-statement/bandwidth-limit-edit-firewall-policer.html

[34] Juniper. 2019. Class of Service Feature Guide for EX Series Switches (Except EX4600 and EX9200 Switches). (2019). https://www.juniper.net/documentation/en_US/junos/information-products/pathway-pages/ex-series/cos-ex-series.html

[35] Mark Karol, S. Jamaloddin Golestani, and David Lee. 2003. Prevention of deadlocks and livelocks in lossless backpressured packet networks. *IEEE/ACM Transactions on Networking* 11, 6 (2003), 923–934.

[36] Pedro Lopez, Juan Miguel Martínez, and Jose Duato. 1998. A very efficient distributed deadlock detection mechanism for wormhole networks. In *Proceedings 1998 Fourth International Symposium on High-Performance Computer Architecture*. 57–66.

[37] David Maltz. 2016. Keeping Cloud-Scale Networks Healthy. (2016). https://video.mtgsf.com/video/4f277939-73f5-4ce8-aba1-3da70ec19345

[38] Juan Miguel Martínez, Pedro Lopez, José Duato, and Timothy Mark Pinkston. 1997. Software-based deadlock recovery technique for true fully adaptive routing in wormhole networks. In *Proceedings of the 1997 International Conference on Parallel Processing (ICPP '97)*. 182–189.

[39] Mellanox. 2019. 4th Generation Server & Storage Adapter Architecture. (2019). http://www.mellanox.com/related-docs/prod_architecture/PB_ConnectX_Architecture_Brochure.pdf

[40] Mellanox. 2019. InfiniBand White paper. (2019). http://www.mellanox.com/page/white_papers

[41] Mellanox. 2019. Interconnect Your Future. (2019). https://www.mellanox.com/related-docs/solutions/hpc/TOP500-January-2019.pdf

[42] Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. 2015. TIMELY: RTT-based Congestion Control for the Datacenter. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15)*. ACM, New York, NY, USA, 537–550.

[43] Radhika Mittal, Alexander Shpiner, Aurojit Panda, Eitan Zahavi, Arvind Krishnamurthy, Sylvia Ratnasamy, and Scott Shenker. 2018. Revisiting Network Support for RDMA. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '18)*. ACM, New York, NY, USA, 313–326.

[44] Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, and Amin Vahdat. 2009. PortLand: A Scalable Fault-tolerant Layer 2 Data Center Network Fabric. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication (SIGCOMM '09)*. ACM, New York, NY, USA, 39–50.

[45] OMNET++. 2019. OMNET++ homepage. (2019). https://www.omnetpp.org/

[46] Valentin Puente, Ramón Beivide, José A Gregorio, JM Prellezo, Jose Duato, and Cruz Izu. 1999. Adaptive bubble router: a design to improve performance in torus networks. In *Proceedings of the 1999 International Conference on Parallel Processing (ICPP '99)*. 58–67.

[47] Sivasankar Radhakrishnan, Yilong Geng, Vimalkumar Jeyakumar, Abdul Kabbani, George Porter, and Amin Vahdat. 2014. SENIC: Scalable NIC for End-Host Rate Limiting. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI '14)*. 475–488.

[48] Aniruddh Ramrakhyani, Paul V. Gratz, and Tushar Krishna. 2018. Synchronized progress in interconnection networks (spin): A new theory for deadlock freedom. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA '13)*. IEEE, 699–711.

[49] Sven-Arne Reinemo, Tor Skeie, and Manoj K Wadekar. 2010. Ethernet for high-performance data centers: On the new IEEE datacenter bridging standards. *IEEE Micro* 30, 4 (2010), 42–51.

[50] Jose Carlos Sancho, Antonio Robles, and Jose Duato. 2004. An effective methodology to improve the performance of the up*/down* routing algorithm. *IEEE Transactions on Parallel and Distributed Systems* 15, 8 (2004), 740–754.

[51] Michael D. Schroeder, Andrew D. Birrell, Michael Burrows, Hal Murray, Roger M. Needham, Thomas L. Rodeheffer, Edwin H. Satterthwaite, and Charles P. Thacker. 1991. Autonet: A high-speed, self-configuring local area network using point-to-point links. *IEEE Journal on Selected Areas in Communications* 9, 8 (1991), 1318–1335.

[52] Alex Shpiner, Eitan Zahavi, Vladimir Zdornov, Tal Anker, and Matty Kadosh. 2016. Unlocking Credit Loop Deadlocks. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks (HotNets '16)*. ACM, New York, NY, USA, 85–91.

[53] Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. 2014. *Operating system concepts essentials*. John Wiley & Sons, Inc.

[54] Tor Skeie, Olav Lysne, and Ingebjørg Theiss. 2002. Layered Shortest Path (LASH) Routing in Irregular System Area Networks. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium (IPDPS '02)*. IEEE Computer Society, 194.
[55] Brent Stephens and Alan L. Cox. 2016. Deadlock-free local fast failover for arbitrary data center networks. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications (INFOCOM '16)*. IEEE, 1–9.
[56] Brent Stephens, Alan L. Cox, Ankit Singla, John Carter, Colin Dixon, and Wesley Felter. 2014. Practical DCB for improved data center networks. In *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications (INFOCOM '14)*. IEEE, 1824–1832.
[57] Erico Vanini, Rong Pan, Mohammad Alizadeh, Parvin Taheri, and Tom Edsall. 2017. Let It Flow: Resilient Asymmetric Load Balancing with Flowlet Switching. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI '17)*. USENIX Association, Boston, MA, 407–420.
[58] Jie Wu. 2003. A fault-tolerant and deadlock-free routing protocol in 2D meshes based on odd-even turn model. *IEEE Trans. Comput.* 52, 9 (2003), 1154–1169.
[59] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. 2015. Congestion Control for Large-Scale RDMA Deployments. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15)*. ACM, New York, NY, USA, 523–536.
[60] Yibo Zhu, Nanxi Kang, Jiaxin Cao, Albert Greenberg, Guohan Lu, Ratul Mahajan, Dave Maltz, Lihua Yuan, Ming Zhang, Ben Y. Zhao, and Haitao Zheng. 2015. Packet-Level Telemetry in Large Datacenter Networks. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15)*. ACM, New York, NY, USA, 479–491.

Appendices are supporting material that has not been peer reviewed.

A PROOF OF THEOREM 4.1

The straightforward idea to eliminating *hold and wait* is working out the worst case of queue length $q(t)$ and ensuring it never exceeds B_m (i.e., $q_{max} < B_m$). First, we provide the general evolution of queue length $q(t)$ under congestion (i.e., $R_i(t) > R_d(t)$). In congestion situation, $q(t)$ will achieves its maximum $q_{\max}$ only when the input rate $R_i(t)$ continuously decreases to the draining rate $R_d(t)$. Considering the general buffer evolution depicted in Figure 21. Supposing q_{max} is approached at t_b, we focus on the continuous increase of $q(t)$ starting from t_a and ending at t_b. The starting instant t_a is selected according to the following criteria: (1) During (t_a, t_b), $q(t)$ increases monotonically, (2) $q(t_a) = B_0$ or $q(t_a - \delta) \geq q(t_a)$, here $\delta \to 0^+$. We consider the time sequence $\{t_k = k\tau + t_0 | k \in [0, n]\}$, where $t_a \leq t_0 < t_a + \tau$ and $t_n = t_b$, in this process. Accordingly, we have

$$\begin{cases} q(t_0) \geq B_0 \\ q(t_{n-1}) = B_m - \frac{R_d}{C}(B_m - B_0) \\ q(t_n) = q_{\max} \end{cases} \tag{7}$$

And for any instant $t \in [t_k, t_{k+1}]$ $(k = 1, \cdots, n-1)$,

$$R_i(t) \leq R_i(t_k) = R_t(t_{k-1}) = \frac{B_m - q(t_{k-1})}{B_m - B_0} C$$

Then the queue increase can be represent as

$$q(t_{k+1}) - q(t_k) \leq \frac{B_m - q(t_{k-1})}{B_m - B_0} C\tau - \int_{k\tau}^{(k+1)\tau} R_d(t)dt \tag{8}$$

Combining (7) and (8), we obtain a series of difference inequations with terminal condition.

Defining $l = \frac{B_m - B_0}{C\tau}$, the condition $B_0 \leq B_m - 4C\tau$ is equivalent to $l \geq 4$. Inequation (8) is rewritten as

$$q(t_{k+1}) - q(t_k) \leq \frac{B_m - q(t_{k-1})}{l} - \int_{k\tau}^{(k+1)\tau} R_d(t)dt \tag{9}$$

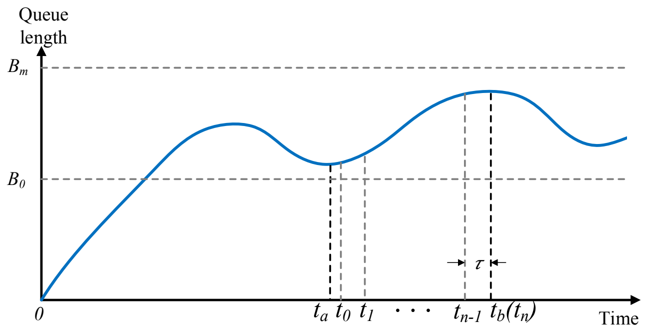


Figure 21: Queue length evolution.

Since R_d is never negative, the last item in (9) can be removed,

$$q(t_{k+1}) - q(t_k) \leq \frac{B_m - q(t_{k-1})}{l} \tag{10}$$

Adding several items in (10), we have

$$q(t_{k+1}) - aq(t_k) - bB_m \leq c(q(t_k) - aq(t_{k-1}) - bB_m) \tag{11}$$

here the coefficients a, b and c satisfy

$$\begin{cases} a + c = 1 \\ a * c = \frac{1}{l} \\ b - b * c = \frac{1}{l} \end{cases} \tag{12}$$

As $l \geq 4$, (12) has real solutions, i.e.,

$$\begin{aligned} a &= \frac{1 \pm \sqrt{1 - \frac{4}{l}}}{2} \\ b &= \frac{1 \mp \sqrt{1 - \frac{4}{l}}}{2} = 1 - a \\ c &= \frac{1 \mp \sqrt{1 - \frac{4}{l}}}{2} \end{aligned}$$

Thus a, b and c are all positive. (11) can be expanded as

$$q(t_{k+1}) - aq(t_k) - bB_m \leq c^k(q(t_1) - aq(t_0) - bB_m) \tag{13}$$

Defining $Q_0 = q(t_1) - aq(t_0) - bB_m$ and setting $k = n - 1$, (13) is equivalent to

$$\begin{aligned} q(t_n) &\leq aq(t_{n-1}) + bB_m + c^{n-1}Q_0 \\ &= a(B_m - \frac{R_d}{C}(B_m - B_0)) + bB_m + c^{n-1}Q_0 \\ &\leq B_m + c^{n-1}Q_0 \end{aligned}$$

That is, once $Q_0 < 0$, we can obtain $q_{\max} < B_m$. Subsequently, we proof $Q_0 < 0$.

Setting $k = 0$, we get the following inequation through expanding (10).

$$q(t_1) - q(t_0) \leq \frac{B_m - q(t_a)}{l} \tag{14}$$

Since $t_a - t_0 < \tau$ and $R_i(t_a)$ is the local maximum, thus

$$q(t_0) \leq q(t_a) + \frac{B_m - q(t_a)}{l} \tag{15}$$

Taking (14) and (15) into the definition of Q_0, we have

$$\begin{aligned} Q_0 &= q(t_1) - aq(t_0) - bB_m \\ &= (q(t_1) - q(t_0)) + (1-a)(q(t_0) - B_m) \\ &\le \tfrac{B_m - q(t_a)}{l} + (1-a)(q(t_a) + \tfrac{B_m - q(t_a)}{l} - B_m) \\ &= (1 + (1-a)(1-l))\tfrac{B_m - q(t_a)}{l} \end{aligned} \quad (16)$$

According to (12), $l = \frac{1}{a(1-a)}$ and then the coefficient item in (16) satisfies

$$\begin{aligned} 1 + (1-a)(1-l) &= 1 + (1-a)(1 - \tfrac{1}{a(1-a)}) \\ &= 2 - a - \tfrac{1}{a} < 0 \end{aligned}$$

where the last less-than sign comes from $0 < a < 1$. Thus, we have $Q_0 < 0$ and then $q_{\max} < B_m$. Combining with the mapping function, it means that input rate never goes to zero, namely *hold and wait* is eliminated in conceptual GFC.

B PROOF OF THEOREM 5.1

Generally, during the period $[kT, (k+1)T]$, the real input rate satisfies

$$R_i(t) = \begin{cases} \frac{B_m - q_{k-1}}{B_m - B_0}C, & t < KT + \tau \\ \frac{B_m - q_k}{B_m - B_0}C, & t \ge KT + \tau \end{cases}$$

Thus, the variation of queue length during $[kT, (k+1)T]$ is

$$q_{k+1} - q_k = \frac{B_m - q_{k-1}}{B_m - B_0}C\tau + \frac{B_m - q_k}{B_m - B_0}C(T-\tau) - \int_{KT}^{(K+1)T} R_d(t)dt \quad (17)$$

Similar to the proof of Theorem 4.1, we proof $q_{\max} < B_m$ in this scenario. In congestion situation, $q(t)$ achieves its maximum $q_{\max}$ only when $R_i(t)$ continuously decreases below $R_d(t)$. Supposing q_{max} is achieved at t_b, we focus on the continuous increase of $q(t)$ starting from t_a and ending at t_b. The starting instant t_a is selected according to the following criteria: (1) During (t_a, t_b), $q(t)$ increases monotonically, (2) $q(t_a) = B_0$ or $q(t_a - \delta) \ge q(t_a)$, here $\delta \to 0^+$. We consider the time sequence $\{t_k = kT + t_0 | k \in [0, n]\}$, here $t_a \le t_0 < t_a + T$ and $t_n = t_b$, in this process. Thus, there is

$$\begin{cases} q(t_{n-1}) \le B_m - \frac{R_d}{C}(B_m - B_0) \\ q(t_n) = q_{\max} \ge B_m - \frac{R_d}{C}(B_m - B_0) \end{cases}$$

Subsequently, we proof $q_{\max} < B_m$.

Defining $l = \frac{B_m - B_0}{CT}$, thus the condition

$$B_m - B_0 \ge \left(\sqrt{\frac{\tau}{T}} + 1\right)^2 CT$$

is equivalent to $l \ge \left(\sqrt{\frac{\tau}{T}} + 1\right)^2$. Equation (17) is rewritten as

$$q(t_{k+1}) - q(t_k) = \frac{B_m - q(t_{k-1})}{l}\frac{\tau}{T} + \frac{B_m - q(t_k)}{l}\frac{T-\tau}{T} - \int_{kT}^{(k+1)T} R_d(t)dt \quad (18)$$

Since $R_d(t)$ is never negative, the last item in (18) can be removed,

$$q(t_{k+1}) - q(t_k) \le \frac{B_m - q(t_{k-1})}{l}\frac{\tau}{T} + \frac{B_m - q(t_k)}{l}\frac{T-\tau}{T} \quad (19)$$

Adding several items in (19), we have

$$q(t_{k+1}) - aq(t_k) - bB_m \le c(q(t_k) - aq(t_{k-1}) - bB_m) \quad (20)$$

here the coefficients a, b, c satisfy

$$\begin{cases} a + c = 1 - \frac{T-\tau}{lT} \\ a * c = \frac{\tau}{lT} \\ b - b * c = \frac{1}{l} \end{cases} \quad (21)$$

When $l \ge (\sqrt{\frac{\tau}{T}} + 1)^2$, (21) has real solutions:

$$\begin{aligned} a &= \frac{1 - \frac{T-\tau}{lT} \pm \sqrt{(1 - \frac{T-\tau}{lT})^2 - \frac{4\tau}{lT}}}{2} \\ b &= \frac{1 + \frac{T-\tau}{lT} \mp \sqrt{(1 - \frac{T-\tau}{lT})^2 - \frac{4\tau}{lT}}}{2} = 1 - a \\ c &= \frac{1 - \frac{T-\tau}{lT} \mp \sqrt{(1 - \frac{T-\tau}{lT})^2 - \frac{4\tau}{lT}}}{2} \end{aligned}$$

Thus a, b, c are all positive. We can expand (20) as

$$q(t_{k+1}) - aq(t_k) - bB_m \le c^k(q(t_1) - aq(t_0) - bB_m) \quad (22)$$

Defining $Q_0 = q(t_1) - aq(t_0) - bB_m$ and setting $k = n-1$, (22) is equivalent to

$$\begin{aligned} q(t_n) &\le aq(t_{n-1}) + bB_m + c^{n-1}Q_0 \\ &\le a(B_m - \tfrac{O}{C}(B_m - B_0)) + bB_m + c^{n-1}Q_0 \\ &\le B_m + c^{n-1}Q_0 \end{aligned}$$

That is, once $Q_0 < 0$, we can obtain $q_{\max} < B_m$.

We can expand (19) by setting $k = 0$,

$$q(t_1) - q(t_0) \le \frac{B_m - q(t_a)}{l}\frac{\tau}{T} + \frac{B_m - q(t_0)}{l}\frac{T-\tau}{T} \quad (23)$$

Since $t_a - t_0 < \tau$ and $R_i(t_a)$ is the local maximum, thus

$$q(t_0) \le q(t_a) + \frac{B_m - q(t_a)}{l} \quad (24)$$

Taking (23) and (24) into the definition of Q_0, we have

$$\begin{aligned} Q_0 &= q(t_1) - aq(t_0) - bB_m \\ &= (q(t_1) - q(t_0)) + (1-a)(q(t_0) - B_m) \\ &\le \tfrac{B_m - q(t_a)}{l}\tfrac{\tau}{T} + (-\tfrac{T-\tau}{lT} + 1 - a)(q(t_0) - B_m) \\ &\le (\tfrac{\tau}{T} + (-\tfrac{T-\tau}{lT} + 1 - a)(1-l))\tfrac{B_m - q(t_a)}{l} \end{aligned} \quad (25)$$

According to (21), $l = \frac{1}{(1-a)(a + \frac{T-\tau}{lT})}$ and then the coefficient item in (25) satisfies

$$\begin{aligned} &\tfrac{\tau}{T} + (-\tfrac{T-\tau}{lT} + 1 - a)(1-l) \\ =\ &\tfrac{\tau}{T} - \tfrac{T-\tau}{lT}(1-l) + (1-a)(1 - \tfrac{1}{(1-a)(a + \frac{T-\tau}{lT})}) \\ =\ &1 - \tfrac{T-\tau}{lT} + 1 - a - \tfrac{1}{a + \frac{T-\tau}{lT}} \\ =\ &2 - (a + \tfrac{T-\tau}{lT}) - \tfrac{1}{a + \frac{T-\tau}{lT}} < 0 \end{aligned} \quad (26)$$

here the last less-than sign comes from $0 < a + \frac{T-\tau}{lT} < 1$. Thus, we have $Q_0 < 0$ and then $q_{\max} < B_m$. Combining with the mapping function, it means that input rate never reaches zero, namely *hold and wait* is eliminated in time-based GFC.