TTL-based Buffer Management Scheme for Deadlock Prevention

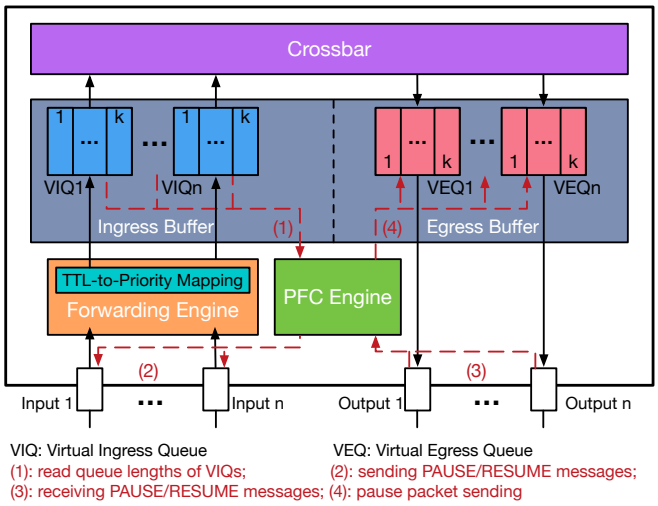

Figure 1: Switch model.

1. Solution

1.1 Switch Model

In this part, we introduce the switch model under which we develop our TTL-based buffer management scheme. As shown in Fig. 1, our switch model consists of five elements: switch port, forwarding engine, PFC engine, crossbar and switch buffer. The function of these elements are list as follows.

- **Switch port**: The element to receive and send packets. Each switch port consists of an input port and an output port, and is connected with a link.
- **Forwarding engine**: The element to make the routing and/or switching decisions. In the forwarding engine, there is a module to decide the priority class of a packet based on the TTL value.
- **PFC engine**: The element to perform PFC function.
- **Crossbar**: The element to switch packet cells from the ingress buffer to the egress buffer.
- **Switch buffer**: The element to buffer incoming packets. We assume shared buffer in our model. In the ingress buffer, we use virtual ingress queue (VIQ) to track the incoming packets. In the egress buffer, we use virtual egress queue (VEQ) to track the outgoing packets. Each VIQ is uniquely corresponding to an input port, while each VEQ is uniquely corresponding to an output port. As shown in Fig. 1, both VIQ and VEQ consist of k queues, with each queue corresponding to a priority class. A packet of priority class j will enter the j-th queue of its destined VIQ and VEQ.

  Discussion: It is not always the case for switch to have both ingress buffer and egress buffer at the same time. It depends on the buffering strategy. If we adopt output-queued strategy, there will be no ingress buffer, and VIQs are simply some counters to record the bytes of buffered packets received by different input ports. If we adopt input-queued strategy [5], there will be no egress buffer, and VEQs can be simulated using virtual output queuing technique [6] at the ingress buffer. If we adopt the combined-input-and-output-queued strategy [3], the switch will have both ingress buffer and egress buffer. The switching function of crossbar can also be virtual, not necessary to involve packet copying.

Enqueue and dequeue operations of VIQ and EIQ: We use q_{in}^{i} to denote VIQ i, and q_{out}^{i} to denote VEQ i. Let $q_{in}^{i,j}$ be the j-th queue of VIQ i, and $q_{out}^{i,j}$ be the j-th queue of VEQ i ($1 \leq i \leq n$, $1 \leq j \leq k$). The dequeue and enqueue operatons of VIQ and VEQ are performed as follows:

1. Packet p of priority class j enters the switch buffer via input port i_1: VIQ i_1 performs enqueue operation and puts packet p to the tail of $q_{in}^{i_1,j}$.
2. Packet p in the head of $q_{in}^{i_1,j}$ is forwarded to VEQ i_2: VEQ i_2 performs enqueue operation and puts packet p to the tail of $q_{out}^{i_2,j}$. At the same time, head point of $q_{in}^{i_1,j}$ is moved to the next packet currently queued in $q_{in}^{i_1,j}$. Note that dequeue operation of packet p will not be performed at $q_{in}^{i_1,j}$ at this point in time.
3. Packet p in the head of $q_{out}^{i_2,j}$ is forwarded to output port i_2: VEQ i_2 performs dequeue operation and removes packet p from $q_{out}^{i_2,j}$. VIQ i_1 performs dequeue operation and removes packet p from $q_{in}^{i_1,j}$ (say packet p was previously forwarded from $q_{in}^{i_1,j}$ to $q_{out}^{i_2,j}$).

Note that in our switch model, dequeue operation is performed at corresponding VIQ only after packet p leaves the switch buffer. This is because the purpose of using VIQ is to track the bytes of currently buffered packets received by an input port, such that PFC can work properly.

1.2 Modified PFC Mechanism

As shown in Fig. 1(b), the function of our modified PFC mechanism consists of four parts.

The PFC engine will track the instant queue lengths of all the VIQs (function (1)). If the queue length of some queue $q_{in}^{i,j}$ exceeds the configured PFC threshold, PFC engine will generate a PAUSE message to pause the packet transmission of the immediate upstream node on priority class j-1 over the link connected with port i. If the queue length of $q_{in}^{i,j}$ becomes less than the threshold later, PFC engine will generate a RESUME message to resume the packet transmission over the previously paused link on priority class j-1 (function (2)).

If some output port, say output port i, receives a PAUSE or RESUME message on priority class j from its immediate downstream node, the PFC engine will then pause or resume the corresponding egress queue $q_{out}^{i,j}$, accordingly (function (3) and function (4)).

The main difference between our modified PFC mechanism and the typical PFC mechanism is that, if the queue length of some queue $q_{in}^{i,j}$ at some network node exceeds the PFC threshold, PFC engine will pause the packet transmission of the immediate upstream node on priority class j-1 over the incoming link instead of on priority class j. This feature is important for realizing our TTL-based buffer management scheme, as we will introduce in the next.

1.3 TTL-based Buffer Management Scheme

Buffer division: we divide the buffer of network nodes into k partitions, and let the j-th partition associated with priority class j. If a packet is classified into priority class j, it will be buffered in the j-th buffer partition. Packets queued in queues $q_{in}^{i,j}$ and $q_{out}^{i,j}$ $(1 \leq i \leq n)$ are the packets currently buffered in the j-th buffer partition.

Let d be the number of hops of the longest legal routing path in the network, n be the number of switch ports, and m_p be the maximum packet size. Our buffer division will always ensure $k \geq d$, and the size of any buffer partition is no smaller than $n * m_p$.

TTL-based packet buffering:

1. Initially, we set the TTL values of all packets to $ttl_0 = d$ at all the source servers. TTL value of every packet will be decreased by 1 per hop. At all the source servers, packets are buffered in a buffer of priority class 0.
2. Let ttl_i be the TTL value of a packet p at its i-th hop $(i \geq 0)$. If $ttl_i = 0$, packet p will be dropped by the receiving switch or server. At every hop, the priority class of any incoming packet p is calculated as $\lambda_p = ttl_0 - ttl_i$. Packet p will be buffered and queued according to the calculated priority class λ_p at every hop.

1.4 Proof of Deadlock-free

In this part, we are going to prove that our TTL-based solution is deadlock-free regardless of the packet scheduling algorithms.

In our proof, we use p_{size}, p_{ttl} and p_{dst} to denote the size, the TTL value and the destination of a packet p, respectively. Any ingress queue $q_{in}^{i,j}$ and any egress queue $q_{out}^{i,j}$ are viewed as packet sets. According to the switch model, we have $q_{in}^{i} = \cup_{j=1}^{k} q_{in}^{i,j}$, and $q_{out}^{i} = \cup_{j=1}^{k} q_{out}^{i,j}$.

We use $|q_{in}^{i,j}|$ and $|q_{out}^{i,j}|$ to denote the queue lengths of $q_{in}^{i,j}$ and $q_{out}^{i,j}$, where $|q_{in}^{i,j}| = \sum_{p \in q_{in}^{i,j}} p_{size}$, and $|q_{out}^{i,j}| = \sum_{p \in q_{out}^{i,j}} p_{size}$.

1.4.1 Definition of switch functions

1. **Packet scheduling function of VIQs f_{sche}^{in}:** This is the switch function to decide which packet currently queued in some VIQ q_{in}^{i} to be processed at first. Formally, this function can be expressed as follows.

$$f_{sche}^{in} : q_{in}^{i} \longrightarrow p \in q_{in}^{i} \tag{1}$$

2. **Packet scheduling function of VEQs f_{sche}^{out}:** This is the switch function to decide which packet currently queued in some VEQ q_{out}^{i} to be processed at first. Formally, this function can be expressed as follows.

$$f_{sche}^{out} : q_{out}^{i} \longrightarrow p \in q_{out}^{i} \tag{2}$$

3. **Packet forwarding function f_{fwd}:** This is the switch function to forward a packet from a VIQ to a VEQ. Formally, this function can be expressed as follows.

$$f_{fwd} : (q_{in}^{i_1,j}, q_{out}^{i_2,j}, p \in q_{in}^{i_1,j}) \longrightarrow \begin{cases} q_{in}^{i_1,j} = q_{in}^{i_1,j} - \{p\}, \text{current node is destination of } p, \\ q_{out}^{i_2,j} = q_{out}^{i_2,j} \cup \{p\}, \text{otherwise.} \end{cases} \tag{3}$$

4. **Packet transmission function f_{trans}:** This is the switch function to send a packet from the VEQ of current network device to the VIQ of the immediate downstream device. Formally, this function can be expressed as follows:

$$\begin{aligned} f_{trans} :& (q_{in}^{i_1,j}, q_{out}^{i_2,j}, q_{in}^{i_3,j+1}, p \in q_{in}^{i_1,j} \cap q_{out}^{i_2,j}) \longrightarrow \\ & (q_{in}^{i_1,j} = q_{in}^{i_1,j} - \{p\}, q_{out}^{i_2,j} = q_{out}^{i_2,j} - \{p\}, \\ & q_{in}^{i_3,j+1} = q_{in}^{i_3,j+1} \cup \{p\}), \\ & \text{if } p_{ttl} > 0 \end{aligned} \tag{4}$$

$$\begin{aligned} f_{trans} :& (q_{in}^{i_1,j}, q_{out}^{i_2,j}, p \in q_{in}^{i_1,j} \cap q_{out}^{i_2,j}) \longrightarrow \\ & (q_{in}^{i_1,j} = q_{in}^{i_1,j} - \{p\}, q_{out}^{i_2,j} = q_{out}^{i_2,j} - \{p\}), \\ & \text{if } p_{ttl} = 0 \end{aligned} \tag{5}$$

Note that here $q_{in}^{i_1,j}$ and $q_{out}^{i_2,j}$ are two queues belonging to the same network node, while $q_{in}^{i_3,j+1}$ is a queue belonging to an immediate downstream network node.

1.4.2 Assumptions

Let $t_{cost}(f)$ be the time cost of performing switch function f. In the following part, we list several assumptions we make in our proof.

1. **PFC threshold is no smaller than maximum transmission unit (MTU):** Formally, we have $t_{PFC} \geq s_{MTU}$.
2. **Finite packet scheduling time:** We assume that packet scheduling functions can decide the next packet to be processed within finite time. Formally, f^{in}_{sche} and f^{out}_{sche} should satisfy the following constraints at any network node:

$$t_{cost}(f^{in}_{sche}(q^i_{in})) < \infty, 1 \leq i \leq n \quad (6)$$

$$t_{cost}(f^{out}_{sche}(q^i_{out})) < \infty, 1 \leq i \leq n \quad (7)$$

3. **No selection of duplicated packet:** the packet scheduling function f^{in}_{sche} will not select the same packet twice. Formally,

$$\begin{aligned} f^{in}_{sche}: \quad & q^{i_1}_{in} \longrightarrow p \in q^{i_1}_{in}, \\ \text{Subject to}: \quad & 1 \leq i_2 \leq n, \forall p\prime \in q^{i_2}_{out}, p \neq p\prime. \end{aligned} \quad (8)$$

4. **Prefering non-paused packet:** We assume f^{out}_{sche} will always prefer non-paused packets over paused packets.
5. **Finite packet forwarding time:** We assume that any packet can be forwarded from a VIQ to a VEQ within finite time. Formally, f_{fwd} should satisfy the following constraint:

$$t_{cost}(f_{fwd}((q^{i_1,j}_{in}, q^{i_2,j}_{out}, p \in q^{i_1,j}_{in}))) < \infty, \quad 1 \leq i_1, i_2 \leq n \quad (9)$$

6. **Finite packet transmission time when no PFC pause:** We assume that any packet can be sent to the VIQ of next hop within finite time when the priority class of the packet is not paused by the immediate downstream device. Formally, f_{trans} should satisfy the following constraint:

$$t_{cost}(f_{trans}(q^{i_1,j}_{in}, q^{i_2,j}_{out}, q^{i_3,j+1}_{in}, p \in q^{i_1,j}_{in} \cap q^{i_2,j}_{out})) = \begin{cases} < \infty, & |q^{i_3,j+1}_{in}| < t_{PFC} \\ \infty, & \text{otherwise.} \end{cases} \quad (10)$$

1.4.3 Network Buffer State

Buffer state: we use a $2n$ dimensional vector to represent the buffer state of a network node u as follows:

$$BS_u =< q^1_{in}, \ldots, q^n_{in}, q^1_{out}, \ldots, q^n_{out} >$$

$BS_u(t)$ is the buffer state of node u at time t.

Let $N(V, E)$ be the network topology, where V is the set of network nodes and E is the set of network links. We represent the buffer state of netwrok N (denoted as BS_N) as follows:

$$BS_N =< BS_{u_1}, \ldots, BS_{u_i}, \cdots >, \forall u_i \in V$$

$BS_N(t)$ is the buffer state of network N at time t.

Legal buffer state: We say BS_u is a legal buffer state for node u when the following two conditions are satisfied:

$$|q^i_{in}| < \infty \quad \text{and} \quad |q^i_{out}| < \infty, \quad 1 \leq i \leq n \quad (11)$$

$$|q^{i,j}_{in}| \leq t_{PFC}, \quad 1 \leq i \leq n, 1 \leq j \leq k \quad (12)$$

We say BS_N is a legal buffer state when the buffer state of every node in N is legal.

Empty buffer state: We say BS_u is an empty buffer state when the following condition is satisfied:

$$|q^i_{in}| = 0 \quad \text{and} \quad |q^i_{out}| = 0, \quad 1 \leq i \leq n \quad (13)$$

We say BS_N is a empty buffer state when the buffer state of every node in N is empty. Specially, we denote the empty buffer state of network N as BS^0_N.

Deadlocked buffer state: $\forall t < \infty$, we say $BS_N(t)$ is a deadlocked buffer state if the following condition is met when no new packets are injected into the network since t:

$$\begin{aligned} & \exists \text{ finite } t_0 > t, \quad \forall t_1 > t_0, \\ & BS_N(t_1) \equiv BS_N(t_0) \quad \text{and} \quad BS_N(t_0) \neq BS^0_N. \end{aligned} \quad (14)$$

1.4.4 Proof of deadlock-free

Claim: our TTL-based solution is deadlock-free regardless of the packet scheduling algorithms.

Proof: To prove our TTL-based solution is deadlock-free, we prove by contradiction that for arbitrary legal buffer state $BS_N(t)$, $BS_N(t)$ is not a deadlocked buffer state under our TTL-based solution.

Assuming there exists a deadlocked buffer state $BS_N(t)$ under our TTL-based solution. If no new packets are injected into the network since t, according to Equation 14, $BS_N(t)$ will converge into a fixed non-empty buffer state $BS_N(t_0)$ at some finite time $t_0 > t$.

According to Equations (6)-(9), any unscheduled packet remaining in the ingress queue will be put into

2. Buffer Analysis

2.1 PFC Headroom

A PAUSE message sent from a receiver to an upstream sender needs some time to arrive and take effect. To avoid packet drops, the receiver (i.e., the sender of the PAUSE message) must reserve enough buffer to accommodate any packets it may receive during this time. In this part, we calculate the amout of buffer needed as the PFC headroom.

Per port per priority PFC headroom: At first, we calculate the time for a PFC message to arrive and take effect at its destination port. It mainly consists of six parts.

1. **The time to send a PAUSE message at the receiver side (denoted as t_{snd})**: The transmission of the PAUSE

frame can pass ahead of any other packet queued in the receiver, but cannot preempt another frame currently being transmitted in the same direction. Hence in the worst case, the receiver generates a PAUSE frame right when the first bit of a maximum-size packet (i.e., the size is equal to Maximum transmission unit (MTU)) has started engaging the transmission logic. So we have $t_{snd} = (s_{MTU} + s_{PFC})/r_l$, where s_{MTU} is the value of MTU, s_{PFC} is the size of PFC message and r_l is the line rate of the network link.

2. **The time for the PAUSE message to propagate from the receiver to the sender over the network link (denoted as t_{wire})**: The value of t_{wire} is related to the materia and the length of the network links in use.

3. **The time to receive a PAUSE message at the sender side (denoted as t_{rev})**: It is easy to know $t_{rev} = s_{PFC}/r_l$.

4. **The time to process a PAUSE message at the sender side (denoted as t_{pro})**: After a PAUSE message has been received by the sender, it will take an implementation-dependent amount of time to process the message and stop packet transmission.

5. **The time to stop packet transmission at the sender side (denoted as t_{stop})**: After the sender finally decides to stop packet transmission, it can stop only at packet boundaries, to avoid packet corruption. In the worst case, the sender will have completed the process of the PAUSE message just when the first bit of a maximum-size packet has started engaging the packet transmission. So we have $t_{snd} = s_{MTU}/r_l$

6. **The time for the remaining bytes of packets on the link to get drained after stopping packet transmission at the sender(also denoted as t_{wire})**: This part of time is equal to the time for the PAUSE message to propagate from the receiver to the sender.

In summary, the per port per priority PFC headroom can be expressed using the following equation:

$$\begin{aligned} b_{hr} &= r_l * (t_{snd} + t_{wire} + t_{rev} + t_{pro} + t_{stop} + t_{wire}) \\ &= 2(s_{MTU} + s_{PFC} + r_l * t_{wire}) + r_l * t_{pro} \quad (15) \end{aligned}$$

For typical TCP/IP based RDMA DCNs, we have $s_{MTU} = 1500$bytes, $s_{PFC} = 64$bytes. The length of links used in a single DCN is usually no larger than 300 meters [4]. As every 100 meters of link delays the reception of a packet by about 500ns for copper cables [1], we have $t_{wire} \leq 1.5us$.

The value of t_{pro} is implementation related. Let a quanta be the time needed to transmit 512 bits at the current network speed. The PFC definition caps this time to 60 quanta for any implementation [1]. Hence we have $r_l * t_{pro} = 512 * 60 = 30,720$ bits.

According to the above parameters, when $r_l = 40Gbps$, the per port per priority PFC headroom $b_{hr} \leq 21968$ bytes ≈ 22 KB.

PFC headroom of a switch: For a n port switch which supports k priority classes, PFC PAUSE is possible to be triggered at all ports and priority classes simultaneously. So we should at least reserve $n*k*b_{hr}$ buffers as PFC headroom at every switch.

For commodity switches like Arista 7050QX32 which has 32 full duplex 40Gbps ports, when supporting 8 priority classes, it requires about $32 * 8 * 22 = 5632$KB buffer per prioriry as the PFC headroom.

2.2 Necessary Buffer for Achieving Work-Conservation

According to the rule-of-thumb [2, 7], we at least need a buffer of size $b = C * RTT$ per port for achieving work-conservation, where RTT is the average round-trip time, and C is the link capacity. As our TTL-based solution allocates dedicated buffer to different priority classes, for a switch of n ports, when supporting k priority classes, it demands a buffer of size $n * k * C * RTT$.

Assuming that the average RTT in the DCN is 50us. For Arista 7050QX32 switch, when supporting 8 priority classes, we need to reserve a buffer of size $32 * 8 * 40Gbps * 50us = 64MB$. This apparently exceeds the total buffer size of most commodity switches. The key to solving this problem is to let packets of different priority classes share some buffer.

On the other hand, we also need to ensure that PFC threshold is not reached before the switch has a chance to mark packets with ECN. This has been well discussed in DCQCN [8]. Basically, we can leverage a dynamic PFC thresholding scheme to dynamically share the switch buffer among different ports.

2.3 Preliminary Solution For Reducing Buffer Demand

Reducing the PFC headroom: For tree-based topologies like Fat-tree, packets of consecutive priority classes will not enter the same switch. So the number of priority classes a switch needs to support is halved. In addition, if we enforce that every switch drops the packets bounced back by the immediate downstream devices, a switch needs to support only $1/4$ of the total priority classes in use.

Reducing the necessary buffer for achieving work-conservation: The main problem here is that our TTL-based solution allocates dedicated buffer to different priority classes. To solve this problem, instead of simply allocating dedicated buffer to different priority classes, we can let packets of different priority classes to have both dedicated and shared buffer.

Specifically, we divide the switch buffer into $k + 1$ partitions. The first k partitions are dedicated for the k priority classes, correspondingly. The $k + 1$ partition is shared by all the priority classes. If a packet is classified as priority class i, at first the switch will check whether partion i has enough spare buffer to accommodate this packet. If yes, this packet will be placed in partition i. Otherwise, the switch will place this packet in partition $k + 1$.

The original dynamic PFC thresholding scheme cannot be directly applied under the new buffer allocation scheme. Hence we modify it as follows. Let b_i be the size of partition i, t^i_{PFC} be the PFC threshold for priority class i, s_i be the amount of buffer of partition i that is currently occupied. We have

$$t^i_{PFC} = \alpha \frac{b_i}{\sum_{j=1}^{k} b_j}(b_{k+1} - s_{k+1}) + \alpha(b_i - s_i) \quad (16)$$

The intuition behind Equation 16 is to share the buffer of partition $k+1$ among different priority classes with respect to the weight $\frac{b_i}{\sum_{j=1}^{k} b_j}$. As for the buffer of dedicated partitions, we follow the idea of original dynamic PFC thresholding scheme to dynamically share it among different ports within the same priority class.

Under the new buffer allocation scheme and the new dynamic PFC thresholding scheme, the buffer needed for achieving work-conservation is reduced to $n * C * RTT$, which is within the buffer capacity of most commodity switches.

3. References

[1] Priority flow control: Build reliable layer 2 infrastructure. `http://www.cisco.com/c/en/us/products/collateral/switches/nexus-7000-series-switches/white_paper_c11-542809.pdf`.

[2] Guido Appenzeller, Isaac Keslassy, and Nick McKeown. Sizing router buffers. In *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '04.

[3] Shang-Tse Chuang, Ashish Goel, Nick McKeown, and Balaji Prabhakar. Matching output queueing with a combined input/output-queued switch. *IEEE Journal on Selected Areas in Communications*.

[4] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitendra Padhye, and Marina Lipshteyn. Rdma over commodity ethernet at scale. In *SIGCOMM '16*.

[5] Nick McKeown. The islip scheduling algorithm for input-queued switches. *IEEE/ACM Trans. Netw.*

[6] Nick McKeown, Martin Izzard, Adisak Mekkittikul, William Ellersick, and Mark Horowitz. Tiny tera: a packet switch core. *IEEE Micro*, 1997.

[7] Curtis Villamizar and Cheng Song. High performance tcp in ansnet. *SIGCOMM Comput. Commun. Rev.*

[8] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. Congestion control for large-scale rdma deployments. In *SIGCOMM '15*.