Traffic-aware Buffer Management in Shared Memory Switches

Sijiang Huang, Mowei Wang, Yong Cui*
Department of Computer Science and Technology, Tsinghua University, China

IEEE INFOCOM 2021 - IEEE Conference on Computer Communications | 978-1-6654-0325-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/INFOCOM42981.2021.9488849

***Abstract*—Switch buffer serves an important role in modern internet. To achieve efficiency, today's switches often use on-chip shared memory. Shared memory switches rely on buffer management policies to allocate buffer among ports. To avoid waste of buffer resources or a few ports occupy too much buffer, existing policies tend to maximize overall buffer utilization and pursue queue length fairness. However, blind pursuit of utilization and misleading fairness definition based on queue length leads to buffer occupation with no benefit to throughput but extends queuing delay and undermines burst absorption of other ports. We contend that a buffer management policy should proactively detect port traffic and adjust buffer allocation accordingly. In this paper, we propose Traffic-aware Dynamic Threshold (TDT) policy. On the basis of classic dynamic threshold policy, TDT proactively raise or lower port threshold to absorb burst traffic or evacuate meaningless buffer occupation. We present detailed designs of port control state transition and state decision module that detect real time traffic and change port thresholds accordingly. Simulation and DPDK-based real testbed demonstrate that TDT simultaneously optimizes for throughput, loss and delay, and reduces up to 50% flow completion time.**

I. Introduction

Switch buffer is used to absorb burst traffic and improve the overall performance of the switch. Insufficient buffer results in reduced port throughput [1], thereby impairing the network quality of service [2]–[4]. To achieve high buffer efficiency, the majority of today's switches adopt on-chip shared memory, instead of private memory that exclusively allocated for each port [2], [5]–[7].

Shared memory switches rely on specific buffer management policies to allocate buffer among different ports [2]. Without buffer management policies, a few ports can occupy as much as the entire shared buffer space, blocking other ports from benefitting from the shared memory, resulting in severe unfairness between switch ports [7]. To avoid unfairness, the simplest way is to split the total buffer evenly and allocate it as private buffer for each port. However, the problem is that ports only have access to its exclusive buffer space. This limits buffer utilization when only a few ports are active, which is against the basic principle of shared memory that buffer should be dynamically shared among ports. For a long time, researchers have believed that an ideal buffer management should be somewhere in between complete sharing (no control) and complete partitioning (evenly split), with high buffer utilization as well as port fairness [7]–[9].

Many buffer management policies have been proposed in the past three decades [7]–[15] to allocate buffer in shared memory switches. Although the design concepts and methods of different schemes are diverse, the design goals of these schemes are basically consistent, that is, pursuing the highest possible buffer utilization, reducing packet loss and maximizing throughput, while ensuring queue length fairness between ports. Since it was proposed over twenty years ago, the dynamic threshold (DT) [7] policy has been used as the default buffer management scheme by switch manufacturers [2] and various congestion control related researches [5], [16]. Besides several variants of DT [8]–[10], no significant progress had been made until 2019, Stanford University hosted a workshop specifically discussing buffer sizing [17] that re-emphasizes the importance of buffering and its management in the network.

Existing buffer management policies fail to make full use of the shared buffer due to misleading optimization goals: buffer utilization [8] and queue length fairness [7]. More specifically, when long-lived over-line-speed traffic arrives at a switch port, blind pursuit of buffer utilization leads to buffer occupation that does not contribute to port throughput but extends queuing delay. On the other hand, misleading fairness definition based on queue length impedes meaningless buffer occupation from being evacuated, impairing burst absorb capacity of other ports through additional buffer. Besides, queue length fairness ignores the demand difference between different ports. Fairness in queue length is not equivalent to fairness in throughput.

In light of the limitations of existing buffer management policies, we aim to propose a buffer management policy that optimize for metrics that actually have impacts on the quality of service, i.e., throughput, loss and delay. Designing such a policy can be challenging because it requires switch ports to proactively detect port traffic and adjust buffer allocation accordingly. More specifically, switch ports have to determine the type of traffic it is transmitting in a timely manner with limited port level information.

In this paper, we propose Traffic-aware Dynamic Threshold (TDT), a buffer management policy that controls the buffer allocation of shared memory switches by detecting port traffic status in real time. TDT can fully utilize the shared buffer to absorb burst traffic, avoid meaningless buffer occupation through proactive evacuation, and ensure the throughput fairness among ports at the same time. By using buffer only when actually needed, TDT is friendly to loss-sensitive burst traffic, throughput-sensitive long-lived traffic and delay-sensitive short traffic in different switch ports simultaneously.

TDT uses a set of port-wise control states to differentiate the status of different ports, and impose different thresholds to

* Yong Cui (cuiyong@tsinghua.edu.cn) is the corresponding author.

978-1-6654-0325-2/21/$31.00 ©2021 IEEE

individual ports accordingly. The transition between different port control states is determined by a state decision module that detects port traffic in real time. The state decision module of TDT is only composed of several counters, comparators and triggers, avoiding dependency on the assumption of time variables. When port traffic changes, TDT proactively raise or lower port threshold according to port control state to allocate buffer among different ports in a dynamic manner.

We evaluate TDT with ns-3 simulation and a DPDK-based switch prototype, comparing TDT with existing buffer management policies. Simulation results demonstrate that when long-lived over-line-speed traffic exists, TDT can absorb ∼60% additional burst traffic while maintaining a similar overall throughput and reducing overall queuing delay in the switch, comparing to current policies. Experiments on real DPDK testbed show that TDT reduces 12% average flow completion time (FCT). Specifically, TDT outperforms DT over 80% of the times and has an up to 50% reduction in FCT. In summary, we make the following contributions:

- The first shared buffer management policy to the best of our knowledge that optimizes for the problem of meaningless buffer occupation (§III).
- A detailed design of threshold determination based on port-wise control states and state decision module (§IV).
- Comprehensive evaluation based on large-scale simulation and real testbed implementation (§V).

The rest of this paper is organized as follows: §II introduces the background of our research and limitations of current buffer management policies. §III overviews the key design ideas of traffic-aware buffer managements. §IV gives the design details of our possible version of traffic-aware buffer management, TDT. Evaluation based on simulation and DPDK testbed is presented in §V. Finally, §VI concludes this paper.

II. Background and Motivation

In this section, we first introduce current buffer management policies. Then, we illustrate the limitations of existing strategies as the motivation of our research.

A. Background

Shared memory switch. Figure 1 shows the architecture of a typical shared memory switch. Shared memory switches are output-queued switches with high speedup switch fabric that can process packets from any input port almost immediately and send them to designated output port queue [6]. To achieve buffer efficiency, every output port can access the shared memory pool which means theoretically any single port can use as much as the entire buffer space. In practice, shared memory switch rely on a specific buffer management policy deployed on a control module (state decision module in Figure 1) to allocate buffer for each output port.

Buffer management policies. Several buffer management policies [7]–[15] have been designed to allocate buffer among ports in shared memory switches. Current policies have two main design goals. First, buffer should be fairly distributed among different ports, i.e., no port is "starved" because a few

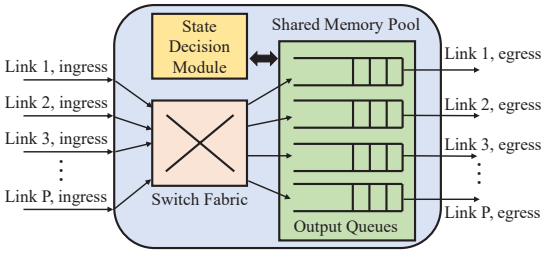


Fig. 1. The architecture of a shared memory switch.

ports have occupied too much buffer [7]. And second, buffer utilization should be as high as possible, i.e., free buffer space when packets are dropped should be as small as possible [8].

In general, existing buffer management policies can be divided into two categories: preemptive policies and non-preemptive policies. For preemptive policies [13], [15], packets that already in the memory can be overwritten, or "pushed out", by newly arrived packets. Preemptive policies have been proved to be optimal in certain circumstances [13], [15] whereas they are very hard to be implemented in practice due to hardware limitations [7], [8]. On the other hand, non-preemptive policies [7]–[12], [14], which only allow packets to be dropped before entering the port queue, are more practical to be deployed on actual devices. To achieve fairness, non-preemptive policies often use thresholds to restrict the amount of buffer each port can have access to. In this paper, we only consider non-preemptive policies in design for practicality.

Dynamic threshold policies. Among a wide variety of non-preemptive policies, dynamic threshold policies [7], [8], [10] are most widely used by switch vendors [2] due to their simplicity of deployment. In dynamic threshold policies, queue length of every switch port is restrained by a threshold, which dynamically changes with switch status indicators (e.g., remaining buffer space). Classic dynamic threshold scheme (DT) [7] sets its threshold, which is shared by all ports, proportional to the current amount of unoccupied buffer space. More specifically, threshold at time t can be calculated by

$$T(t) = \alpha \cdot (B - \sum_i Q_i(t)) \tag{1}$$

where $T(t)$ is the threshold at time t, B is the total buffer size, $Q_i(t)$ is the queue length of port i at time t, and α is a control function normally set to a constant value for simplicity. To avoid unfairness when traffic changes, DT reserves a certain amount of buffer in "stable state" (i.e., when queue length is equal to port threshold).

On the basis of the classic dynamic threshold scheme, in recent years, researchers have proposed several variants to cope with specific problems in networking. A typical one is the Enhanced Dynamic Threshold (EDT) [8]. EDT improves the burst absorbing ability of DT by temporally relaxing threshold restraint to absorb microburst traffic in data center networks. Although dynamic threshold policies have made huge success in the past decade, we will demonstrate the limitations of existing dynamic threshold policies in the next part.

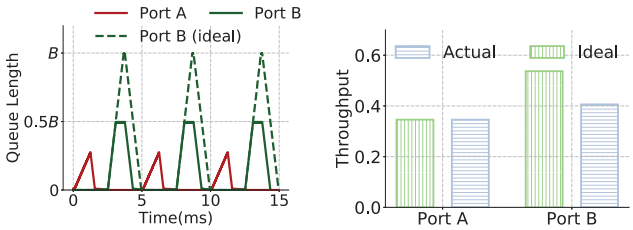
B. Motivation

The optimization goal of current buffer management strategies is to maximize total throughput, minimize overall packet loss while maintaining port fairness [7], [8], [10], [15]. In order to optimize the metrics above, current policies focus on maximizing buffer utilization and fairness based on queue length, for the reasons that "drop packet only when inevitable" intuitively means fewer packet losses, and "different queue has similar chances to reach the same queue length" intuitively equals to fair buffer distribution among different port queues.

Unfortunately, neither of these intuitions is actually correct. In the following, we show why blindly pursuing buffer utilization or using queue length to measure fairness actually results in meaningless buffer occupation, which further leads to performance degradation.

Blind pursuit of utilization. Existing buffer management schemes contend that buffer should be as full as possible when switch port is overloaded, otherwise buffer is not fully utilized [7], [8], [12]. This opinion comes from ancient cognitive habit that if resources are not used, they are wasted. However, resources that are not wasted are not equivalent to resources that are beneficial. Due to the transmission rate limitation of the switch port, switch buffer management is a problem in which the law of diminishing marginal utility [18] is very obvious. For an overloaded port (i.e., a port transmitting an aggregate flow whose rate is over port line speed), throughput limitation comes from port line speed instead of available buffer. In that case, additional buffer allocated for this port brings little benefit to overall throughput. Although high buffer utilization in this case does not directly lead to performance degradation, when multiple ports are competing for buffer, high utilization potentially impair the absorption capacity of ports that became active later. On the other hand, maintaining high buffer utilization results in longer port queues, which prolongs the queueing delay of delay-sensitive packets.

Misleading definition of fairness. Fairness is one of the most important considerations when designing a buffer management policy. Current policies either use descriptive definition of fairness [7] or use queue length of competing ports to measure fairness [8]. A buffer management scheme is considered fair if different ports that start active at different times can achieve similar queue length. Under this definition of fairness, the fairest allocation strategy is evenly split, i.e., the shared buffer pool is equally distributed among all output ports as private buffer. However, the above fairness description ignores the difference in demand between ports and only limit the fairness of the queue length, an intermediate variable, without considering the fairness of metrics that actually matter (e.g., throughput). For example, Figure 2 shows a simple example generated by simulation where two ports with different flow rate competing for the whole buffer space. With the fairness definition based on queue length, the resource (buffer) should be equally divided between these two ports. Port A receives full additional throughput with the help of buffer while port B can only partially benefit from the buffer. Obviously, relative



(a) Queue length fairness. (b) Throughput unfairness.

Fig. 2. Queue length fairness is not equivalent to throughput fairness. Same queue length threshold shared by ports with different throughput demand causes unfairness in throughput.

to the ideal throughput, port B is affected more than port A, as a result of buffer allocation. Therefore, in terms of throughput, fairness measurement based on queue length is not always fair. This misleading definition of fairness has caused unnecessary restrictions in the design of existing schemes. Given the reasons above, in this paper we will directly use throughput as a measure of fairness.

Meaningless buffer occupation. As a result of blind pursuit of utilization and misleading definition of fairness, current strategies tend to maintain relatively longer queues in active ports, keeping average queue length at a high level. However, as discussed above, buffer occupation is not a sufficient condition for neither high throughput or fairness. On the contrary, maintaining long port queues undermines the overall capacity of the switch to absorb potential burst traffic, and extends the overall queuing delay of the switch [5]. Therefore, reducing meaningless buffer occupation is a vital step toward better buffer management policies. In order to achieve that, a buffer management policy needs to be able to determine whether or not the buffer occupation at a specific time is beneficial. In conclusion, there is an urgent need for a buffer scheduling strategy that can perceive changes in the traffic status of switch ports and take actions accordingly.

III. Traffic-aware Buffer Management

Discussion in §II-B have shown that the key factor causing performance degradation is buffer occupied by ports overwhelmed by long-lived over-line-speed traffic. Solving this problem requires a buffer management policy that can determine whether a switch port is transmitting long-lived over-line-speed flows or short bursty flows. In this section, we present the key ideas of traffic-aware buffer management. By real time detection of port traffic, a traffic-aware buffer management policy is expected to fully utilize buffer when absorbing burst traffic (§III-A) or proactively evacuate meaningless buffer occupation (§III-B) when traffic changes.

A. Burst absorption

In the context of this paper, burst traffic refers to fast (transmitting at over the port line rate) but short (no longer than switch buffering time[1]) flows that arrive at switch port [19].

[1]Switch buffering time is the time switch can buffer packets arriving at the line speed of the port, e.g 1MB buffer on a 16-port switch can be equivalently expressed as 500us buffer per port with 1Gbps line rate.

In other words, we only consider burst that can be fully absorbed if it can monopolize all the buffer. Otherwise, buffer management policies is futile for this flow because no possible allocation can achieve lossless transmission. Burst traffic caused by "incast" scenario often carries deadline-sensitive short messages [5], [20]. Retransmissions triggered by packet loss will result in the service carried by this flow missing its deadline [4], [21]. It can be concluded that this sort of burst is loss-sensitive, which means the optimization goal is lossless transmission. Once packet loss occurs the absorption fails so the number of packet loss, except zero, is irrelevant.

For a single burst, buffer management policy should leverage as much as possible buffer space to avoid packet loss. In this paper, we share a similar basic design idea to absorb burst traffic as EDT [8]. When burst traffic arrives at a port, port queue length should not be restrained by the normal threshold. If the burst traffic transmission is completed or exceeds the buffer capacity of the switch, buffer management should quickly react to traffic changes by putting the corresponding ports back under the normal threshold control.

The most difficult part of design is to determine whether a port has started or finished transmitting burst and anticipate whether buffer can fully absorb that burst. Traffic detection is expected to be sensitive and accurate, otherwise the buffer management policy will not be able to change the threshold in a timely fashion, resulting in inability to fully absorb burst traffic or a single port occupying too much buffer for too long.

Based on the above discussion, we propose several conditions for traffic status judgement. First, we define several traffic states to represent typical port traffic status. When a port is idle or transmitting under-line-speed traffic, it is "underloaded". "Overloaded" port refers a port that transmitting short burst flows. And a port transmitting long-lived over-line-speed flows is called an "overwhelmed" port. Conditions for judgement of port traffic state are listed as follows:

- A port becomes "overloaded" when burst traffic arrives at this port. When a port becomes "overloaded", its queue length increases while no packets are dropped [8].
- A port becomes "underloaded" when over-line-speed traffic transmission has completed. When a port becomes "underload", there are consecutive packet dequeue events between packet enqueue events.
- A port becomes "overwhelmed" when over-line-speed traffic exceeds buffer capacity. Indications for a port being "overwhelmed" are buffer overflow and consecutive packet drops.

Burst absorption only applies to "overloaded" ports. Because extra buffer can only be of help when there is possibility to fully absorb burst traffic. Extra buffer is needless to "Underloaded" ports and useless to "overwhelmed" ports. Burst absorption should be terminated as soon as an "overloaded" port becomes "underloaded" or "overwhelmed". *Port traffic state* will be used to determine *port control state*. Detail designs of port control states will be shown in §IV-A.

B. Proactive evacuation

When a port is transmitting long-lived over-line-speed traffic, no amount of buffer can absorb the part of traffic that exceeds line speed. It is worth pointing out that typical long-lived over-line-speed traffic is not common in switch for the purpose of congestion control algorithm (CC) [16], [22]–[24] is to avoid continuous overload scenarios. As a very recent cencus [25] have shown, TCP CUBIC [24] remains the dominant TCP variant on the internet and undocumented TCP variants have occupied a non-negligible proportion on the internet, some of which may focus more on maximizing link utilization when facing competition instead of maintaining a relative short queue like TCP BBR [23], making buffer occupany by long-lived traffic a reasonable concern. In the context of this paper, long-lived traffic is a relative concept, which refers to aggregated flows whose duration exceeds switch buffering time. It is possible that the duration of an aggregated flow fits the description of long-lived traffic due to untimely response or temporal failure of CC [16], [22]. The rationality of this description is that if a flow exceeds the buffer capacity of the switch, even if its absolute duration is relatively short, from the perspective of the switch, this flow is sufficient to be considered as long-lived.

When a port is transmitting long-lived over-line-speed traffic, port serving rate cannot match arriving rate, causing rapid growth in port queue length. Once the queue length matches the port threshold, it will remain stable until transmission of the whole flow has completed and consecutive packet drop will occur due to threshold limitation. However, as discussed in §II-B, buffer occupation in this scenario does not benefit port throughput but seriously reduces the amount of buffer that can be used by other ports and extend overall queuing delay. Therefore, if a port is transmitting long-lived over-line-speed traffic, its port queue should be proactively evacuated, for the purpose of avoiding meaningless buffer occupation.

Proactive evacuation can be achieved with a simple mechanism. When a switch port determines that the traffic its transmitting is long-lived and over-line-speed, it proactively lower its port threshold in order to reduce its queue length. It is worth noting that queue length reduction in this circumstance dose not affect port throughput. When the transmission is over, buffer management policy should quickly restore its threshold to normal threshold for fairness consideration.

Proactive evacuation only applies to "overwhelmed" ports. For the reason that additional buffer to "overwhelmed" ports does not bring extra throughput, whereas long-term buffer occupied by these ports potentially impedes other ports from leveraging the same buffer space to absorb bursts, and increases the delay caused by queuing. Proactive evacuation should never be activated unless there is complete confidence that a port will remain in "overwhelmed" state for a relatively long time thus port throughput is determined by port line speed, notwithstanding the reduction of queue length. Once a port is no longer in "overwhelmed" state, proactive evacuation should be terminated instantly to avoid throughput loss.

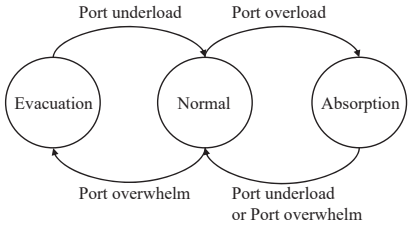


Fig. 3. Port control state transition diagram of TDT.

IV. Traffic-aware Dynamic Threshold

In this section, we present a possible design of traffic-aware buffer management policies, Traffic-aware Dynamic Threshold (TDT). On the basis of classic dynamic threshold [7], TDT controls port thresholds by assigning a traffic state indicator to each port and detecting traffic changes in a time-independent way. Our design controls threshold on the port level, without maintaining flow-level information [14], which gives our design natural scalability. Moreover, Parameter tuning of TDT is more convenient than existing ones, because it only considers the parameters of the switch itself, and avoids the use of network state-related parameters (e.g., burst duration as in EDT [8]). At the end of this section, a simple but typical example is given to show how TDT works.

A. Port-wise control state

In TDT, each switch port has three possible control states. When the traffic is relatively mild, the port is in **"normal" state** and controls its queue length by the shared dynamic threshold as in equation (1). When a port is transmitting burst traffic, port state transits to "absorption", temporally raising the threshold to absorb burst traffic as much as possible. On the other hand, when a port is transmitting long-lived over-line-speed traffic, the port turns to "evacuation" state that proactively evacuate the port queue by lowering the port threshold. State transition diagram is shown in Figure 3.

"Absorption" state. In "absorption" state, port threshold is the total amount of buffer divided by the total number of ports in "absorption" state. "Absorption" state is triggered by port becoming "overloaded", which indicates that the port is transmitting burst traffic. Therefore, trigger condition for "absorption" control state is the same as the judgement condition of port becoming "overloaded", that is: *port queue length increases while no packets are dropped.*

For fairness consideration, a port should quickly lower the threshold and return to the "normal" state when additional buffer is no longer needed. Under two circumstances additional buffer is no longer beneficial: first, the port has finished transmitting burst traffic and second, the size of traffic exceeds buffer capacity. In other words, switch should cease buffer control in the "absorption" manner when a port has transit from "overloaded" state to either of the other two states.

For the former condition, two possible scenarios exist: a) After the burst traffic transmission is completed, no traffic continues to be transmitted on this port, or the rate of traffic that continues to be transmitted is low, which is manifested as *consecutively packets dequeue from the port queue*; b) After the burst traffic transmission is completed, there is still traffic of a certain rate (within the line speed of the port) being transmitted in the port. In that case, queue length will decrease slowly, resulting in the port occupying excessive buffer for a long time after the completion of the burst traffic. *Cumulative packets dequeue from the port queue* should be restricted to ensure timely return to the "normal" state.

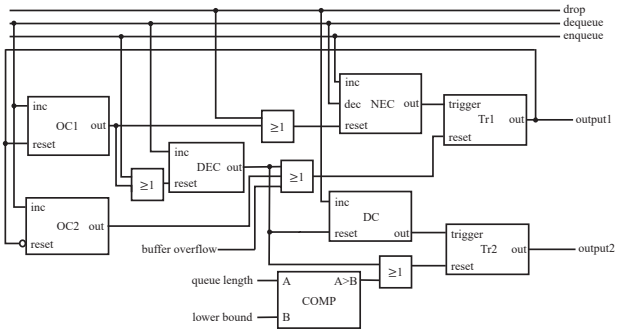


Fig. 4. Circuit logic of state decision module.

For the latter condition, even the entire buffer is not enough to avoid packet drop, which means that this "burst" traffic is actually long-lived, at least from the perspective of switch buffer, and the port is actually in "overwhelmed" state. In that circumstance, *buffer overflow event* will occur, indicating a state transition back to the "normal" state.

"Evacuation" state. In "evacuation" state, port threshold is the total amount of buffer divided by the total number of ports. "Evacuation" state is triggered by *consecutive drop events*, which indicates the port is transmitting traffic that cannot be absorbed by buffer, i.e., long-lived over-line-speed traffic. A port should return from "evacuation" state to "normal" state once the long-term traffic has finished transmitting and port traffic state becomes "underloaded". *Consecutively packets dequeue from the port queue* can indicate port being "underloaded" for most cases. We use a lower bound on queue length for "safety reassurance" in case over-line-speed traffic is followed by traffic transmitting at slightly below port line speed so that there might not be consecutive packets dequeue.

B. Time-independent state decision

To be directly deployed on high speed switches, buffer management policies should be able to directly interact with port signals. This means circuit level design is needed for a practical buffer management policy. TDT uses a state decision module in each output port to detect traffic state and decide port control state. State decision module leverages signals of packets enqueue, dequeue and drop in the port, as well as state information such as queue length and buffer overflow signal. Two-bit output is generated by the state decision module whenever any of the above signals triggers a state decision. Next state of the port is determined by the two-bit output and the following mapping rules: (0, 0) → "normal" state; (1, 0) → "absorption" state; (0, 1) → "evacuation" state. The state decision module consists of three parts: absorption decision, evacuation decision and restoration decision. The circuit logic

design of port state decision module is shown in Figure 4. Port state decision module is composed by several counters, triggers and comparators. Notice that timers are not part of state decision module. This is an important design choice in TDT that we will explain later.

Absorption decision. The core part of absorption decision is the Net Enqueue Counter (NEC) that keeps tracking the amount of net enqueue packet. When NEC reaches it threshold, it will trigger the first trigger (Tr1) to inform a state transition from "normal" state to "absorption" state. Function of NEC is to detect rapid queue length increasing in the condition that no packets are dropped, which means there should be several reset conditions for NEC if the above requirements are not met. First, if packet drop event occurs, remaining buffer is not sufficient to absorb burst traffic, thus NEC should be reset for future burst detection. Second, if the first output counter (OC1) reaches its threshold, indicating at least a certain amount of time has passed before NEC reaches its threshold, it can be assumed that the arriving rate of traffic does not meet the speed requirement of burst traffic, thus NEC should also be reset. Notice that OC1 serves a similar role of determining a "time out" event, yet we use an output counter instead of a timer.

Evacuation decision. The Drop Counter (DC) is the core of evacuation decision. The value of DC keep adding with packet drop events in each port. When the value of DC reaches its threshold, a signal will be sent to trigger the second trigger (Tr2) to transit port control state to "evacuation". "Evacuation" state should only be triggered by consecutive packet drop events instead of long-term accumulation of packet dropping. Therefore, when Dequeue Counter (DEC) detects consecutive packet dequeue events, it will send a signal to reset DC to restart the counting.

Restoration decision. When a port is in the "absorption" state or "evacuation" state for a long time, it may affect the ability of itself or other ports to maintain line-speed transmission, causing fairness problem. When a port is in "absorption" state, there are three possible conditions indicating state restoration: 1) DEC reaches its threshold, indicating burst traffic has finished transmitting; 2) the second output counter (OC2) reaches its threshold, indicating the port has already transmitted burst traffic of the maximum possible size allowed by the switch; 3) buffer overflow, which is the sign that the buffer is unable to fully absorb the burst traffic.

On the other hand, if a port is in "evacuation" state, there are two conditions for state restoration: 1) DEC reaches its threshold, indicating that the port is no longer "overwhelmed"; 2) queue length is less than a lower bound. The latter condition serves the role of a "safety reassurance" if the over-line-speed traffic is followed by traffic transmitting at a rate slightly lower than the line speed. In that case, DEC might not reach it threshold yet evacuation should be terminated whereupon queue length is used for restoration decision.

Time-independent design. As mentioned above, unlike existing method [8], TDT avoids the use of timer in the design of state decision module. Instead, TDT uses two output counters to indicate "time out" event. Two major reasons are behind this design choice. First, output counters can adjust the trigger interval according to the change of port traffic. More specifically, if a port is overloaded, port queue cannot remain empty, in this case, output port will definitely transmit at maximum speed, i.e., port line speed. Output counter is equivalent to a timer in that case. Second, the timer timeout setting usually needs to consider the changes of the network environment which means parameter of the timer often needs retuning when network condition changes. On the contrary, the parameter of output counters is directly related to switch buffer settings, which means using output counters can help achieve designs independent to the often inaccurate assumptions on specific network environments.

C. Convenient parameter setting

In TDT, several parameters needs tuning before deployment. As discussed above, TDT avoids the use of time-related parameters. Only threshold of several counters and one input of one comparator requires setting. We assume that the line rate of all output ports are equal in the same switch for simplicity. The switch has a buffer size of B and n ports.

NEC and OC1 together make absorption decision thus their thresholds should be set together. NEC should detect rapid queue length growth, thus a smaller NEC threshold means TDT is more sensitive to small burst and larger NEC would mean otherwise. In practice, the evenly split threshold B/n is good enough for most cases. OC1 limit the speed of burst traffic. Burst traffic with speed less than a certain value cannot trigger NEC because OC1 will reset NEC in advance. If we set that the rate of burst traffic should be at least k-times the line speed of the port, then the threshold of OC1 should be set to $\frac{1}{k-1}$ of the NEC threshold. In practice, rate of bursts from incast scenarios should be at minimum twice the port line-speed, so we can simply set OC1 threshold to B/n.

Evacuation decision relies on DC. DC is used to detect consecutive packet drop events. Consecutive packet drops can be a result of long-live over-line-speed traffic or temporal traffic changes. Therefore, the threshold should be large enough to avoid frequent triggers. When α of DT is set to 1, at most half of the total buffer is reserved when a port reaches its threshold [7]. The number of consecutive packet drops exceeds half of the total buffer indicates the traffic cannot be absorbed even with the whole buffer and for the switch this traffic is enough to be consider to be long-lived. Base on that we conservatively set the threshold of DC to $B/2$.

Restoration decision is based on DEC, OC2, and the input of the comparator. DEC detects consecutive packet dequeue events, which is a characteristic behavior rather than quantitative behavior. Therefore, DEC can be set to a small number such as 3 or 5. OC2 limits the maximum size of burst. For burst traffic of twice the port line rate, the whole buffer can absorb burst with size up to twice the buffer space. The amount of packets in a burst that the buffer can absorb decreases as the traffic rate increases. For most scenarios, we can set the threshold of OC2 to $2B$. Finally, the lower bound of queue

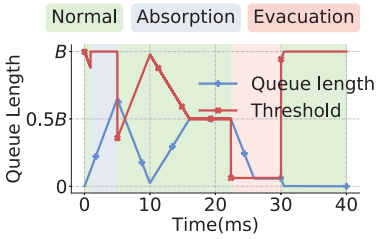


Fig. 5. An example of queue length and threshold evolution of TDT.

length is a "safety reassurance" for state restoration, which should not be too sensitive considering in most cases the end of long-lived traffic should be signaled by DEC instead of the comparator. Setting it to half the evacuation threshold, i.e., $1/2 \times B/n = B/2n$ is sufficient for most situations.

D. Putting it all together

In this part, we will use a simple but typical example to show how TDT works. First, a burst starts at time $t = 0$ and finishes at $t = 5ms$, then a long-lived slightly over-line-speed traffic starts at $t = 10ms$ and continues over-line-speed transmission until $t = 30ms$. Figure 5 shows the evolution of queue length and corresponding threshold.

Port starts at "normal" control state, threshold decreases as queue length increases. When state decision module detects burst traffic by rapid queue length growth, it instructs the port to change its state to "absorption" to raise its threshold to the total amount of buffer, ensuring the absorption of traffic burst. After the burst transmission has finished, state decision module makes a restoration decision and put the port control back to "normal". When the second flow arrives at $t = 10ms$, it does not trigger "absorption" state because its arriving rate does not meet the speed requirement of a burst. Then this queue length of this flow is control by typical dynamic threshold as in "normal" state. When queue length converges with threshold, it stops increasing and packet drop occurs. When cumulative packet drop has reached a certain amount, state decision module decide that this port is in "overwhelm" traffic state, thus control state should transit to "evacuation" and threshold undergo a cliff-like drop while queue length decreases gradually due to the restriction of port transmission speed. After the long-lived over-line-speed traffic has finished transmitting, state decision module quickly let "normal" state retake control and raise the threshold for future traffic.

V. Evaluation

In this section, we compare the performance of TDT and existing buffer management policies with large scale simulation based on ns-3 network simulator[2] [26] and deep dive into the reasons behind performance gain of TDT. We use DT [7] and EDT [8] as comparison schemes of TDT. Evenly Split (ES) and Complete Sharing (CS) are also used as auxiliary baselines. We also implement TDT and its comparison dynamic threshold policies in real DPDK testbed and compare their performance of flow completion times.

[2]Vanilla ns-3 simulator does not support shared memory switch architecture and threshold control. We implement total buffer control and port queue threshold control by adding static functions and buffer-related variables to ns-3 source code using C++ programming.

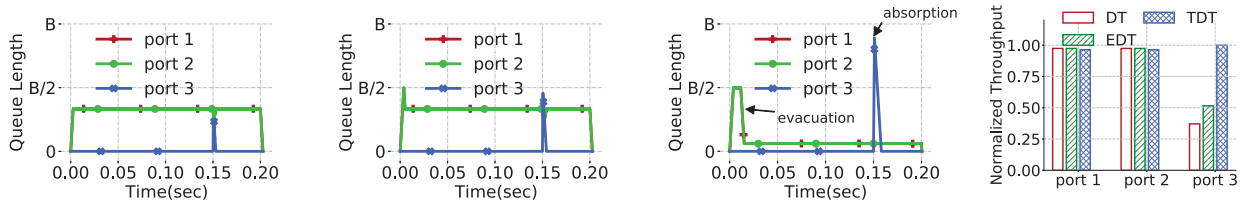
A. Micro-benchmarks

In this section, we consider a 16-port shared memory switch with 1MB buffer and 1Gbps port line speed [8], [27]. Packet size is set to 1500 bytes which makes the total buffer size 667 packets. We use 1 as the control factor α of DT, EDT and "normal" state of TDT as suggested in [7]. Parameters of EDT are set according to [8] as $C_1 = 3$, $C_2 = 8$, $TM_1 = 2.1ms$, $TM_2 = 10ms$. As for TDT, threshold of NEC, DEC, DC, OC1 and OC2 are set to 42, 3, 333, 42 and 1344, respectively, according to suggested parameter setting in §IV-C.

Burst absorption. First, we use a simple scenario with long-lived over-line-speed traffic and burst traffic to demonstrate how TDT improves the absorption capacity of burst traffic by proactive evacuation. Queue length evolution of TDT and its comparison policies are shown in Figure 6. The first two ports transmits 2Gbps traffic throughout the entire $0.2s$ experiment. The third port start transmitting 8Gbps burst traffic at $t = 0.15s$ and finish $1ms$ later. As shown in Figure 6(a), due to strict queue length fairness restriction of DT, packet drop of burst occurs almost immediately when queue length of port 3 reaches the queue length of the first two ports. EDT improves its burst absorption capacity by relaxing threshold restriction for port 3, as shown in Figure 6(b). Unfortunately, burst absorption of EDT stops just after a while when buffer overflow occurs, as a consequence of meaningless buffer occupation of port 1 and port 2. As for TDT in Figure 6(c), early proactive evacuation of queues of port 1 and port 2 pays off when burst traffic arrives at port 3. TDT allows ports that transmit bursty traffic to use up to the entire buffer without being limited by the evacuation rate of ports occupying buffer.

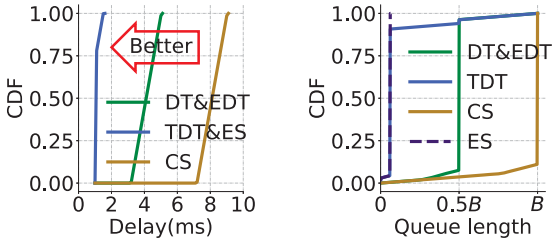
It may come as a surprise that proactive evacuation makes TDT a "fairer" policy, in terms of throughput of different port. As shown in Figure 6(d), TDT achieves nearly the optimal throughput in all three ports whereas its comparison policies fail to bring similar additional throughput for port 3 as for the first two ports. By getting rid of meaningless queue length restrictions, TDT can fairly provide as much additional throughput as possible for different ports.

Delay reduction. We use a different micro-benchmark to demonstrate TDT's improvement on delay-sensitive traffic and the reasons behind. Figure 7 shows the delay results of single active port transmitting 2Gbps traffic from $t = 0$ to $t = 0.2s$ and a delay-sensitive flow starts right after $t = 0.2s$, with 0.8Gbps arriving rate and lasts for $10ms$. We focus on the delay of the second flow. As shown in Figure 7(a), TDT have an identical delay performance as ES, which outperforms DT and EDT by a large margin. CS is considered as the upper bound for delay. The reason behind lower delay of TDT is straight-forward, reduction of average queue length. Queuing delay is determined by the total amount of packets in queue when a packet arrives at a port. As shown in Figure 7(b), by proactive evacuation of meaningless buffer occupation, TDT



(a) DT. (b) EDT. (c) TDT. (d) Throughput.

Fig. 6. Queue length evolution of two ports transmitting long-lived over-line-speed traffic and burst traffic arrives at the third port at 0.15s.



(a) CDF of delay. (b) CDF of queue length.

Fig. 7. TDT reduce delay by keeping a relatively short queue length.

can reduce unnecessary queue length in ports transmitting a lot of long-term traffic without affecting its throughput.

B. Large scale simulation

We test TDT and its comparison policies on large-scale stochastic scenarios in this section. Switch and parameter settings are identical as §V-A. Following to the experiment settings in [7] and [8], we consider the following scenario: Among 16 output ports, 8 of them are transmitting mixed traffic which is composed of poisson background traffic with 20% average load and burst traffic with 8Gbps arriving rate, $250us$ average "on" time and $19.75ms$ average "off" time. The total average load of is 30%. We test the lossless burst absorption capability of these ports with 0,1 or 2 additional port(s) transmitting long-lived over-line-speed traffic and discuss the performance of TDT and its comparison policies in terms absorption of loss-sensitive burst, throughput of long-lived traffic and delay of light load short traffic.

Loss-sensitive. When no port is transmitting long-lived over-line-speed traffic, as shown in Figure 8(a), TDT can achieve a similar lossless ratio as EDT because in that case hardly any port needs evacuation. DT achieves poor absorption performance due to strict threshold restriction. However, as shown in Figure 8(b) and Figure 8(c), when overwhelmed ports exist, performance of EDT drops dramatically while performance of TDT remains basically unchanged. Especially, in the case of two overwhelmed ports, almost all burst traffic longer than $0.5ms$ cannot be absorbed using DT or EDT, whereas TDT achieves an over 80% lossless ratio when burst are no longer than $0.75ms$ and an over 50% lossless ratio otherwise. In general, the overall lossless ratio of Figure 8(c) are 5.5%, 57.8%, 92.7%, respectively, which means TDT has a $\sim$60% performance gain than the best performing existing policies when it comes to absorption of loss-sensitive burst traffic.

Delay-sensitive. Figure 8(d) focus on the delay distribution of background traffic. Notice that in Figure 8(d) only delay of background traffic in overwhelmed ports is presented, since delay in ports absorbing burst traffic cannot be optimized simultaneously as burst absorption capability on the port level. For DT and EDT, most of the time queue length is equal to port dynamic threshold, i.e., half the total buffer space when only one port is overwhelmed and α of DT is 1, which causes a half-buffering-time queuing delay for packets. CS always allow a single port to occupy as much as the entire buffer. For an overwhelmed port, this means queue length is as long as possible within the total buffer limit, causing a queuing delay close to buffering time for packets. With proactive evacuation, TDT can keep the average queue length similar to ES, resulting in less than $2ms$ queuing delay for more than 90% packets.

Throughput-sensitive. As discussed in §III-B, proactive evacuation of TDT should not affect the throughput of long-lived over-line-speed traffic. Figure 8(e) shows the throughput of the long-lived over-line-speed flow. All three dynamic threshold policies have similar throuput results. EDT achieves a slight lower throughput than DT and TDT due to its time-dependent design which potentially leads to reletively long-term buffer occupation of ports transmitting burst traffic that further causes throughput degradation of overwhelmed ports.

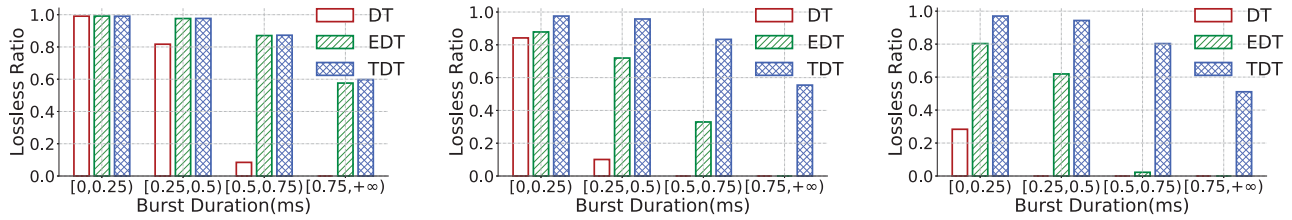
Fairness. TDT is fair among different ports on the long run because when multiple ports are active, TDT ensures each port has a similar chance to obtain additional throughput. We use the throughput fairness combined with Jain's Fairness Index [28] to measure fairness numerically. Figure 8(f) shows the overall fairness index of different ports. All policies actually have similar fairness indexes except CS, which has no queue length restriction and is considered the most "unfair" policy. Let us zoom in on the fairness attenuation[3] of three dynamic threshold policies. TDT is "fairer" than DT and EDT because it can achieve closer to the ideal throughput[4] on ports with burst traffic while achieve nearly ideal throughput on ports transmitting long-lived traffic.

C. TDT in a DPDK-based switch

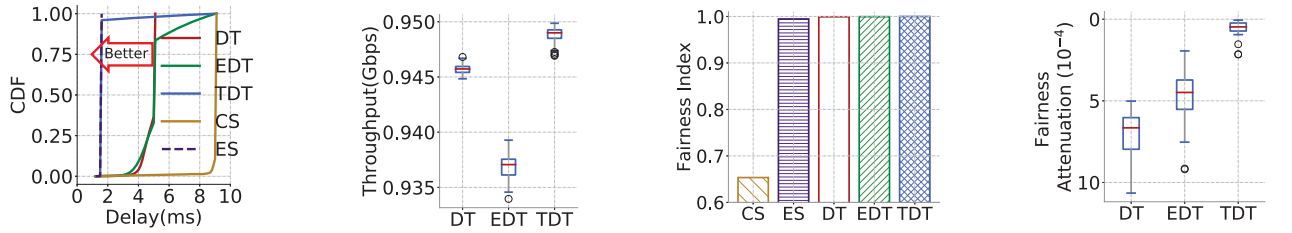
We implement TDT and its comparison buffer management policies on a DPDK [30] testbed with 4 host servers connecting to an emulating switch with four ports. The server

[3]This metric is proposed in [29] based on the concept of measuring relative performance degradation.

[4]Ideal throughput is obtained by assuming every port can monopolize the entire shared buffer.

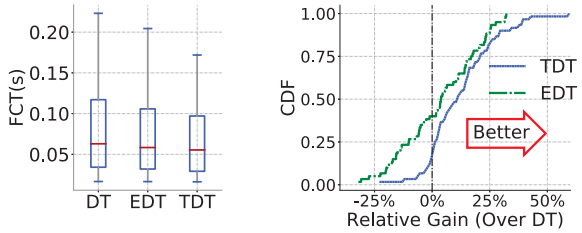


(a) Burst absorption with no overwhelmed port. (b) Burst absorption with one overwhelmed port. (c) Burst absorption with two overwhelmed ports.



(d) Delay of background traffic. (e) Throughput of long-lived traffic. (f) Overall fairness. (g) Zooming in on fairness attenuation.

Fig. 8. Performance of TDT and its comparison policies in large scale stochastic simulation.



(a) Mean FCT. (b) CDF of relative gain over DT.

Fig. 9. TDT reduces FCT on DPDK testbed.

emulating the switch has a 28-core Intel Xeon E5-2660 2.00GHz CPU, along with 32GB of memory, a hard disk with 2TB storage, and four Intel 82599ES 10-Gigabit Ethernet NICs. We use Ubuntu 16.04 LTS GNU/Linux kernel 4.15.0 as the operating system. The other four servers have 4-core Intel i3-3220 3.30GHz CPU and an Intel 82599ES 10-Gigabit Ethernet NIC. Buffer size of the switch server is set to 256KB and we use a rate limiter to limit the sending rate of each port to 1Gbps. TCP CUBIC [24] is used as the congestion control algorithm [31]. We use an empirical traffic generator [32] to generator traffic of simple client/server application. In our experiment, we use the WebSearch flow size distribution given in [5]. Two active ports are set to have different offered load. A port has 40% average load while the other is overwhelmed over 100% average load. We focus on the flow completion time (FCT) [33] of the first port under the influence of the overwhelmed port. Using different random seed, we repeat the experiment 50 times and report the average FCT. We have generated over 20,000 flows in total.

The parameters of EDT and TDT are adjusted according to port number and buffer size. The threshold of dequeue counter (C1 in EDT, DEC in TDT) are set a little larger because TCP data flow has relatively larger fluctuations. Mean FCT results of TDT and its comparison policies are shown in Figure 9(a). On average, TDT reduces the flow completion time by 12% compared to DT, and 8% compared to EDT. As shown in Figure 9(b), more than 80% of the time, TDT performs better than DT, and the maximum gain can reach up to 50%. EDT also outperforms DT, but not as much as TDT. The reason for TDT's better performance is its time-independent feature and proactive evacuation design. TDT reserves more buffer for burst absorption by proactive evacuation and accurately determine port traffic type in a timely manner. On the contrary, the time-dependent design makes it difficult for EDT to accurately determine the arrival and departure of burst traffic, and the buffer occupied by the overwhelmed port limits the other port's ability to absorb burst traffic.

VI. Conclusion

In this paper, we demonstrate that buffer occupation caused by blind pursuit of buffer utilization and misleading definition of port fairness does not contribute to throughput but undermines other metrics. To avoid meaningless buffer occupation, we propose the design concepts of traffic-aware buffer management policy and a specific policy, TDT, that simultaneously optimize for loss-sensitive burst traffic, throughput-sensitive long-lived traffic and delay-sensitive short traffic. By utilizing the buffer as much as possible to absorb burst traffic and proactively evacuate meaningless buffer occupation, TDT can fully exploit the benefit of buffer. We test TDT on the ns-3 simulator and a real DPDK switch prototype. Experimental results show TDT outperforms existing policies in terms of burst absorption, throughput, delay and FCT of TCP flows, while maintaining throughput fairness among ports.

Acknowledgment

This work is supported by NSFC (No. 61872211), National Key R&D Program of China (no. 2018YFB1800303, 2017YFB1010002).

REFERENCES

[1] G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing router buffers," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4, pp. 281–292, 2004.

[2] S. Das and R. Sankar, "Broadcom smart-buffer technology in data center switches for cost-effective performance scaling of cloud applications," 2012.

[3] "Intelligent buffer management on cisco nexus 9000 series switches white paper." https://www.cisco.com/c/en/us/products/collateral/switches/nexus-9000-series-switches/white-paper-c11-738488.html, accessed August 15, 2020.

[4] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav *et al.*, "Conga: Distributed congestion-aware load balancing for datacenters," in *ACM SIGCOMM*, 2014.

[5] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," in *ACM SIGCOMM*, 2010.

[6] U. Cummings, A. Lines, P. Pelletier, and R. Southworth, "Shared-memory switch fabric architecture," Oct. 12 2010, uS Patent 7,814,280.

[7] A. K. Choudhury and E. L. Hahne, "Dynamic queue length thresholds for shared-memory packet switches," *IEEE/ACM Transactions On Networking*, 1998.

[8] D. Shan, W. Jiang, and F. Ren, "Analyzing and enhancing dynamic threshold policy of data center switches," *IEEE TPDS*, 2017.

[9] H. Yousefi'zadeh and E. A. Jonckheere, "Dynamic neural-based buffer management for queuing systems with self-similar characteristics," *IEEE Transactions on Neural Networks*, 2005.

[10] E. L. Hahne and A. K. Choudhury, "Dynamic queue length thresholds for multiple loss priorities," *IEEE/ACM Transactions On Networking*, 2002.

[11] A. Kesselman and Y. Mansour, "Harmonic buffer management policy for shared memory switches," *Theoretical Computer Science*, 2004.

[12] G. Ascia, V. Catania, and D. Panno, "An evolutionary management scheme in high-performance packet switches," *IEEE/ACM Transactions On Networking*, 2005.

[13] S. X. Wei, E. J. Coyle, and M.-T. Hsiao, "An optimal buffer management policy for high-performance packet switching," in *IEEE GLOBECOM*, 1991.

[14] M. Apostolaki, L. Vanbever, and M. Ghobadi, "Fab: Toward flow-aware buffer sharing on programmable switches," in *Online Program: Workshop on Buffer Sizing*, 2019.

[15] I. Cidon, L. Georgiadis, R. Guerin, and A. Khamisy, "Optimal buffer sharing," *IEEE Journal on Selected Areas in Communications*, 1995.

[16] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel, M. H. Yahia, and M. Zhang, "Congestion control for large-scale rdma deployments," *ACM SIGCOMM*, 2015.

[17] "Workshop on buffer sizing." http://buffer-workshop.stanford.edu, accessed Augest 15, 2020.

[18] H. R. Varian, *Intermediate microeconomics with calculus: a modern approach*. WW Norton & Company, 2014.

[19] Q. Zhang, V. Liu, H. Zeng, and A. Krishnamurthy, "High-resolution measurement of data center microbursts," in *Proceedings of the 2017 Internet Measurement Conference*, 2017, pp. 78–85.

[20] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," *ACM SIGCOMM*, 2011.

[21] B. Vamanan, J. Hasan, and T. Vijaykumar, "Deadline-aware datacenter tcp (d2tcp)," *ACM SIGCOMM*, 2012.

[22] Y. Li, R. Miao, H. H. Liu, Y. Zhuang, F. Feng, L. Tang, Z. Cao, M. Zhang, F. Kelly, M. Alizadeh *et al.*, "Hpcc: high precision congestion control," in *ACM SIGCOMM*, 2019.

[23] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "Bbr: Congestion-based congestion control," *Queue*, vol. 14, no. 5, pp. 20–53, 2016.

[24] S. Ha, I. Rhee, and L. Xu, "Cubic: a new tcp-friendly high-speed tcp variant," *ACM SIGOPS*, 2008.

[25] A. Mishra, X. Sun, A. Jain, S. Pande, R. Joshi, and B. Leong, "The great internet tcp congestion control census," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 3, no. 3, pp. 1–24, 2019.

[26] G. F. Riley and T. R. Henderson, "The ns-3 network simulator," in *Modeling and tools for network simulation*. Springer, 2010.

[27] "Packet buffers." https://people.ucsc.edu/ warner/buffer.html, accessed June 29, 2020.

[28] R. Jain, A. Durresi, and G. Babic, "Throughput fairness index: An explanation," in *ATM Forum contribution*, 1999.

[29] D. C. Reeve, "A New Blueprint for Network QoS," Ph.D. dissertation, Computing Laboratory, University of Kent, Canterbury, Kent, UK, August 2003. [Online]. Available: http://www.cs.kent.ac.uk/pubs/2003/1892

[30] D. Intel, "Data plane development kit," 2014.

[31] B. Levasseur, M. Claypool, and R. Kinicki, "A tcp cubic implementation in ns-3," in *Proceedings of the 2014 Workshop on ns-3*, 2014.

[32] "empirical-traffice-gen." https://github.com/datacenter/empirical-traffic-gen.git, accessed June 29, 2020.

[33] N. Dukkipati and N. McKeown, "Why flow-completion time is the right metric for congestion control," *ACM SIGCOMM*, 2006.