FAFC: Fast and Accurate Flow Control in Data Center Networks

Chengdi Lu, Yuang Chen, *Graduate Student Member, IEEE*, Fangyu Zhang, and Hancheng Lu, *Senior Member, IEEE*

***Abstract*—In data centers, large-scale many-to-one traffic can rapidly exhaust switch buffers and trigger priority-based flow control (PFC) pause, resulting in increased flow completion time (FCT) for uncongested flows. To address this issue, we propose an innovative switch-side fast and accurate flow control (FAFC) scheme. By differentially allocating pause time for each port during congestion, FAFC can minimize the performance loss for uncongested flows. Furthermore, FAFC is also coupled with an effective queue length prediction algorithm to enable proactive and reliable estimation of the congestion level. Extensive system-level simulations demonstrate that FAFC can flexibly allocate pause times across congested ports, which are not only compatible with existing PFC but also do not require per-flow states. We implemented FAFC in P4 programmable switches, showing it as lightweight flow control method that is portable for implementation in hardware. Remarkably, our large-scale simulations illustrate that compared to traditional PFC, FAFC improves the average FCT slowdown and 95% FCT slowdown by 10.6% and 23.3%, respectively, under Hadoop workload when performing HPCC congestion control.**

***Index Terms*—Data center networks (DCNs), flow control, priority-based flow control (PFC), congestion control**

I. Introduction

Nowadays, more and more applications have been deployed in data centers as they can provide extremely high computational and storage resources to confront increasing quality-of-service (QoS) requirements [1]. The emergence of new applications, such as large-scale deep learning, data mining, and parallel computing, has further intensified the demand for computing, storage, and network resources. These applications typically adopt a distributed architecture, requiring efficient data transfer and message passing across a large number of hosts. As a result, the need for high-bandwidth and low-latency communication has made the shallow-buffered interconnect network a critical bottleneck in modern data centers [2]. To address this challenge, remote direct memory access (RDMA), which originates in the high-performance computing (HPC) community, has been extensively employed in data centers since it can reduce CPU usage efficaciously and offer high-bandwidth and low-latency communication [3], [4]. Nevertheless, due to the limited resources of commodity NICs, widely deployed RDMA schemes, including InfiniBand [5] and RDMA over Converged Ethernet (RoCE) [6], necessitate near-zero packet loss to achieve optimal performance.

DCN traffic patterns tend to be highly aggressive, as many data center applications start sending at the line rate to fully exploit available bandwidth and only reduce their transmission rate when congestion exists [7]. Moreover, most data center applications leverage clustered operations to distribute the heavy workloads, resulting in large-scale many-to-one traffic patterns. When multiple hosts simultaneously transmit data to the same destination, packets rapidly accumulate at the bottleneck switch, leading to buffer overflow, increased queuing delays, and degraded throughput. This phenomenon, known as *the incast problem* [8], [9], exacerbates network congestion, degrades flow completion time (FCT), and leads to severe fluctuations in latency and throughput, posing significant challenges to maintain stable performance of DCNs.

Dealing with congestion in data centers, especially solving the incast problem, is a trending topic. It can be categorized into two main types: hop-by-hop flow control (FC) and end-to-end congestion control (CC). FC is mainly used by switches and acts on the port-level, while CC is implemented in endpoints and acts on the flow-level. Current DCNs mainly relay on advanced CC schemes to enable fast and precise reactions to congestion for an individual flow, which achieve low-latency communication and improve overall FCT. Recent advancements in data center transport (e.g., [10], [11], [12], [13], [14], [15]) mainly focus on achieving fast and accurate congestion control to maintain low queuing delay and high link utilization, all while avoiding modifications to the flow control mechanism.

Although CC schemes are widely adopted in data centers and become a key research focus, they have immanent limitations. In these schemes, most congestion indications (e.g., explicit congestion notification (ECN) [16], round trip time (RTT) [17], and in-band network telemetry (INT) [4]) are needed to be feedbacked from the receiver, which enforces long feedback loop delay and limits their minimal control loop delay to a single RTT. As a result, they cannot effectively handle congestion that happens within one RTT, especially during the first RTT of the incast. Furthermore, CC schemes lack the ability to leverage historical flow information when initiating new flows, requiring at least one RTT to converge. In contrast, flow control methods have a much lower control delay, which enables fast reaction and recovery from congestion. They are capable of mitigating congestion within a single RTT, overcoming the constraints of congestion control. In addition, flow control schemes mostly operate on the

(Chengdi Lu and Yuang Chen are co-first authors.) (Corresponding author: Hancheng Lu.)

Chengdi Lu, Yuang Chen, Fangyu Zhang, and Hancheng Lu are with the University of Science and Technology of China, Hefei 230027, China (email: {lcd1999, yuangchen21, fv215b}@mail.ustc.edu.cn; hclu@ustc.edu.cn). Hancheng Lu is also with the Institute of Artificial Intelligence, Hefei Comprehensive National Science Center, Hefei 230088.

switches, which can exploit its local queue and port statistics to obtain more accurate control of queue length to prevent congestion. Building upon these advantages, our work focuses on optimizing flow control to enable fast congestion response while simultaneously minimizing its potential negative impact.

To achieve lossless Ethernet, priority-based flow control (PFC) is widely adapted in DCNs. It enhances traditional Ethernet flow control by extending the control granularity from a single port to eight priority queues in each port. As a result, those priority queues can be separately paused and resumed without affecting others. PFC uses pause frames as a backpressure signal, which is sent upstream in case of congestion and keeps the buffer from overflowing, thus preventing the switches from dropping packets and achieving zero packet loss. A switch asks its upstream to stop sending when the corresponding ingress queue length is above the threshold and asks the upstream to resume if this queue length returns to its normal value, without keeping track of individual flows. It has a simple implementation and its reaction to congestion is faster than end-to-end solutions because of the low feedback loop delay. However, it can only act on aggregate flows incoming from a specified port with a certain priority, which causes many problems. Although PFC is an essential building block for data center to prevent buffer overflow during incast scenarios, its coarse-grained control introduces several challenges, and using it solely does not give promising performance results.

Victim Flow Problem: PFC reintroduces the head-of-line blocking problem to today's non-blocking switching architecture. When flow control pauses happen, the egress queue of the specified port and priority is paused in the upstream switch, even if there are packets destined to other uncongested path, it has to wait until the congestion resolves and the link is resumed [18]. Due to the head-of-line blocking problem, congestion can spread multiple levels upstream not only on the congested path, severely degrading the performance of flows as they experience throughput loss and increased queuing delay [19]. Those victim flows will experience large delay spikes during large-scale incast scenarios because of the consecutive triggering of the flow control pause. Particularly, malicious user or faulty hardware which stops processing received packets can continuously send pause frame to the upstream and stop a large area of traffic [20].

Late Reaction Problem: Due to the performance degradation caused by PFC, modern data centers leverage congestion control to keep minimal queue length and avoid triggering of the flow control pause, thus it can minimize the PFC problems. In these schemes, PFC is leveraged as a last resort to ensure lossless and is configured a high triggering threshold. Also, this threshold checking is based on the length of ingress queues. When congestion control cannot reduce rate in time during congestion, flow control will also take no action until the ingress queue is about to full, resulting in late reaction. This means that this problem is mainly caused by considerations to protect victim flows, so, it cannot be easily solved without addressing the first problem.

Many enhancements are addressing these challenges of flow control. While most focus on Ethernet, it is trivial to adapt to flow control methods in other similar network architectures like InfiniBand's credit based flow control (CBFC) [6]. Gentle flow control (GFC) [21] extends pause states to various discrete rates, eliminates hold and wait, and solves the deadlock problem of PFC. Congestion aware priority flow control (CaPFC) [22] monitors both ingress and egress queues and sends flow control pauses proactively to prevent the egress queue from being fully occupied. Proactive PFC (P-PFC) [23] also proactively triggers flow control pauses in advance using derivatives of buffer occupancy, which effectively reduces tail FCT. Backpressure flow control (BFC) [24] and FlowSail [25] approximates per-hop per-flow flow control by only allocating dedicated status and queues for active flows. Although BFC and FlowSail does not have victim flows, switches still need to track the costly per-flow state and queue collision is unavoidable if active flows exceed max number of available queues.

In this paper, we propose fast and accurate flow control (FAFC) to address these two problems. To address the victim flow problem, we show that the bad performance impact on victim flows can be minimized in terms of throughput loss by modeling it as an optimization problem that adaptively allocates pausing time to each ingress port. To help solve the late reaction problem, we incorporate an effective queue length prediction algorithm and a proactive ECN marking mechanism. The queue prediction algorithm can speed up congestion reaction further and provide a more accurate estimation of the congestion level, while ECN marks can not only benefit from the proactive prediction to reduce buffer occupancy, but also avoid false marking caused by traditional flow control schemes to improve link utilization. Furthermore, We adopt egress queue statistics to trigger flow control pauses, which enables accurate protection for victim flows and fast reaction to congestion.

Our main contributions are as follows.

- We defined a measure to characterize the performance loss for uncongested flows using only per-port statistics and proposed a novel flow control method to push egress congestion back to ingress ports by allocating pausing time to each port. This scheme minimizes the throughput loss for uncongested flows using optimization methods, which can effectively protect victim flows.
- We trigger flow control mechanism proactively by using first-order prediction of both ingress and egress queue statistics in FAFC, which enables fast reaction to congestion and lowers queuing delay. We also eliminated the resume frame used in traditional PFC as this method calculates pausing time in advance.
- We implemented FAFC using P4 programmable switches in our testbed, showing FAFC is lightweight and portable for practical implementation in commodity hardware. We also conducted extensive large-scale simulations and demonstrated that FAFC can achieve performance boost by minimizing the degradation of victim flows using optimization method and solving it efficiently.

The rest of the paper is organized as follows. The overall design as well as individual methods used in FAFC are shown

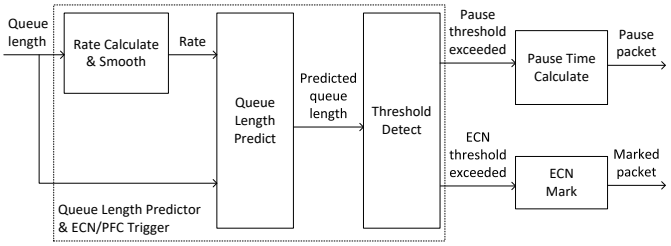


Fig. 1. FAFC architecture.

in Sec. II. Sec. III presents the implementation and detailed algorithms of the proposed FAFC. In Sec. IV, the results and analysis of extensive simulations are provided. Finally, Sec. V concludes this paper and discusses future directions.

II. Method Design

In this section, we show the design and main considerations of FAFC and provide some mathematical analysis to illustrate these design factors which can overcome problems of PFC and improve performance. Fig. 1 shows the architecture of FAFC, which mainly consists of four components: queue length predictor, pause time solver, pause frame sender and receiver, and ECN marker.

A. Queue Statistic Definition

Current data center switches commonly use the shared-buffer memory architecture. In this architecture, packet buffer are shared by all ports, and switches can use both static and dynamic thresholds to ensure fair use of the buffer [26]. Each port still has its ingress queues and egress queues. However, those queues do not hold the packet itself, instead, they keep the pointer of the packet which resides in the shared packet buffer. Each unicast packet is counted twice, in both ingress and egress queues. Those queues have associated counters that track the size and are used for PFC triggering and ECN marking.

Counters in switches can also work independently and be used to track a set of flows without creating separate queues. In FAFC, we introduce per-input-port per-priority counters for egress ports as basic queuing statistics for switches, meaning it requires switches to keep track of the buffer statistics of every individual (ingress port, egress port, priority) pair. Those queue statistics are useful and have been used in various works such as CaPFC [22] to achieve different goals. For simplicity but not losing generality, we target flows with a specific priority and omit the priority field in most discussions below. We also assume all links in the data center have a uniform one-way delay T_{hop} for each link. We show that FAFC still works under heterogeneous topologies with different link delays in Section III. To mathematically analyze the flow control schemes, we need proper definitions for these statistics.

For every queue with ingress port i and egress port e, we use notations as shown in TABLE I. We also use notations for aggregated queue statistics, as shown in TABLE II. They are simply the summation of the corresponding individual queue statistics. When there is no congestion, for any ingress queue of port i, we have $r^i_{i*} = r^o_{i*}$ and $r_{i*} = 0$, and for any egress queue of port e, we have $r^i_{*e} = r^o_{*e}$ and $r_{*e} = 0$. So both ingress and egress queues won't build up. When congestion happens, the queue of the corresponding port starts building up. In that case, the input rate r^i_{i*} is greater than the service rate r^o_{i*}, and the incoming rate r^i_{*e} is greater than the output rate r^o_{*e}. These relationships become the opposite when congestion begins to alleviate and the queue is draining out.

TABLE I
Individual queue statistics

Symbol	Meaning
q_{ie}	queue occupancy of all flows from port i and to port e
r^i_{ie}	ingress rate of all flows from port i and to port e
r^o_{ie}	egress rate of all flows from port i and to port e
r_{ie}	buildup rate of q_{ie}

TABLE II
Aggregated queue statistics

Symbol	Meaning
$q_{i*} = \sum_e q_{ie}$	ingress queue length of port i
$q_{*e} = \sum_i q_{ie}$	egress queue length of port e
$r^i_{i*} = \sum_e r^i_{ie}$	total ingress rate of all flows from port i
$r^o_{*e} = \sum_i r^o_{ie}$	total egress rate of all flows to port e
$r^o_{i*} = \sum_e r^o_{ie}$	total ingress rate of all flows from port i
$r^i_{*e} = \sum_i r^i_{ie}$	total egress rate of all flows to port e
$r_{i*} = \sum_e r_{ie}$	ingress queue buildup rate of port i
$r_{*e} = \sum_i r_{ie}$	egress queue buildup rate of port e

B. Queue Length Prediction

Currently, ECN marking and PFC sending behavior in data center switches are based on simple thresholds of instantaneous queue length. A simple idea of improvement is to use queue length prediction so that it can proactively cope with congestion. P-PFC [23] already shows that sending flow control pauses in advance can reduce buffer occupation and improve tail FCT. If we apply prediction to the queue length and use it to trigger flow control pauses, the reaction to congestion will be faster. The peak queue length decreases, and the flows will experience a lower delay.

In the above optimization problem, switches exploit queue statistics to determine the proportion of congested flows within each ingress port, and use it to allocate pause time for ports selectively. However, queue statistics do not foresee and account for the inflight data, so they are not fast enough to detect congestion. Combining queue prediction with egress PFC can obtain more forward-looking results and improve performance further. Also, we can consider the effect of flow control pauses during prediction by deducting the bytes during the pause and adding them back after the resumption. This enables us to estimate the traffic demands more precisely during flow control pauses.

As switches have strict limits on algorithm complexity for per-packet operations, we adopt Holt-Winter's exponential prediction methods to provide a base estimation for queue length. Holt-Winter's algorithm is simple but effective, and it has been widely used in many time series forecasting tasks.

Holt-Winter's first-order method, also called Holt's linear trend method, primarily involves three equations, including one forecast equation and two smoothing equations, which can be specifically expressed as follows:

$$\hat{y}_{t+h|t} = l_t + hb_t, \tag{1}$$
$$l_t = \alpha y_t + (1-\alpha)(l_{t-1} + b_{t-1}), \tag{2}$$
$$b_t = \beta(l_t - l_{t-1}) + (1-\beta)b_{t-1}. \tag{3}$$

The primary advantage of our proposed method is that it considers the trend of values to derive the predictions. The trend is characterized by differencing the time series. We use this method to predict the queue length after a single RTT ($2\times T_{hop}$) and use the predicted values to trigger ECN marking and flow control pauses.

To account for the inference of flow control pauses, we introduce time-slotted statistics for tracking the pending, effective, and expired pause. Each time slot has a length of T_{hop} and starts when an ingress port is being paused and when the earlier time slot ends but the congestion persists. During each time slot, three variables are tracked: $pausNext$, $pausOld$, and $pausPending$. $pausNext$ indicates the bytes deferred by the pause that are effective in this time slot and will be deducted in the prediction. $pausOld$ indicates the bytes of expired pause during the last time slot. Upstream hops will send these paused data in the current time slot, so they need to be included in the prediction. $pausPending$ represents the amount of paused bytes that will take effect in the next time slot, transitioning to $pausNext$ once the next time slot begins. When switches send pause to ingress ports, they will calculate the bytes deferred and add them to $pausNext$ during the beginning of each time slot. If its value is too large or the pause happens near the end of the time slot, the residue is added to $pausPending$.

C. Egress Flow Control Triggering

As flow control mechanisms stop the ingress port to throttle the upstream, traditional PFC uses the length of ingress queues q_{i*} as the criterion for triggering flow control pauses, which is quite straightforward. As a result, traditional PFC behavior only relies on ingress queue statistics. We classify them as *ingress flow control triggering*. However, as congestion happens on egress ports, egress queues fill up first, earlier than the increasing of ingress queues. So, the length of egress queues q_{*e} is more suitable for indicating congestion and should be a better metric to trigger flow control pauses. In this paper, egress queue statistics are also considered in the triggering of pause, and we classify this scheme as *egress flow control triggering*. Moreover, because egress flow control triggers before the filling up of egress queues, egress processing is not blocked and prevents head-of-line blocking from happening.

CaPFC [22] and our FAFC share the idea of using both ingress and egress queue lengths to make flow control decisions. However, CaPFC stops ingress ports with exceeded contributions to an egress queue. The ingress ports in CaPFC are treated identically, without considering protecting victim flows. Although P-PFC also incorporates the idea of the prediction and pause port selection using buffer statistics, it randomly chooses the pause port according to their fraction in egress queue. As a result, there are still chances that ports with victim flows are selected and paused, especially when both incast and victim flows share the same ingress port. Moreover, its critical parameter T, which controls its aggressiveness and reduces false alarm, is tricky to find as the optimal value varies with different topologies and flow patterns.

To incorporate egress queue statistics to flow control, we need a method to propagate egress congestion to ingress ports. This is not straight forward and usually difficult to efficiently calculate at switches. FAFC solves this by modeling it as an optimization problem. By solving this optimization problem, we can find out the ingress ports that have a high contribution to queue buildup and proactively take action.

D. Selective Port Pausing

Since the backpressure mechanism of traditional PFC is not per-flow but per-port, it has the problem of congestion spreading and victim flows. Normally, congested flows won't experience throughput loss because the switch is draining buffered packets for these paused flows. However, it inevitably pauses other flows that share the same ingress port, if the pausing time is too long, the queue will become empty, and throughput loss happens. In most cases, congestion will spread towards the upstream, exhausting the buffer of most upstream switches. This congestion spreading largely impacts the performance of flows traversing upstream ports, although they have no contribution to the congestion.

Since PFC imposes different impacts on flow performance, we can separate flows into two types according to whether they traversed the bottleneck link, i.e., congested and uncongested/victim flows. For congested flows, PFC only pushes inflight data to buffer at upstream hops and does not cause performance degradation. However, the flow control pause stops other flows sharing the same congested ingress port as well and causes performance loss. Moreover, although these flows do not contribute to the congestion, they experienced increasing delay and ECN marking probability, which cause source node to decrease its sending rate. For the purpose of improving fairness, the same pause threshold is typically configured for switch ports with the same link speed. However, in some scenarios, selectively suspending ports with heavy incast traffic to protect upstream ports with little incast traffic from being paused can provide better performance.

As illustrated in Fig. 2, we consider a typical topology called 'dumbbell' topology [11], [27], which consists of two interconnected switches and a bunch of servers connected to each switch, to represent two racks in the data center. The link between two switches carries both congested and uncongested flows, while most of the time each endpoint link only carries a single type of them. Therefore, if switches detect that the endpoint link only has congested flows, it can safely pause this port to reduce buffer usage without compromising performance. On the contrary, the switch interconnect link should be protected and given the minimal pausing time to reduce side effects on uncongested flows. This results in fewer

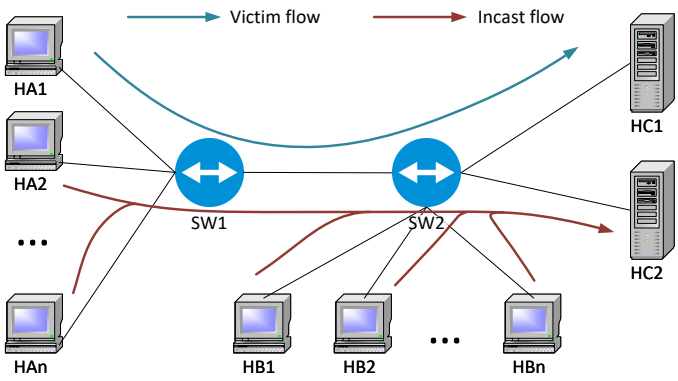


Fig. 2. Dumbbell topology.

packets of congested flows buffered at the congested switch. Instead, they are kept buffered at the upstream hops, which improves performance due to the more efficient usage of the switch's buffer.

FAFC addresses this problem by incorporating link utilization loss into its objective function. As a result, FAFC tends to pause ports with more congested flows and protects the uncongested flows.

Since we changed the trigger source of flow control pauses from pure buffer usage of ingress ports to egress queue length, we need to find a method to propagate the congestion from an egress port to ingress ports and determine the ingress ports to be paused. In this stage, we also calculated the corresponding pause time for each paused port using queue statistics to avoid the transmission of PFC resume packet. This removes the need of keeping additional buffer to avoid link underutilization because the time of resume is precalculated without additional delay.

The flow control pause time has the following requirements: it needs to be large enough to prevent buffer overflow, and it also needs to be minimal to avoid performance loss. Unfortunately, these two objects are contradictory, and preventing buffer overflow is more critical. Therefore, it is a good idea to minimize the performance loss while preventing buffer overflow. The performance loss can be characterized by the wasted egress link capacity during the pausing state, which can be represented as follows:

$$loss_i(T_i) = \sum_j \max\left(T_i\ r^o_{ij} - q_{*j}, 0\right). \tag{4}$$

To maximize system performance, we model the selection of pause ports and the determination of pause time as an optimization problem to maximize overall performance. The optimization variable is the pause time T_i for each ingress port i. T_i is a non-negative value and $T_i = 0$ means the port is neither selected as a pause port nor needed to pause.

(1) Deriving objective function.

By summing out $loss_i(T_i)$ for all ingress ports and swapping the order of summation, the objective function of this proposed optimization problem can be indicated as follows:

$$f(\mathbf{T}) = \sum_{j\neq k} \max\left(\sum_i T_i\ r^o_{ij} - q_{*j}, 0\right), \tag{5}$$

where $\mathbf{T} = \{T_1, T_2, \ldots, T_n\}$ and k is the egress port that triggered the flow control pause.

(2) Deriving constraints.

Due to the propagation delay, upstream hops will not stop sending immediately when flow control pauses occur. The pausing time should be large enough to drain out the same amount of bytes of these additional inflight packets and ensure queue length not increasing after per-hop round-trip delay $2T_{hop}$. If the pause time is too small to prevent queue growth, it will eventually fill the headroom and cause buffer overflow during long-existing congestion. To keep queue length from further increasing, we ensure the admitted bytes are less than transmitted bytes during a single period T_{hop} and get the first constraint (7). Also, due to the time-slotted prediction in FAFC, we obtain fresh statistics every T_{hop}. In this configuration, pause frames are normally sent periodically in every single T_{hop} until the congestion eliminates. So, the pausing time should not be larger than T_{hop}. Hence, we have the optimization problem as follows:

$$\textbf{Minimize}\ f(\mathbf{T}), \tag{6}$$

$$\textbf{s.t.}\ \sum_i (T_{hop} - T_i) r^i_{ik} \leq r^o_{*k}\ T_{hop}, \tag{7}$$

$$0 \leq T_i \leq T_{hop}. \tag{8}$$

The resulting objective function is a complicated piecewise linear function. Although we can convert this optimization model to a mixed integer linear programming problem by introducing auxiliary 0-1 variables, it is still extremely non-trivial and the efficiency of conventional algorithms is not sufficient to solve it in real time. To this end, we need to further simplify this problem by utilizing the different characteristics of queue length between congested and uncongested ports.

For other congested ports $j \neq k$ on the same switch, q_{*j} is usually larger than $\sum_i T_i\ r^o_{ij}$. Thus, its contribution to the value of $f(\mathbf{T})$ is 0. On the contrary, uncontested ports have a small q_{*j}, thus its contribution to the value of $f(\mathbf{T})$ can be approximated as $\sum_i T_i\ r^o_{ij}$. Also, we rewrite (7) to (10) using the equation $r_{*k} = r^i_{*k} - r^o_{*k}$. This converts the optimization problem to a linear programming problem, which can be formulated as follows:

$$\textbf{Minimize}\ g(\mathbf{T}) = \sum_{j\ \text{uncongested}} \sum_i T_i\ r^o_{ij}, \tag{9}$$

$$\textbf{s.t.}\ r_{*k}\ T_{hop} \leq \sum_i T_i r^i_{ik}, \tag{10}$$

$$0 \leq T_i \leq T_{hop}. \tag{11}$$

When $r_{*k} \leq 0$, the optimal value is $T^*_i = 0$ for every port i. This means FAFC will stop sending flow control pauses after queue length starts decreasing, which effectively prevents overreaction to congestion. Otherwise, it is easy to verify that

the inequities are always strictly satiable. So, optimal non-zero solutions always exist when $r_{*k} > 0$.

E. ECN Integration with Dynamic Thresholds

Most congestion control schemes require the congestion signal being triggered before the flow control pause. To achieve this in traditional PFC and ECN networks, PFC thresholds need to be $\sim N$ times larger than the ECN thresholds, where N is the port number of the switch [28]. Due to the usage of egress queue status as the trigger of flow control pauses, we can well integrate end-to-end ECN to FAFC. In FAFC, the triggering of ECN marking uses the same queue length statistics with a threshold lower than the pause threshold. With enough safe margin between pause and ECN threshold, this will not affect the operation of end-to-end congestion control. Furthermore, ECN marking will also benefit from the prediction. It will start marking promptly when congestion happens and will stop marking early when queue length starts decreasing. This can reduce occurrences of buffer underflow and improve link utilization.

FAFC can only provide the guarantee against overflow when the actual trends have been correctly predicted. Buffer overflow may rarely occur if the prediction diverges too much from the actual value. In shared-buffer switches, dynamic pause thresholds are used in traditional PFC for the efficient use of available buffer spaces. By combining FAFC with traditional PFC with dynamic thresholds, we can not only prevent buffer overflow but also leave enough buffer for FAFC to operate effectively.

F. Resume Packet Elimination

Traditional PFC operation consists of two thresholds, i.e., X_{OFF} and X_{ON}. When queue length exceeds X_{OFF}, a pause packet is sent. Contradictory, when queue length drops below X_{ON}, a resume packet is sent. They need T_{hop} to reach upstream hops and another T_{hop} to take effect downstream. This control delay imposes some constraints when setting these two thresholds. If the value of X_{OFF} is too high, the buffer overflow will occur. If the value of X_{ON} is too low, throughput loss will occur. In this case, a good value setting for X_{OFF} and X_{ON} should satisfy:

$$T_{hop}Bw \leq X_{ON} \leq X_{OFF} \leq B_t - T_{hop}Bw, \qquad (12)$$

where B_t denotes the total buffer for the port and Bw denotes the link bandwidth. This means we need to reserve at least two bandwidth-delay product (BDP) of buffer to prevent both queue overflow and underflow, and fully utilize bottleneck link bandwidth with a low latency. SWING [3] and Bifrost [29] halve the required buffer to a single BDP by periodically sending PFC frames. However, these solutions both require a short sending interval that needs to be times shorter than T_{hop}. So, their methods only apply to large-delay long-range cross-DC links and are infeasible for low-delay inter-DC links because of its large overhead.

In FAFC, we also reduce the required buffer to a single BDP, but with a different approach. By converting to time-slotted control and determining the pause time in the next time slot, we eliminate the need for resume packets. PFC frames include a pause time field, which indicates that its upstream port would resume automatically after this specified time. Commercial devices only set this field to the maximum value or zero, which stands for pause and resume packets respectively, so the capability of the pause time field has not been fully utilized. By adopting egress PFC, we can determine the required pause time when sending flow control pauses, the sending of resume packets can be skipped.

Eliminating resume packets also solves the blocking problem when the resume packet is corrupted during transmission and silently dropped by the receiver. In this situation, the port of the upstream switch will remain in a paused state for a long time as specified in the pause packet. This is similar to the PFC deadlock, which is also rare, but both can severely degrade the performance of DCN once it happens.

III. Implementation

In this section, we describe the implementation of FAFC in detail. It can be divided into three main algorithms: queue length estimation, pause time calculation, and ECN marking. We also discuss the setting of some parameters.

A. Estimation Algorithm

The estimation algorithm consists of two parts: the rate calculating and smoothing function $UpdateRate()$, which is triggered periodically; the queue prediction and congestion checking function $CheckPFC(pkt, i, e)$, which is triggered every time when packet is being sent.

α and β are exponential smoothing factors to attenuate noise and improve the stability of the value. To achieve fast response times, this value should be close to 1. We set both values to 0.8 in simulation. The time interval for prediction should be able to compensate for the propagation delay, so we have $t = T_{hop}$. To simplify calculating, the egress queue length and rate are collected separately even though they can be derived from individual statistics q_{ie} and r_{ie}.

Here we assume all links have the same delay and use a single T_{hop} for them in our algorithms. However, we can easily extend FAFC to handle heterogeneous topologies with different link delays by using separate T_{hop} for different links. By differencing T_{hop}, FAFC can even handle inter data center topologies with long-range links. The queue statistics collecting process still runs at the same frequency for all ports, which is determined by the link with the shortest delay.

B. Pause Time Calculation

When flow control pauses are triggered, the switch first determines the pause time for each ingress port by solving the linear optimization problem formulated in Section II. Then, it sends pause frame to all its upstream hops with a positive pause time. Here we choose the simplex method to solve the linear optimization problem. Most data center switches do not have a large port number. For example, a common ToR switch usually has ~48 ports. So the problem size is small and limited, and solutions can be got almost instantly.

Algorithm 1: Statistic collection and prediction

Parameter:
pause threshold th_p
ECN threshold th_e
Queue length smoothing factor α
Rate smoothing factor β

Data:
(smoothed) queue length q_{ie}
raw queue length q'_{ie}
ingress bytes counter b^i_{ie}
ECN offset of port e $ECNof_e$

Function $onEnqueue(pkt, i, e)$
  $q'_{ie} \leftarrow q'_{ie} + pkt.len$
  $b^i_{ie} \leftarrow b^i_{ie} + pkt.len$
  $onEnqueueECN(pkt, i, e)$
Function $onDequeue(pkt, i, e)$
  $q'_{ie} \leftarrow q'_{ie} - pkt.len$
  $CheckCong(pkt, i, e)$
  if $chkRsrv(e)$ **then**
    $onDequeueECN(pkt, i, e)$
    Start transmission
  else
    Hold packet and defer transmission
Function $CheckCong(pkt, i, e)$
  $q'_{ie} \leftarrow q'_{ie} - pkt.len$
  $qPred \leftarrow q_{*e} + 2r_{*e}T_{hop} + pausOf_{*e}$
  if $qPred > th_e + ECNof_e$ **then**
    $pkt.markECN()$
  if $qPred > th_p$ **then**
    $calculateAndSendPause(e)$
Function $UpdateRate()$
  for *every port* i **do**
    for *every port* e **do**
      $qEst \leftarrow q_{ie} + r_{ie}T_{hop}$
      $q_{ie} \leftarrow \alpha q'_{ie} + (1-\alpha)qEst$
      $r_{ie} \leftarrow \beta(q'_{ie} - qLast_{ie})/T_{hop} + (1-\beta)r_{ie}$
      $qLast_{ie} \leftarrow qr_{ie}$
      $r^i_{ie} \leftarrow \beta(b^i_{ie} - bLast^i_{ie})/T_{hop} + (1-\beta)r^i_{ie}$
      $bLast^i_{ie} \leftarrow b^i_{ie}$
  schedule next run after T_{hop}
Function $TimeSlotExpire(i)$
  $pausOld_i \leftarrow pausNext_i$
  $pausNext_i \leftarrow pausPending_i$
  $pausPending_i \leftarrow 0$
  if $pausOld_i \neq 0$ *and* $pausNext_i \neq 0$ **then**
    schedule next run after T_{hop}
  for *every port* e **do**
    $pausOf_{ie} \leftarrow (pausOld_i - pausNext_i) \times q_{ie}/q_{i*}$

Algorithm 2: Pause time calculation

Parameter: congested port threshold th_c
Data: egress rate of uncongested flows R_i

Function $calculateAndSendPause(e)$
  if $r_{*e} \leq 0$ **then**
    return
  for *every port* i **do**
    $R_i \leftarrow 0$
    for *every port* k *with* $q_{ik} \leq th_c$ **do**
      $R_i \leftarrow R_i + r^o_{ik}$
  if *last pause trigger of port* e *within* $4\ T_{hop}$ **then**
    Decrease r^i_{ie} for all i in later calculation by multiplying f_{dec}
  Find optimal T_i using simplex method with
    Object Function $\sum R_i T_i$
    Constraint $\sum r^i_{ie} T_i \geq r_{*e} T_{hop} + pausOf_{*e}$
    Bounds $0 \leq T_i \leq T_{hop}$
  for *every port* i *with* $T_i > 0$ **do**
    SendPFC(i, T_i)
    if $TimeSlotExpire(i)$ *is not scheduled* **then**
      $pausNext_i \leftarrow r^i_{i*} T_i$
      schedule $TimeSlotExpire(i)$ after t_{hop}
    else
      $tAvail \leftarrow \min(tsSch - tsCur, t_{hop} - pausNext/r^i_{i*})$
      $tNext \leftarrow \min(tAvail, T_i)$
      $pausNext_i \leftarrow pausNext_i + r^i_{i*} \times tNext$
      $pausPending_i \leftarrow pausPending_i + r^i_{i*}(t_{hop} - tNext)$
    for *every port* e **do**
      $pausOf_{ie} \leftarrow (pausOld_i - pausNext_i) \times q_{ie}/q_{i*}$

The pause frame in FAFC is slightly different from the traditional pause in PFC. It changes the pause time from a 16-bit quantum to a 32-bit value in nanoseconds, which both simplifies calculation and increases control precision. Besides, the later pause frames accumulate pause time together with the former pause frames rather than overriding them. Aside from these differences, we can convert these two values to each other at the cost of minor performance loss. Hence, FAFC can coexist with traditional PFC switches and be incrementally deployed. The pause frame in FAFC will also cause the sending node to decrease its sending rate to achieve fast congestion avoidance.

C. ECN marking algorithm

The flow control pause usually causes upstream switches to mark ECN for packets going through that port because its queue length exceeds the ECN threshold during the pause. However, as shown in [30], those ECN markings at upstream switches usually do not indicate real congestion. Therefore, they can also cause throughput loss for uncongested flows.

Algorithm 3: Pause handling

```
Function PauseReceive(e, T)
    Pause the sending of port e for T
    if Device is endpoint then
        Decrease sending rate of all flows by
          multiplying 1 − T/T_hop
```

To prevent this situation, switches should not account for the queue buildup resulting from flow control pauses during ECN marking decisions. In FAFC this is done by applying a positive offset $ECNof$ to the original ECN threshold. $ECNof$ is initialized to zero, when the port is paused by a downstream switch, every new packet enqueue will increase $ECNof$ by the corresponding packet length. After this port is resumed, $ECNof$ becomes the minimal queue length seen until it goes back to zero. This ensures that packets buffered at upstream switches would not receive ECN marking during the flow control pause, which effectively eliminates false congestion marking.

Algorithm 4: ECN offset mechanism

```
Function onEnqueueECN(pkt, i, e)
    if Port e is paused by downstream switch then
        ECNof_e ← ECNof_e + pkt.len
Function onDnqueueECN(pkt, i, e)
    if ECNof_e = 0 then
        return
    ECNof_e ← max(0, min(q_*e, ECNof_e))
```

D. Overreact Prevention

It is possible that multiple pauses for a specific ingress port are triggered in a single time slot. This usually happens at the beginning of the incast. To prevent overreaction, new pause frames should consider the influence of pause frames sent previously.

After the pause is triggered, the predicted queue length in the per-hop RTT should decrease to provide an accurate estimation. To address this demand, FAFC will increase $pausNext$ after sending the flow control pause, and that value is deducted after the subsequent prediction. If multiple pauses happen in the same time slot, the last pause may not be fully effective in the next time slot. Instead, they can pause traffic in the next two time slots, and we use $pausPending$ to indicate the residue after the next time slot. Also, paused bytes will be added back to the predicted value in the next time slot to reflect the resumption of the upstream hop, which is represented by the handling of $pausOld$ in our algorithm.

In FAFC, both pause frames and ECN marking will cause sending nodes to decrease their sending rate, which may also cause overreaction as we have an overestimated ingress rate. To address this problem, FAFC employs a rate decrease factor f_{dec} to r^i_{ie} in the pause time calculation algorithm when the same egress port triggers pauses consecutively within $4\,Thop$.

TABLE III
FCT SLOWDOWN OF DUMBBELL TOPOLOGY

Scheme	Average FCT slowdown		
	All	Incast flows	Large flows
∞-BUF	16.088512	17.586524	1.857397
PFC	17.142368	18.697751	1.866665
FAFC	16.104492	18.750172	1.868229

IV. PERFORMANCE EVALUATION

We evaluate the performance of FAFC by using the ns-3 simulator [31] with some modules adapted from the HPCC simulator [32]. This simulator implements RDMA NIC, shared-buffer switch, and PFC mechanism, which are required for simulating data center environments. It also includes various data center congestion control schemes including DCTCP, DCQCN, TIMELY [33], and HPCC. We implemented the linear programming solver by linking the ns-3 simulator against the GNU Linear Programming Kit (GLPK) library, which implements the simplex method. Then, we use GLPK functions to define and solve linear programming directly in switches during simulation. We also replaced FIFO scheduling with port-based weighted fair queuing (WFQ) scheduling in the switches to improve the fairness of FAFC between the ingress ports, as our optimization problem assumes near equal service rate for each ingress port.

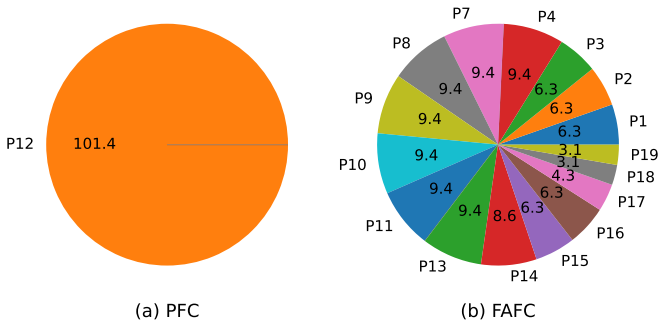
A. Functional Evaluation

Setup: We used the same dumbbell topology discussed in Section II to evaluate the function of FAFC. There are 5 hosts HA1–5 connected to the first switch SW1, 18 hosts HB1–16, HC1–2 connected to the second switch SW2. All links are 25 Gbps with a propagation delay of 1 µs. Two Hosts, named HA1 and HA2, send long-lived large flows with a size of 60 MB to HC1 and HC2 separately. Other hosts send a burst to HC2 simultaneously after the start of long-lived flows to simulate an incast.

We compare FAFC with two baseline configurations: one with infinite buffer switches and uses end-to-end congestion control methods only to handle congestion (∞-BUF), the other configuration uses standard PFC with static XON and XOFF thresholds on the ingress queue length (PFC). For both configurations, we chose DCTCP as the end-to-end congestion control method and applies 200 kB as ECN threshold. For FAFC, the triggering threshold is set to 350 kB. We set the PFC threshold to 50 kB, as it triggers right after the egress queue length increases to about 350 kB.

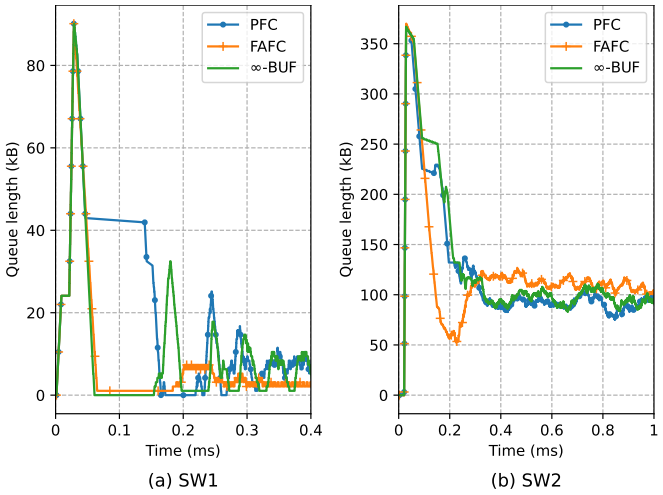
First, we collect the average FCT slowdown for small incast flows and long-lived large flows separately. Result in TABLE III shows that FAFC achieves lower slowdown for both incast and long flows compared to PFC. FAFC also outperforms the infinite buffer case due to its proactive reaction to congestion.

We also collect and plot the total paused time for each switch port during incast events to see which port is paused in each scheme. In this case, only the second switch SW2 has its queue length exceeded thresholds and triggered flow control



(a) PFC (b) FAFC

Fig. 3. Total pause time of each port in SW2.



(a) SW1 (b) SW2

Fig. 4. Total buffer occupancy in SW1 and SW2.

pauses. So, only the total pause time of ports within SW2 is shown in the plot, as P1-P19. P5 and P6 are connected to HC1 and HC2 separately. They are receivers and do not send other flow, so they should have zero pause time. P12 corresponds to the link between SW1 and SW2. The result in Fig. 3(a) shows that PFC always pauses the link between two switches, as it always has the largest queue length during congestion. On the contrary, FAFC distributes pause time among all the congested links, as shown in Fig. 3(b). The link between two switches receives minimal pausing time during incast events, so the uncongested flows are protected, and its FCT is greatly reduced.

Last, Fig. 4(a) and Fig. 4(b) shows the total buffer occupancy of switches against time. FAFC has successfully lowered the queue length sharply of the switch SW2 after being congested and all the three schemes both have a peak buffer occupancy of about 350 kB. For the upstream switch SW1, the stage of high queue length utilization after the triggering of flow control pauses is eliminated, which protects the uncongested flows.

B. Testbed Implementation

We implemented most of the functions of the FAFC on Tofino P4 programmable switches to verify its feasibility. The queue length is implemented in the data plane using the P4 register extern, as well as the rate calculation using the

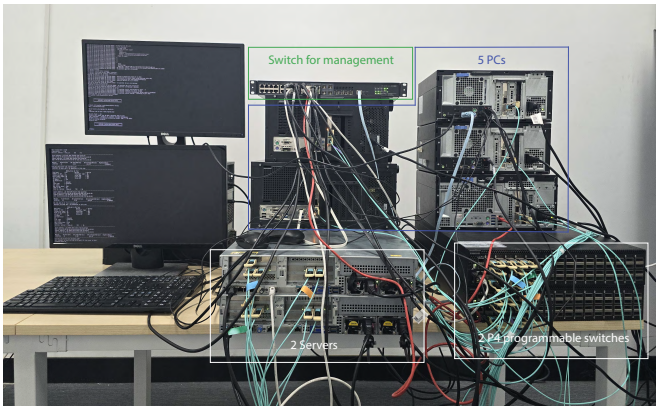


Fig. 5. Data center testbed with programmable switches.

TABLE IV
LATENCY OF A 24:1 INCAST IN TESTBED

Scheme	Latency of 65 kB message (µs)		
	Average	99% percentile	99.9% percentile
PFC	32.35	33.66	38.80
FAFC	32.27	32.44	32.56

Tofino lpf (low pass filter) extern. When the predicted queue length exceeds the pause threshold, the data plane notifies the control plane running in the switch CPU using the P4 digest extern. The control program, written in BfRt C++ API, periodically gathers queue status and processes the digest from the data plane. It constructs and solves the linear programming problem, and sends pause frames to the data plane using the CPU PCIe interface.

As shown in Fig. 5, our testbed includes two Ingrasys S9280-64X, which are based on Intel Tofino programmable ASICs with a reconfigurable match table (RMT) architecture [34], and 7 hosts with multiple RDMA capable NICs including Mellanox ConnectX-5 and Intel E810. We used the same dumbbell topology in functional evaluation with a total of 28 nodes. Each node is simulated by a single port on the NIC. Each link is configured with 25 Gbps speed and uses DCQCN [35] as the end-to-end congestion control method. We use multiple ib_write_bw instances to simulate the incast traffic and ib_write_lat to measure the delay of the victim flows.

First, we show the resource usage of the FAFC pipeline. Our testbed implementation, capable of tracking queue statistics for 16 ports, uses 3 MAU stages, 10% of SRAM without using any TCAM. Since the switch only has 16 QSFP28 ports in a single pipeline, FAFC can track all the ports with small resource usage if no breakout is used.

Second, we measured the average processing time of each digest in the control plane by taking nanosecond timestamps at both the beginning and the end of the pause time calculation process. The switch uses an Intel Xeon D-1527 CPU, and the average processing time is 30.6 µs, which is about 100 kB queuing delay in 25 Gbps. Thus, the solving of linear programming is fast enough for 25 Gbps and lower link rates.

Finally, we conduct a 24:1 incast and test the latency of victim flows. Latency is measured using 64 kB messages and

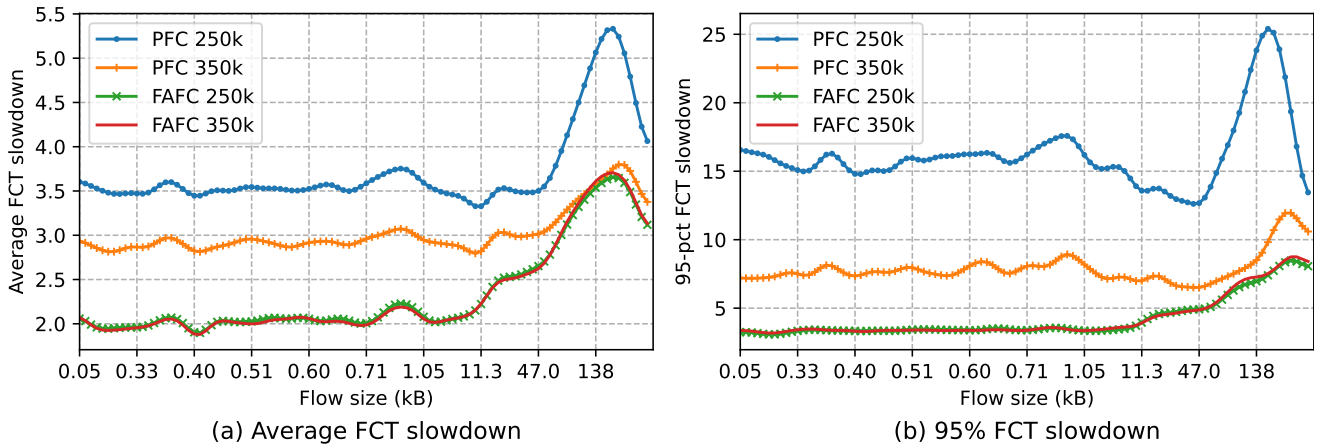


Fig. 6. FCT Slowdown of PFC and FAFC under Hadoop workload with TIMELY.

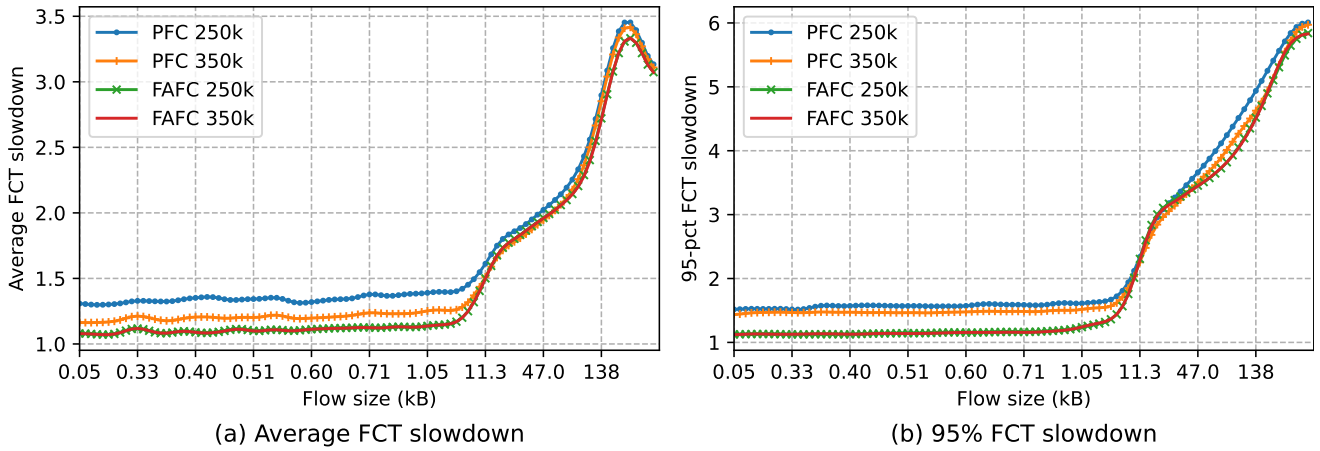


Fig. 7. FCT Slowdown of PFC and FAFC under Hadoop workload with HPCC.

1000 iterations with an ECN threshold of 200 kB and a flow control threshold of 1 MB. The result in TABLE IV shows that FAFC achieves lower latency compared to PFC.

However, there are some limitations in the Tofino implementation. First, the dequeue operation and the queue statistics are only observable in the egress pipeline, however, FAFC tracks the queue in the ingress pipeline, which cannot access the required information directly. So, we need to mirror and recirculate every packet once to collect queue statistics in the data plane. Besides, each pipeline tracks its own values separately, another recirculation is required to notify the local queue statistics to other pipelines. Our testbed implementation uses ports in a single pipeline to avoid excessive recirculations. Those recirculations are only needed as workarounds of the current RMT-based programmable switches. They would be unnecessary if the FAFC queue tracking mechanism is directly implemented in the traffic manager. Besides, the BfRt C++ API lacks vectorized access to register arrays, the control plane can only access them one by one, which is inefficient and greatly limits the update rate of queue statistics. This can be solved by bypassing the API and talking to the hardware directly, but this requires extensive work and a deep knowledge of the switch ASIC.

We used a spine-leaf topology with 288 hosts, 12 aggregation switches, and 6 core switches. There are 24 hosts connected to each aggregation switch. All links to hosts are 25 Gbps and links to core switches are 100 Gbps. We use FCT slowdown as the performance metric and plot it against flow size. For each run, we also collect the distribution of the switch buffer occupancy. We compare the performance of FAFC to traditional PFC using two different triggering thresholds: 250 kB and 350 kB, with the ECN threshold setting to 150 kB and 200 kB respectively. It is worth noting that ECN is not used in HPCC and TIMELY.

C. Large-scale Simulation

First, we compare the performance using different workloads. We choose Facebook Hadoop [36] and Google RPC (GRPC) [37] workloads as they have plenty of short-lived flows, which represent the flow characteristics in data centers. These workloads are randomly generated using a load factor of 30% according to their corresponding cumulative distribution function (CDF) distributions. Apart from the workload, incast flows are also generated periodically with randomly selected 60 source nodes and one destination node, with 500 kB of data and contribute 2% of the network capacity. The performance is evaluated under different congestion control schemes. For

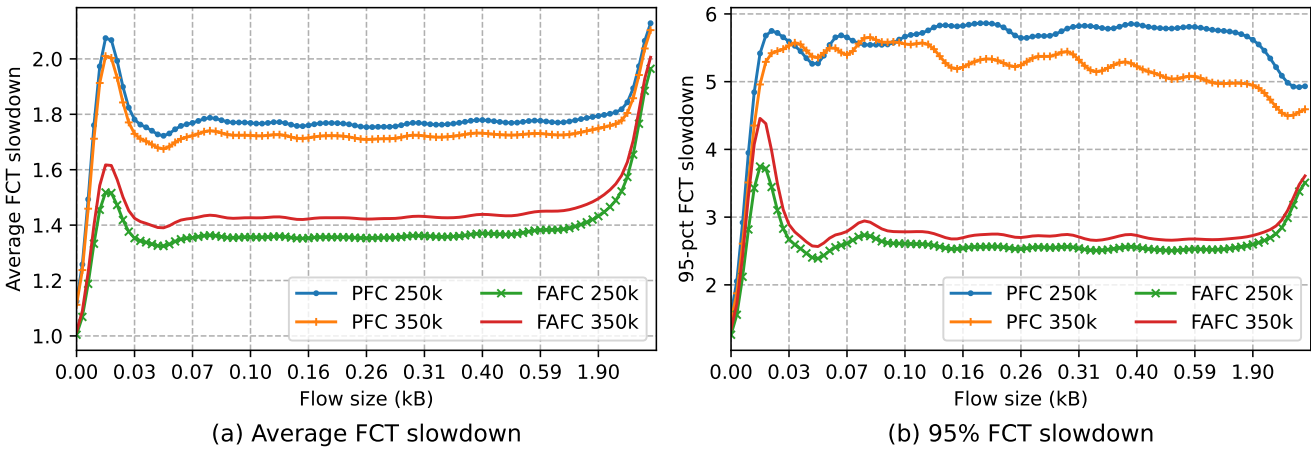


Fig. 8. FCT Slowdown of PFC and FAFC under GRPC workload with DCTCP.

the Hadoop workload, we use TIMELY and HPCC. For the GRPC workload, we use DCTCP.

Fig. 6(a) and Fig. 6(b) shows the average and 95% FCT slowdown against the flow size of the Hadoop workload using TIMELY CC. The flow size is plotted unevenly to match the actual flow distribution. For small flows, FAFC achieves about a 34.5% decrease on average FCT slowdown and a 47.5% decrease on 95% FCT slowdown compared to PFC. For large flows, FAFC still has better performance on FCT slowdown. For FAFC, lowering the triggering threshold does not degrade its performance, but for PFC, lowering the triggering threshold greatly impacts the performance.

Fig. 7(a) and Fig. 7(b) shows the average and 95% FCT slowdown against flow size of the Hadoop workload using HPCC. Although HPCC has a FCT slowdown very closing to its theoretical minimum 1, FAFC can still achieve about a 10.6% decrease on average FCT slowdown and a 23.3% decrease on 95% FCT slowdown for small flows, with no performance loss on large flows.

Fig. 8(a) and Fig. 8(b) shows the average and 95% FCT Slowdown of the GRPC workload using DCTCP CC. The GRPC workload has numerous small flows that would finish in a single RTT, so the FCT slowdown graph has a peak when the flow length is near zero. For other flows, FAFC achieve about a 21.1% decrease on average FCT slowdown, and a 49.1% decrease on 95% FCT slowdown.

Second, we plot the CDF of buffer occupancy for congested ports during incast events. Fig. 9 shows that FAFC increases the probability of low buffer occupancy compared to PFC, and for both PFC and FAFC lower trigger threshold also decreases the queue length. PFC with 250 kB threshold has more probability near zero queue length, indicating that throughput loss is more likely to happen.

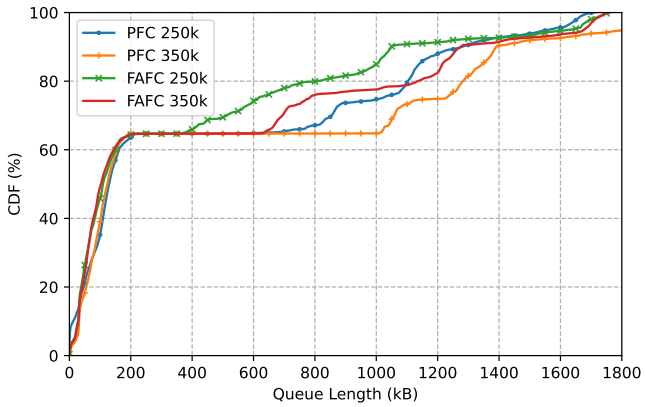


Fig. 9. Buffer Occupancy CDF of PFC and FAFC under Hadoop workload with HPCC.

V. Conclusion

This study conducts a pioneering systematic analysis of large-scale many-to-one traffic challenges in modern data centers, focusing on switch buffer exhaustion and spurious flow-control pauses affecting non-congested traffic. Our findings reveal that when large-scale flow synchronization occurs, severe delay spikes in data center victim flows due to successive triggering of flow control pauses. To this end, we developed FAFC, an innovative switch-level control mechanism that dynamically allocates port-specific pause times during congestion. The solution employs first-order prediction of queue statistics to proactively activate flow control, enabling rapid congestion mitigation and queuing delay reduction. Through optimized pause-time allocation, FAFC strategically transfers egress congestion to ingress ports while minimizing throughput loss for uncongested flows. Extensive system-level simulations and physical testbed experiments demonstrate FAFC's superiority to flexibly assign pause times across congested ports. Compared to traditional PFC for Hadoop workloads and HPCC's end-to-end congestion control method, FAFC achieves superior performance with 10.6% and 23.3% improvements in average and 95% FCT, respectively.

In future work, we plan to extend our prediction and pause time calculation algorithms to support multi-queue scenarios, as performance isolation is crucial in multi-tenant data centers. This can be done by exploiting more statistical information such as the link capacity of the physical link and the bandwidth utilization of other queues and then include them in the objective function. In addition, to make FAFC more efficient and lightweight, we will conduct more experiments to investigate the performance and complexity of different pause

time calculation algorithms such as greedy allocation.

REFERENCES

[1] T. Chen, X. Gao, and G. Chen, "The features, hardware, and architectures of data center networks: A survey," *J. Parallel Distrib. Comput.*, vol. 96, pp. 45–74, 2016.

[2] A. Singh *et al.*, "Jupiter rising: a decade of clos topologies and centralized control in Google's datacenter network," *Commun. ACM*, vol. 59, no. 9, pp. 88–97, 2016.

[3] Y. Chen *et al.*, "Swing: Providing long-range lossless RDMA via pfc-relay," *IEEE Trans. Parallel Distrib. Syst.*, vol. 34, no. 1, pp. 63–75, 2023.

[4] Y. Li *et al.*, "HPCC: high precision congestion control," in *Proc. ACM Special Interest Group Data Communication (SIGCOMM)*, Beijing, China, Aug. 19–23, 2019, pp. 44–58.

[5] I. T. Association, "Infiniband architecture specification volume 2 release 1.3.1," 2016. [Online]. Available: https://cw.infinibandta.org/document/dl/8125

[6] I. T. Association, "Infiniband architecture specification volume 1 release 1.4," 2020. [Online]. Available: https://cw.infinibandta.org/document/dl/8567

[7] J. Woodruff, A. W. Moore, and N. Zilberman, "Measuring burstiness in data center applications," in *BS '19: 2019 Workshop on Buffer Sizing, Stanford University, Palo Alto, CA, USA*, Dec. 2–3, 2019, pp. 5:1–5:6.

[8] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph, "Understanding TCP incast throughput collapse in datacenter networks," in *Proc. 1st ACM SIGCOMM 2009 Workshop Res. Enterprise Netw.*, Barcelona, Spain, Aug. 21, 2009, pp. 73–82.

[9] Y. Gao, Y. Yang, C. Tian, J. Zheng, B. Mao, and G. Chen, "DCQCN+: taming large-scale incast congestion in RDMA over ethernet networks," in *26th IEEE Int. Conf. Netw. Protocols (ICNP)*, Cambridge, UK, Sep. 25–27, 2018, pp. 110–120.

[10] V. Addanki, O. Michel, and S. Schmid, "PowerTCP: Pushing the performance limits of datacenter networks," in *19th USENIX Symp. Networked Syst. Des. Implementation (NSDI)*, Renton, WA, USA, Apr. 4–6, 2022, pp. 51–70.

[11] S. Arslan, Y. Li, G. Kumar, and N. Dukkipati, "Bolt: Sub-RTT congestion control for ultra-low latency," in *20th USENIX Symp. Networked Syst. Des. Implementation (NSDI)*, Boston, MA, USA, Apr. 17–19, 2023, pp. 219–236.

[12] B. Che *et al.*, "FCC : A fast-converging low-latency congestion control algorithm for datacenter RDMA network," in *8th Asia-Pacific Workshop Netw., (APNet)*. Sydney, Australia: ACM, Aug. 3–4, 2024, pp. 200–201.

[13] B. Yan, Y. Zhao, S. Xu, J. Liu, and H. Xu, "LHCC: low-latency and hi-precision congestion control in RDMA datacenter networks," in *32nd IEEE/ACM Int. Symp. Qual. Service, (IWQoS)*. Guangzhou, China: IEEE, Jun. 19–21, 2024, pp. 1–10.

[14] T. Wang, Y. Zhang, A. Zhou, and S. Wang, "An enhancement framework for RDMA congestion control in multi-tenant datacenters," in *2024 IEEE Netw. Operations Manage. Symp. (NOMS)*. Seoul, Republic of Korea: IEEE, May 6–10, 2024, pp. 1–9.

[15] Q. Meng *et al.*, "BCC: re-architecting congestion control in dcns," in *2024 IEEE Conf. Comput. Commun. (INFOCOM)*. Vancouver, BC, Canada: IEEE, May 20–23, 2024, pp. 1441–1450.

[16] M. Alizadeh *et al.*, "Data center TCP (DCTCP)," in *Proc. ACM SIGCOMM 2010 Conf. (SIGCOMM)*, New Delhi, India, Aug. 30 – Sep. 3, 2010, pp. 63–74.

[17] G. Kumar *et al.*, "Swift: Delay is simple and effective for congestion control in the datacenter," in *Proc. Annu. Conf. ACM Special Interest Group Data Communication Appl. technologies architectures protocols Comput. communication (SIGCOMM)*, Virtual Event, USA, Aug. 10–14, 2020, pp. 514–528.

[18] B. E. Stephens, A. L. Cox, A. Singla, J. B. Carter, C. Dixon, and W. Felter, "Practical DCB for improved data center networks," in *2014 IEEE Conf. Comput. Commun. (INFOCOM)*, Toronto, Canada, Apr. 27 – May 2, 2014, pp. 1824–1832.

[19] C. Guo *et al.*, "RDMA over commodity ethernet at scale," in *Proc. 2016 ACM SIGCOMM Conf.*, Florianopolis, Brazil, Aug. 22–26, 2016, pp. 202–215.

[20] "Pause flood protection," accessed: Jul., 2024. [Online]. Available: https://www.hpe.com/psnow/resources/ebooks/a00111821en_us_v6/s_logical-switches-pauseflood-about.html

[21] K. Qian, W. Cheng, T. Zhang, and F. Ren, "Gentle flow control: avoiding deadlock in lossless networks," in *Proc. ACM Special Interest Group Data Communication (SIGCOMM)*, Beijing, China, Aug. 19–23, 2019, pp. 75–89.

[22] S. N. Avci, Z. Li, and F. Liu, "Congestion aware priority flow control in data center networks," in *2016 IFIP Netw. Conf., Networking 2016 and Workshops*, Vienna, Austria, May 17–19, 2016, pp. 126–134.

[23] C. Tian *et al.*, "P-PFC: reducing tail latency with predictive PFC in lossless data center networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 6, pp. 1447–1459, 2020.

[24] P. Goyal, P. Shah, K. Zhao, G. Nikolaidis, M. Alizadeh, and T. E. Anderson, "Backpressure flow control," in *19th USENIX Symp. Networked Syst. Des. Implementation (NSDI)*, Renton, WA, USA, Apr. 4–6, 2022, pp. 779–805.

[25] W. Li, C. Zeng, J. Hu, and K. Chen, "FlowSail: Fine-grained and practical flow control for datacenter networks," *IEEE/ACM Trans. Netw.*, vol. 32, no. 5, pp. 3916–3928, 2024.

[26] J. Hu, Y. He, J. Wang, W. Luo, and J. Huang, "RLB: reordering-robust load balancing in lossless datacenter networks," in *Proc. 52nd Int. Conf. Parallel Process. (ICPP)*, Salt Lake City, UT, USA, Aug. 7–10, 2023, pp. 576–584.

[27] W. Cheng, K. Qian, W. Jiang, T. Zhang, and F. Ren, "Re-architecting congestion management in lossless ethernet," in *17th USENIX Symp. Networked Syst. Des. Implementation (NSDI)*, Santa Clara, CA, USA, Feb. 25–27, 2020, pp. 19–36.

[28] N. Luangsomboon and J. Liebeherr, "Necessary and sufficient condition for triggering ECN before PFC in shared memory switches," *IEEE Netw. Lett.*, vol. 6, no. 2, pp. 119–123, 2024.

[29] P. Yu *et al.*, "Bifrost: Extending RoCE for long distance inter-DC links," in *31st IEEE Int. Conf. Netw. Protocols (ICNP)*, Reykjavik, Iceland, Oct. 10–13, 2023, pp. 1–12.

[30] Y. Zhang, Q. Meng, Y. Liu, and F. Ren, "Revisiting congestion detection in lossless networks," *IEEE/ACM Trans. Netw.*, vol. 31, no. 5, pp. 2361–2375, 2023.

[31] "The ns3 simulator," accessed: Jul., 2024. [Online]. Available: https://www.nsnam.org/

[32] "HPCC simulation," accessed: Jul., 2024. [Online]. Available: https://github.com/alibaba-edu/High-Precision-Congestion-Control/

[33] R. Mittal *et al.*, "TIMELY: RTT-based congestion control for the datacenter," in *Proc. 2015 ACM Conf. Special Interest Group Data Communication (SIGCOMM)*, London, UK, Aug. 17–21, 2015, pp. 537–550.

[34] P. Bosshart *et al.*, "P4: programming protocol-independent packet processors," *Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, 2014.

[35] Y. Zhu *et al.*, "Congestion control for large-scale RDMA deployments," in *Proc. 2015 ACM Conf. Special Interest Group Data Communication (SIGCOMM)*, London, UK, Aug. 17–21, 2015, pp. 523–536.

[36] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *Proc. 2015 ACM Conf. Special Interest Group Data Communication (SIGCOMM)*, London, UK, Aug. 17–21, 2015, pp. 123–137.

[37] B. Montazeri, Y. Li, M. Alizadeh, and J. K. Ousterhout, "Homa: a receiver-driven low-latency transport protocol using network priorities," in *Proc. 2018 Conf. ACM Special Interest Group Data Communication (SIGCOMM)*, Budapest, Hungary, Aug. 20–25, 2018, pp. 221–235.