AQ-CAR: Adaptive-Queueing Congestion-Aware Routing for Datacenter Traffic Forwarding

Sultan Alanazi and Bechir Hamdaoui
School of EECS, Oregon State University, Corvallis, Oregon, USA
{alanazsu,hamdaoui}@oregonstate.edu

***Abstract*—Modern datacenters are constructed with multi-rooted tree topologies and support multiple service queues per switch port. They support a wide variety of applications and services with stringent performance needs and conflicting requirements. To meet these requirements, recent works focus on load balancing or ECN marking approaches. Though existing load-balancing approaches can deliver good performance for both short and long flows, they do not consider integrating active queue management with load balancing to further improve the network performance and meet applications' requirements. In this paper, we propose AQ-CAR, a novel framework that combines active queue management with an efficient load-balancing algorithm to deliver high throughput and low latency simultaneously. More specifically, it classifies the queues in each switch port into small, medium, and large classes, each serving a specific flow type. Then it performs adaptive queueing where a flow initially is enqueued at the small service queue and then gets migrated adaptively to the other queues based on the number of bytes it has sent. To achieve congestion- and traffic-aware routing/rerouting, the load balancer first detects the flow type and then piggybacks the congestion information of the queues that serve this flow type. Then it routes/reroutes the flow to the path with minimum congestion. Large-scale ns-2 simulations show that AQ-CAR outperforms the state-of-the-art load balancing and ECN marking schemes for different performance metrics under a variety of workloads.**

***Index Terms*—Datacenter networks, congestion control, explicit congestion notification, queue management, fairness.**

I. Introduction

Cloud datacenters host a variety of applications and services, which generate a mix of delay-sensitive short flows and throughput-oriented long flows. For instance, file transfer [1] requires high throughput, whereas group video calls [2] require low latency. Moreover, some services such as online data-intensive applications [3] require both high throughput and low latency. Thus, it is important to have a resource allocation and management framework that can deliver both low latency and high throughput simultaneously. To meet these requirements, some researchers focus on improving load balancing while others aimed at providing active queue management of the network switches.

Modern datacenter networks are typically organized in multi-rooted tree topologies such as leaf-spine to provide multiple paths for communications between any host pairs [4]. Thus, many load balancing approaches have been proposed to efficiently balance the traffic across multiple available paths. As a flow-level load balancer, Equal Cost Multi-Path (ECMP) randomly maps each flow to one of the paths by using a hash taken over the packet headers. Even though ECMP is very simple, it suffers from the well-known hash collision problem and the lack of flexible traffic rerouting. To mitigate this issue, several load balancing schemes have been proposed with finer traffic routing granularity such as per-packet and per-flowlet. While packet-based load balancing schemes [5, 6] can rapidly maximize the link utilization and balance the traffic more efficiently, they suffer from serious packet reordering that can severely degrade the network performance.

Flowlet-based load-balancing schemes [7, 8], on the other hand, split each flow into small flowlets, which are a group of packets belonging to the same flow that are separated by a large enough time in order to avoid reordering. Flowlet-based approaches can make a good balance between packet reordering and link utilization. Nonetheless, they perform congestion-oblivious rerouting, which may increase the tail latency of short flows. On the other hand, a great body of work leverages ECN (explicit congestion notification) marking [9] to deliver both low latency and high throughput simultaneously. ECN-based transport mechanisms such as DCTCP [10] and DCQCN [11] have been broadly used in industry because of their simplicity. Although DCTCP and other transport schemes can deliver low latency and high throughput, their ECN marking schemes are designed for switches with only one single queue per switch port. However, current industry trends towards manufacturing switches with up to 8 classes of service queues per port [12].

MQ-ECN [13] has been proposed to optimize the ECN marking scheme of DCTCP in multi-queue scenarios. MQ-ECN periodically adjusts the ECN marking threshold for each queue independently based on the queue weight and the round-trip time. However, imprecise measurement is considerable in MQ-ECN since data traffic in DCN is bursty in nature. Moreover, setting the threshold dynamically requires periodic round-trip time measurements which incur non-negligible overhead for switches [13]. A-ECN [14] designs an adaptive ECN marking scheme that does not consider time interval, as in MQ-ECN, to update the marking threshold. Instead, it uses the number of enqueued packets as the interval to update the marking threshold. Although the ECN marking schemes mentioned above can achieve low latency and high throughput, they cannot ensure fairness among flows during the enqueueing and ECN marking process.

Therefore, in this paper, we propose AQ-CAR, a framework that combines active queue management and congestion-aware

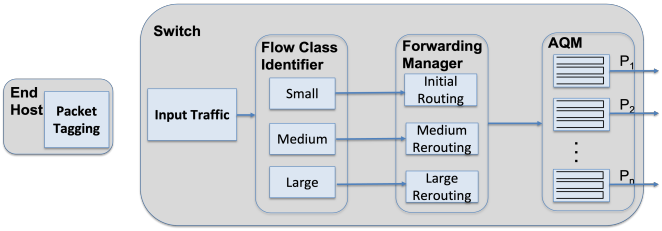


Fig. 1: AQ-CAR Overview

load balancing to deliver low latency for short flows while maintaining high throughput for long flows. To this end, the main contributions of the paper are:

1) Integrating active queue management with load balancing capability to develop efficient resource allocation and management framework for cloud datacenter networks.
2) Proposing a congestion-aware routing algorithm that makes initial flows to paths allocation during TCPâĂŹs 3-way handshaking process.
3) Proposing congestion- and traffic-aware rerouting algorithm that reroutes medium flows that contribute the most to link congestion without causing packet reordering.
4) Large-scale NS2 simulations of AQ-CAR under different realistic workloads. Our results show that AQ-CAR greatly reduces the average flow completion time over the state-of-the-art load balancing mechanisms.

The paper is organized as follows. Section II presents the proposed framework, AQ-CAR. Section III presents the performance results. Finally, Section IV concludes the paper.

II. AQ-CAR Design

We now present AQ-CAR, our proposed adaptive-queueing and congestion-aware routing framework for datacenter switches with multiple queues per port. At its core, AQ-CAR has two main components: packet tagging and switch design.

A. Packet Tagging

Maintaining per-flow states at switches requires counting and storing how much data has been sent for each flow, which is not supported in existing commodity switches. Therefore, AQ-CAR relies on end hosts' TCP to count the number of sent bytes and include this packet tagging information in the packet header. Switches can then leverage the tagged information to perform efficient routing and management of traffic. However, as it is not feasible for packets to carry accurate flow size information, which requires large header bits, we set and rely on threshold levels to indicate the size of the flow. Once the number of bytes sent by a flow reaches a certain threshold, the corresponding threshold level is tagged, again by end host's TCP, into the packet header to convey the flow's current size. Thus, at initialization when a flow starts, its packets are initially tagged with the first level, and as more bytes are sent, the flow's packets are tagged with increasing levels.

B. Switch Design

As shown in Fig. 1, AQ-CAR switch design has three main components: a flow class identifier, a forwarding manager, and an active queue manager (AQM). First, the flow class identifier extracts the flow size information from packet headers and arranges packets into small, medium, and large classes. The forwarding manager handles the routing and rerouting of each flow class. AQM is responsible for enqueueing, ECN marking, and dequeuing packets based on their flow class.

1) Flow Class Identifier: Upon a packet arrival, the flow class identifier extracts the flow size information tagged, by end hosts, in the IP DSCP field in the packet header and compares it to predefined thresholds, T_{small} and T_{medium}, to determine its flow class (stage value). Thus, when the stage value is less than or equal to T_{small}, an arriving packet will be handed to the initial routing module to forward the packet. When the stage value is greater than T_{small} and less than or equal to T_{medium}, the packet will be delivered to the medium rerouting module to check the possibility of rerouting without incurring packet reordering. Otherwise, the packet will be handed to the large rerouting module.

2) Forwarding Manager: Congestion-aware load-balancing protocol designs require the following key elements:

- Congestion Measurement: Leaf switches need to acquire spine switches' queue length information. One way is to carry this information in packets' headers, but this would require a large number of bits in the packet header. Instead, we use multi-level thresholds to represent queue length levels, which are then used to infer congestion levels. Specifically, AQ-CAR designates 2 bits in the packet header for this information. Following [10], to achieve full link utilization while maintaining low latency, the maximum threshold level is set as: $K = C \times RTT \times \lambda$, where C is the link capacity, RTT is the average round-trip time, and λ is a parameter set by the congestion control algorithm. AQ-CAR considers four levels, based on the value of $K/4$, with $K = 64$. These levels serve to exchange congestion information between leaf switches.
- Congestion Feedback Table: Congestion-aware load-balancing protocols require source leaf switches to have real-time congestion information from all paths between the flowâĂŹs source and destination. In the switch port as shown in Fig. 1, we have three service queues and each queue serves one of the flow classes (small, medium, and large). To manage the forwarding of each flow class, source leaf switches need to know the congestion level of each queue class. On each spine, the congestion level of the queue class serving the data packet will be tagged in the packet header before forwarding it to the destination leaf switches. This information is stored at the destination leaf switch in a flow class table and is fed back to the source leaf (when needed) through packet piggybacking in the reverse direction.
- Path Allocation Table (PAT): Path allocation decisions are maintained in PAT in each leaf switch. Each PAT entry represents a flow-to-path allocation, and records flow ID, selected port number, destination leaf ID, arrival time of the last enqueued packet, a valid bit indicating whether a flow is active (1) or not (0), and an age

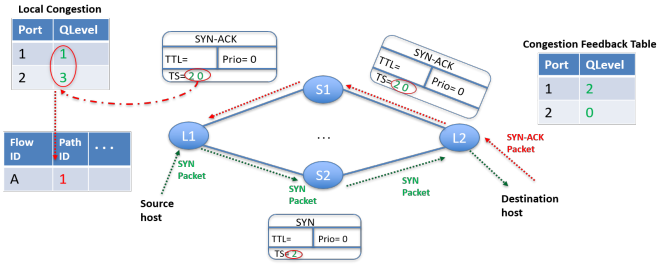


Fig. 2: ML-ECN Overview

bit indicating entry timeout. Upon the arrival of each packet, the switch looks for an entry based on its flow ID. If the entry is found and is valid, the packet is forwarded through the corresponding port indicated in the entry. If the entry is invalid or no entry exists, then the packet represents a new flow and starts a new path allocation process. PAT entries timeout after a period of inactivity indicating expired entries and triggering a new path allocation process. Only one bit is needed to implement the timer for each entry and a global timer for the entire PAT [5].

a) Initial Routing: Figure 2 illustrates our path allocation for a new flow. There is a new TCP connection between a source host under leaf switches L1 and a destination host under leaf switch L2. The flow is initiated using SYN in the forward direction from L1 to L2. When the first packet (SYN in this case) reaches the leaf switch L1, the flow class identifier detects that it is a new flow and hands the packet to the initial routing component for packet forwarding. The initial routing triggers the path selection mechanism after checking PAT. It forwards the SYN packet using ECMP and inserts a new entry into the PAT. Spine switches tag SYN and data packets with the congestion level of the queue that dequeued the packet. Upon the arrival of the SYN or a data packet, the destination leaf switch, L2 in this example, detects the flow class, pulls the congestion information, and updates the small congestion feedback table. However, if the packet belongs to a medium flow, then the medium congestion feedback table will be updated. The destination leaf then forwards the SYN packet to the destination host. This finishes the forward direction part of the path allocation process.

The reverse direction part starts when the destination host initiates SYN-ACK packet and forwards it to leaf switch L2. The leaf switch tags the packet with the ingress congestion information saved in the small feedback table for each port and routes the packet to the upper switch level using ECMP, which in turn forwards the packet to the leaf switch L1. L1 pulls the congestion information and aggregates it, entry-by-entry, with its egress link loads. The effective congestion of all n paths between the source leaf and the spine level is determined simply as the maximum load of the two hops. L1 then chooses the spine switch ID with the minimum effective congestion, which is 1 in this case. Finally, all the subsequent packets of this flow will be routed through port 1.

b) Medium Reouting: Since flow sizes are not known a priori, the initial routing algorithm can't achieve good load balancing. Initially, assigned flows in some ports may finish early while the flows in other ports grow in size and create bottlenecks in the network. Thus, it is essential to reroute parts of the traffic for better load balancing and for alleviating congestion and ensuring high network throughput. AQ-CAR adopts a rerouting approach with three key design goals:

- Alleviate congestion: Ensure that the newly selected path is not congested.
- Mitigate packet reordering: Ensure that earlier arrived packets are forwarded before latter arrived packets.
- Incur low overheads: Minimize the number of rerouted flows while achieving efficient load balancing.

To achieve these goals, AQ-CAR's rerouting algorithm leverages cooperation among network switches, with ECN marking is triggered on certain flows once the length of a medium queue class exceeds any of the predefined thresholds.

Destination leaf switches react to the marked packet by extracting the queue length information tagged in the packet header and checking if the following conditions exist. First, the queue length should exceed the rerouting threshold. Second, the flow must be among the larger flows in the medium class range. Medium queue class has a range of flow sizes, so as to minimize the number of rerouted flows and hence to reduce the rerouting overhead; i.e., AQ-CAR reroutes the flows that contribute the most to the queue buildup.

Once these conditions are met, the switch inserts the flow in the medium rerouting table. Then, it piggybacks every ACK for this flow with congestion information of all the ports recorded in the medium feedback table. The destination leaf stops packet piggybacking either if the flow is rerouted or the congestion is alleviated (the queue length falls below a predefined threshold) to reduce the network overhead.

Upon the arrival of the tagged ACK packet, the source leaf switch extracts the piggybacked congestion information and stores it in a piggy table indexed by the destination leaf switch ID. When a packet p_i, belongs to a medium flow f that needs to be rerouted, arrives at the source leaf switch at time t_i, the leaf calculates and records the leaf queueing delay as

$$LQD_i = \frac{P_{size} \times (mq_{length} + 1)}{C \times mq_{weight}/Total_{weight}} \quad (1)$$

where P_{size} is the packet size, mq_{length} is the length of the medium queue class in p_i's destined output port, C is the link bandwidth corresponding to the output port, mq_{weight} is the weigh of the medium queue class, and $Total_{weight}$ is the weight sum of all the queues. When the following packet p_{i+1} that belongs to a medium flow f arrives at time t_{i+1}, then the leaf switch reads the medium queue length information of the current spine queue, stored in a piggy table, and calculates the spine queueing delay as

$$SQD_i = \frac{P_{size} \times (mq_{length})}{C \times mq_{weight}/Total_{weight}} \quad (2)$$

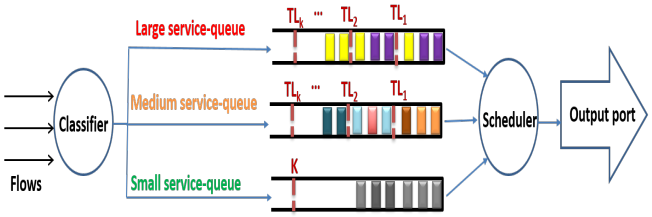


Fig. 3: AQM with ML-ECN Overview

The remaining queueing delay of packet p_i is then

$$RQD_i = SQD_i + LQD_i + (t_i - t_{i+1}) \quad (3)$$

The forwarding manager of the switch then finds the paths with a larger queueing delay than RQD_i and selects the path with the current minimum queueing delay among all possible paths. Packet p_{i+1} and all the subsequent packets of flow f will be routed through the newly selected path.

c) Large ReRouting: 10% of TCP flows generate about 90% of overall data traffic, while 90% only generate around 10% of the traffic [10, 15]. This means that a significant portion of the data traffic comes from large flows. Thus, the aim of the large rerouting component is to distribute these large flows across the network and to prevent multiple large flows from sharing the same port. Once a destination leaf switch detects that two large flows share the same port and there exists another port with no large flows, it piggybacks one of the large flows' ACK packets with the ports that do not have large flows, thus signaling the source leaf to initiate large flow rerouting. Upon receiving these packets, the source leaf reroutes the flow to a new path that has no large flows.

3) Active Queue Manager:

a) ECN Marking: As depicted in Fig. 3, AQ-CAR adopts multi-level ECN (ML-ECN) marking in each switch queue. ML-ECN is designed with the three-fold goal of ensuring:

- Low latency for small (latency-sensitive) flows.
- High throughput for large (throughput-sensitive) flows.
- Fair packet enqueueing and ECN marking so as to not penalize smaller flows due to high buffer pressure that could result from larger flows.

ECN-marking based on a single threshold value cannot always guarantee high throughput, low latency, and fairness in the presence of flows with various sizes [13, 16]. To overcome this limitation, we leverage the classification of queues into different classes and design a specific ECN-marking scheme for each queue class as follows:

- Small service-queue: We employ per-queue standard threshold K for this queue class.
- Medium service-queue: We design a scheme that marks packets in the queue based on the current queue length and the total length of all queues in the port buffer. First, the scheme compares the total length of all queues to the standard threshold K. If the total length is less than K, then it applies ECN-marking, based on multi-level thresholds, as follows: $TL_i = i \times K/L$ with $i \in [1, L]$ where TL_i, K, and L are respectively the i-th threshold level, standard thresholds and the total number of thresholds. Furthermore, for each threshold level employed on the service queues, we perform fairness-aware probabilistic ECN marking on a selective range of flows to ensure that smaller flows are not marked (penalized) at the early stages of queue build-up. Therefore, at the first (minimum) threshold, ML-ECN only performs ECN marking, with low probability, on large flows with maximum stage value tagged in their packet headers. As the threshold level increases, ECN marking is also performed, with low probability, on flows with lower stage values. Moreover, the marking probability will increase on the flows with larger stage values. Thus, packet marking in ML-ECN is proportional to the queue length. On the other hand, if the total length of all queues in the port buffer is greater than the standard threshold K, then the ECN-marking scheme marks all dequeued packets if the queue length exceeds its minimum threshold TL_1.
- Large service-queue: We employ similar ECN-marking as in the medium service-queue but with the following changes. Instead of applying selective marking, based on flow sizes, when queue length reaches each threshold level, the scheme in large service-queue marks all packets at each threshold level with increasing marking probability. Moreover, if the total length of all queues in the port buffer is greater than the standard threshold K, the scheme on each queue marks all packets regardless of the current queue length.

b) Scheduler: As the case of most commodity switches, AQ-CAR adopts Weighted Round Robin (WRR) scheduling to dequeue packets from the different queues, where the number of packets dequeued from each queue is proportional to the weight assigned to the queue. With packets enqueued into different queues according to their current stage level, the built-in WRR scheduler leverages the classification of queues into small, medium, and large service queues, and assigns weights to the queues with the goal of achieving low latency for small flows without impacting the throughput of large flows. Therefore, ML-ECN assigns higher weights to small and medium queues and lower weights to large queues.

III. Performance Evaluation

Large-scale ns-2 simulations are conducted to evaluate the performances of AQ-CAR, and compare them against representative schemes that focus on either load balancing or ECN marking to improve datacenter network performances.

- **Flex [7]:** A load balancing scheme based on load-adaptive flowlet timeout. It splits network traffic into flowlets at end hosts and forwards each packet at switches using both the five-tuple and a flowlet tag. It uses the default ECN-marking scheme of DCTCP [10] for AQM.
- **A-ECN [14]:** An adaptive ECN marking scheme designed to minimize the flow completion time (FCT) while maintaining high throughput. It uses the default equal-cost multipath (ECMP) for load balancing.

- **ECMP [17]:** The most commonly used routing in datacenters that provides equal-cost multipath. ECMP routes each packet by taking a hash of the packet's TCP 5-tuple (src IP, dest IP, src port, dest port, protocol). It uses the default ECN-marking scheme of DCTCP [10] for AQM.

Topology: We run the experiments on 144-host Leaf-Spine topology. The network in our experiments has 12 leaf (Top-of-Rack) switches, and 12 spine (Core) switches. Each Leaf switch is connected to 12 hosts through 10Gbps downlink ports and is connected to 12 Spine switches through 10Gbps uplink ports, forming a non-blocking network. The round-trip-time (RTT) latency across spines (4 hops) is $85.2\mu sec$.

Workloads: We run 4 different workloads, a Web search [10], a Data mining [18], a Cache [19], and a Hadoop [19] with 914 KB, 1671 KB, 4149 KB, and 7495 KB of mean flow sizes, respectively. In addition, we run a workload consisting of the four different flow distributions all mixed together (mix workload). We use FCT (flow completion time), throughput, and the number of dropped packets as the performance metrics in our simulation. We divide all flows based on their sizes into three classes, including short flows $(0, 100KB]$, medium flows $(100KB, 10MB]$, and large flows $(> 10MB)$. We consider the results of overall flows and the breakdown across different flow sizes independently (short, medium, and long). All simulations last for 50K flows.

A. Flow Completion Time (FCT)

FCT is the time from when the SYN packet of a flow is sent until the last packet is received by the destination. In this section, we show the FCT performance of all the schemes when running them across different workloads.

1) Mix Workload:

a) Short Flows: The majority of short flows in datacenter networks are generated by applications and services that demand low latency [4]. Figure 4b shows how the schemes perform in delivering low latency for short flows when running Mix Workload. Observe that AQ-CAR significantly reduces its average FCT by 54% and 49% compared to ECMP and A-ECN respectively. The reason for the inferior performance in A-ECN and ECMP is the long queueing delay that short flows experience before leaving the switches. Moreover, both schemes perform congestion-oblivious routing decisions at a very coarse granularity. They route multiple long-lived flows on the same path, leading to congestion and thus increased latency. Compared to Flex, AQ-CAR yields about 33% reduction in FCT at high loads. Flex achieves better performance compared to A-ECN and ECMP because it uses flowlet-level routing granularity which is fine-grained compared to the flow-level granularity used by ECMP and A-ECN. We attribute the improvement of AQ-CAR to the congestion-aware routing and to the enqueueing mechanism that isolates short flows from other flows into a separate queue.

b) Medium and Overall Flows: For medium flows $(100KB, 10MB]$, AQ-CAR also achieves superior performance in FCT compared to all other schemes as shown in Figure 4c. AQ-CAR yields about 25%, 17%, and 13% reduction in FCT, at high loads, when compared to A-EC, Flex, and DCTCP, respectively. This is mainly because AQ-CAR performs congestion-aware routing and rerouting load balancing. Even though Flex performs flowlet routing, it is oblivious to the congestion in the spine-level switches which may incur long queueing delays and cause packet reordering.

Figure 4a shows a similar trend for the average FCTs (averaged over all flows) as for the FCT of medium flows.

c) Large Flows: Figure 4d shows the average FCT of large flows $(> 10MB)$ when comparing AQ-CAR to other schemes. Observe that the performance gap between the schemes is not significant. This shows the efficiency of AQ-CAR in reducing the FCT for small and medium flows without jeopardizing the performance of large flows.

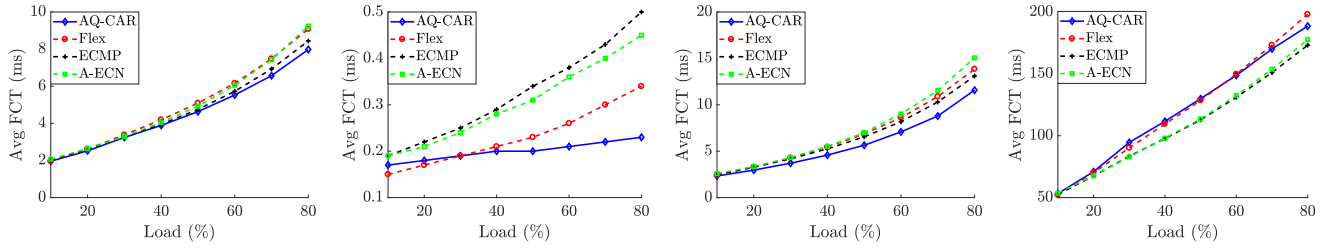
2) Web search and Cache Workloads:

a) Short Flows: Figures 5b and 6b show the average FCT of short flows under the Web search and Cache workloads, respectively. AQ-CAR improves FCT significantly compared with the other three schemes, especially at high loads. More specifically, in the Web search workload, AQ-CAR minimizes FCT by about 65%, 62%, and 56% at 80 percent load over ECMP, A-ECN, and Flex, respectively. Similarly, in the Cache workload, AQ-CAR outperforms the other schemes by around $47\%- 57\%$. These results demonstrate the advantage of AQ-CAR, with adaptive queueing and congestion-aware routing, over the other schemes in alleviating the impact of large queueing delays that severely degrade the performance of the delay-sensitive short flows. When the load becomes high, more flows with different sizes coexist in the same output ports at the switches. Thus, more short flows experience long-tail queueing delays. ECMP and A-ECN perform congestion oblivious routing, resulting in a hash collision and more queueing delay.

b) Medium and Overall Flows: Figure 5c shows the average FCT of medium flows under the Web search workload. Observe that AQ-CAR achieves up to 17% reduction in FCT compared to the other schemes. However, the performance of AQ-CAR decreases at the 0.8 load. Note that under Cache workload in Figure 6c, the performance of AQ-CAR also decreases at high loads similar to the Web search workload. The reason is that most traffic in both workloads are from medium flows and AQ-CAR puts them in a designated queue and triggers ECN marking based on the queue length. Thus, at high loads, more medium flows are concurrently active, leading to larger queue length and more frequent ECN marking. On the other hand, the medium flows in the other schemes are distributed among the queues rather than a single queue in our scheme, leading to less ECN marking as the ECN marking is triggered based on each queue independently.

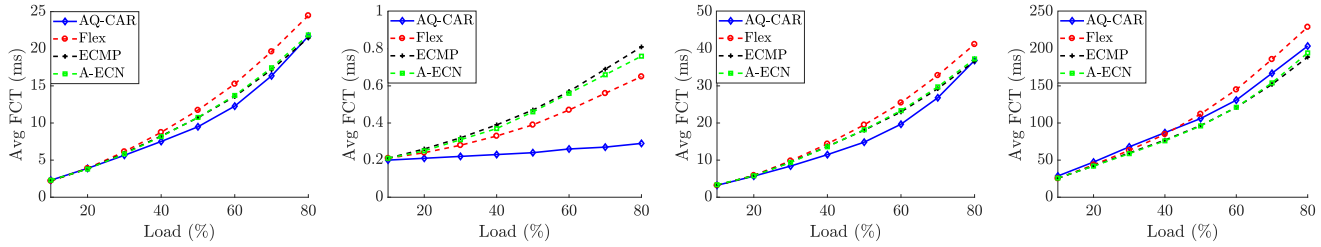
Figures 5a and 6a show the average (over all flows) FCT under both Web search and Cache workloads. Observe that they are similar to the medium flow figures as in Web search and Cache workloads, most traffic comes from medium flows.

c) Large Flows: Figures 5d and 6d show the average FCT of large flows when comparing AQ-CAR to the other schemes, under both Web search and Cache work-



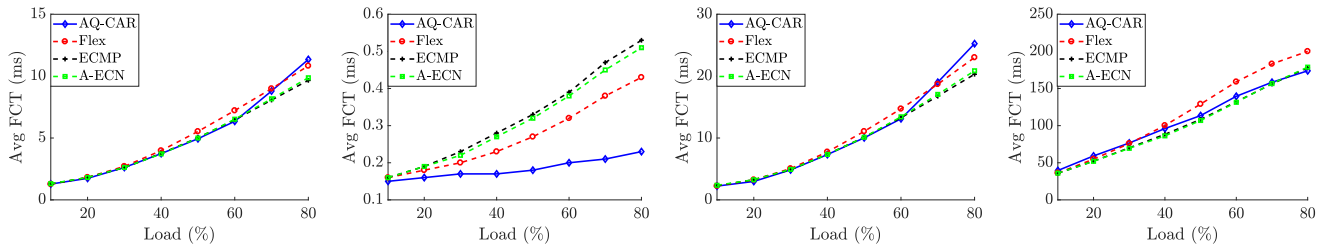
(a) Overall (b) Small (c) Medium (d) Large

Fig. 4: FCT statistics across different flow sizes for Mix workload



(a) Overall (b) Small (c) Medium (d) Large

Fig. 5: FCT statistics across different flow sizes for Web search workload



(a) Overall (b) Small (c) Medium (d) Large

Fig. 6: FCT statistics across different flow sizes for Cache workload

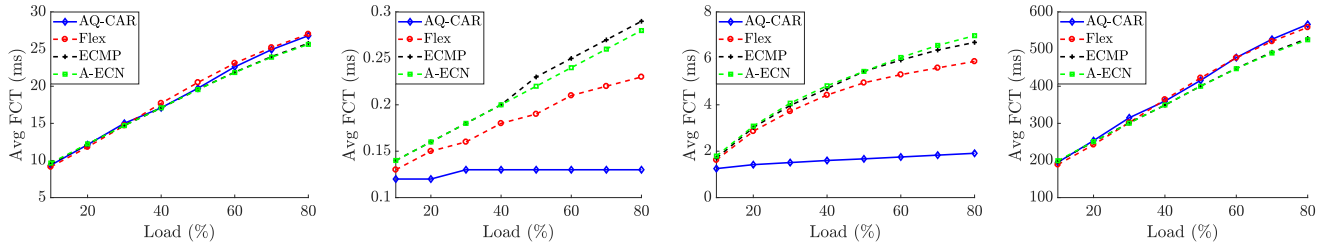
loads, respectively. Observe that all schemes achieve similar performances. This again shows the efficiency of AQ-CAR in significantly reducing the FCT for small flows without jeopardizing the performance of medium and large flows.

3) Data mining and Hadoop Workloads:

a) Short Flows: Figures 7b and 8b further confirm the superiority of AQ-CAR in reducing the latency for short flows under both Data mining and Hadoop workloads. The performance gap of AQ-CAR reaches about 56%, 54%, and 44% for the Data mining workload when compared to DCTCP, A-EC, and Flex, respectively. Under the Hadoop workload, AQ-CAR reduces the average FCT by about 56%, 53%, and 39% compared to DCTCP, A-ECN, and Flex, respectively.

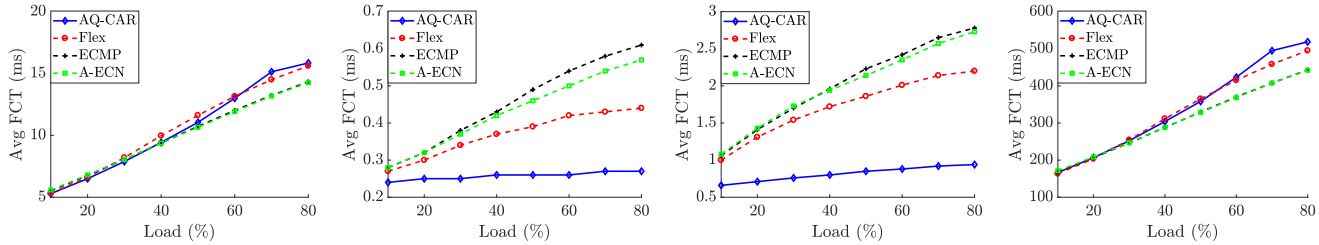
b) Medium Flows: FCT of medium flows running Data mining and Hadoop workloads present interesting results as shown in Figures 7c and 8c. Compared to Web search and Cache workloads, the performance gain of AQ-CAR over the other schemes becomes more significant in the Data mining and Hadoop workloads. This is because in these workloads, only a small portion of the total bytes is generated by medium flows. For example, less than 5% of bytes in Data mining are from medium flows while 95% of bytes are from large flows [20]. Therefore, AQ-CAR eliminates the negative impact of large flows on the other flows in two ways. First, it distributes large flows across the network to avoid hash collisions that cause long queueing delays and frequent packet drops. Second, it isolates large flows in separate queues and triggers ECN marking on each queue independently. This results in a remarkable improvement in the average performance of medium flows under both Data mining and Hadoop workloads.

c) Large and Overall Flows: For large flows, Figures 7d and 8d show that AQ-CAR achieves performances comparable to the other schemes. This confirms the efficiency of AQ-CAR in reducing the latency of short and medium flows while not impacting the throughput achieved by long flows. Similarly in Figures 7a and 8a, all the schemes provide comparable



(a) Overall (b) Small (c) Medium (d) Large

Fig. 7: FCT statistics across different flow sizes for Data mining workload



(a) Overall (b) Small (c) Medium (d) Large

Fig. 8: FCT statistics across different flow sizes for Hadoop workload

performance in the overall average FCT when running both Data mining and Hadoop workloads.

B. Throughput

1) Mix Workload: Figure 9 shows the average throughput of the studied schemes achieved under the Mix workload. Observe that AQ-CAR has noticeably improved the throughput of short and medium flows without significantly degrading the throughput of large flows, thus achieving better overall throughput performance. For short flows, Figure 9b shows that AQ-CAR, at high loads, yields about 17%, 35% and 39% performance gain in throughput when compared to Flex, A-ECN, and ECMP, respectively. Furthermore, AQ-CAR improves the average throughput for medium and overall flows by more than 22% compared to other schemes as shown in Figures 9c and 9a. Even though AQ-CAR shows no gain in throughput for large flows as shown in Figure 9d, it manages to deliver comparable performance against the other schemes.

2) Web search and Cache Workloads: Figures 10b and 11b show the average throughput for short flows under both Web search and Cache workloads. Observe that AQ-CAR achieves more than 42%, 48%, and 50% gain in throughput in comparison to Flex, A-ECN, and ECMP, respectively.

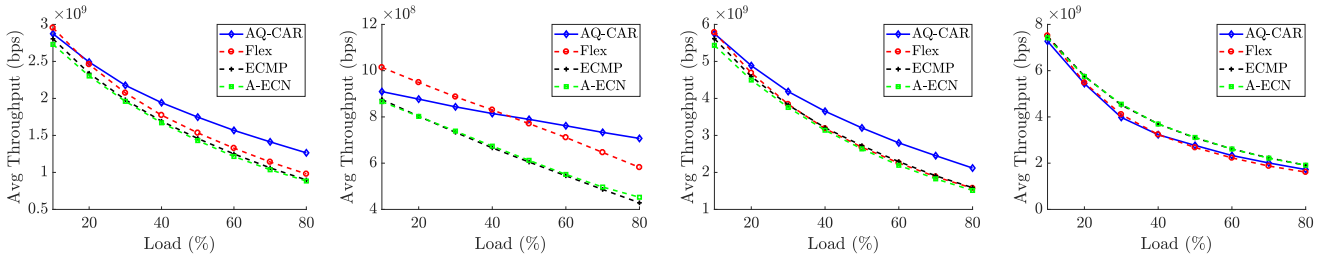
For medium and overall flow sizes under the Web search workload, the performance gain of AQ-CAR exceeds 42% compared to the other schemes as shown in Figures 10c and 10a. Running the Cache workload, the throughput gain of AQ-CAR surpasses 30% in contrast to the other schemes as demonstrated by Figures 11c and 11a.

Figures 10d and 11d show that AQ-CAR degrades the throughput performance of large flows by less than 7% contrasted with the other approaches. This confirms the efficiency of the proposed work in reducing latency for short flows without significantly impacting the throughput of long flows.

3) Data mining and Hadoop Workloads: Figures 12 and 13 show the throughput statistics across different flow sizes for Data mining and Hadoop Workloads. Observe that AQ-CAR outperforms the other schemes in throughput across all flow sizes except for long flows, where it shows no throughput gain. This shows the efficiency of the proposed framework in delivering high throughput across all flow sizes.

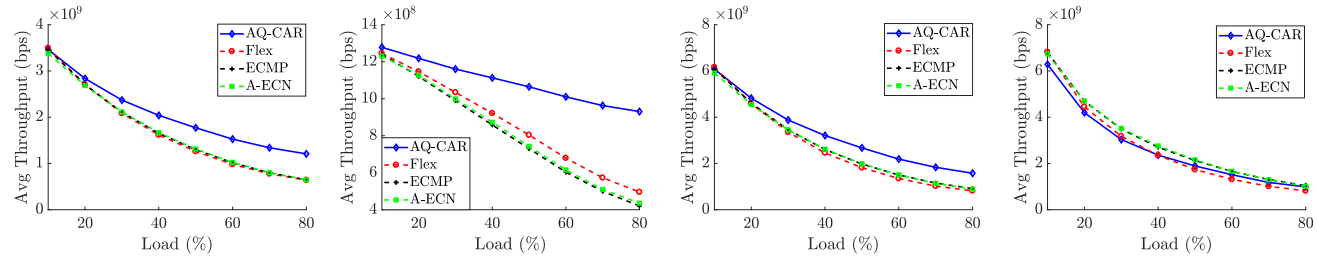
C. Packet Drop Loss

Figure 14 compares the total number of dropped packets, normalized to the maximum dropped value across all loads, running all the schemes across different workloads. Observe that AQ-CAR significantly reduces the number of dropped packets compared to the other frameworks across all workloads. Three factors contribute to AQ-CAR's remarkable reduction in the number of dropped packets. First, AQ-CAR performs congestion-aware routing for new flows. Then, it triggers adaptive rerouting on larger flows at bottleneck links to alleviate congestion that causes frequent packet drops. Finally, it performs adaptive queueing and ECN marking on each queue independently. This allows AQ-CAR to trigger more ECN marking on the flows that contribute the most to the queue buildup, which results in a gradual increase in the port buffer, and hence in a lower number of dropped packets.



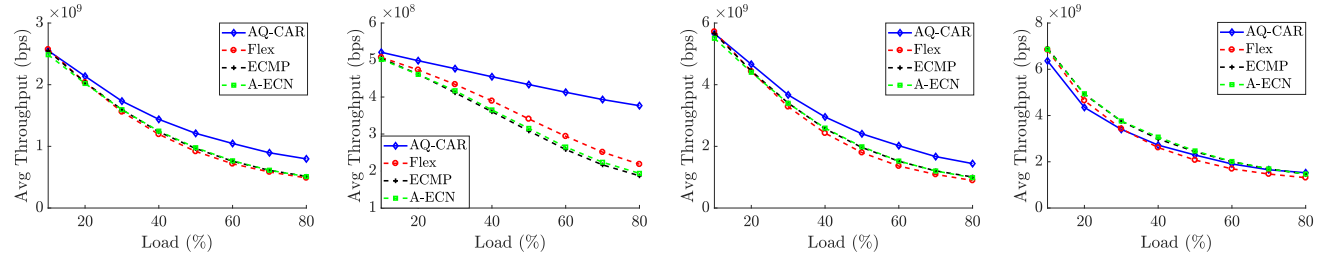
(a) Overall (b) Small (c) Medium (d) Large

Fig. 9: Throughput statistics across different flow sizes for Mix workload



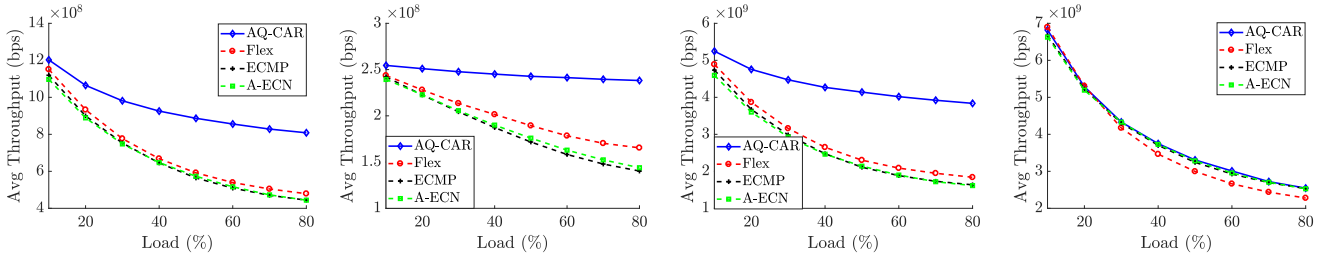
(a) Overall (b) Small (c) Medium (d) Large

Fig. 10: Throughput statistics across different flow sizes for Web search workload



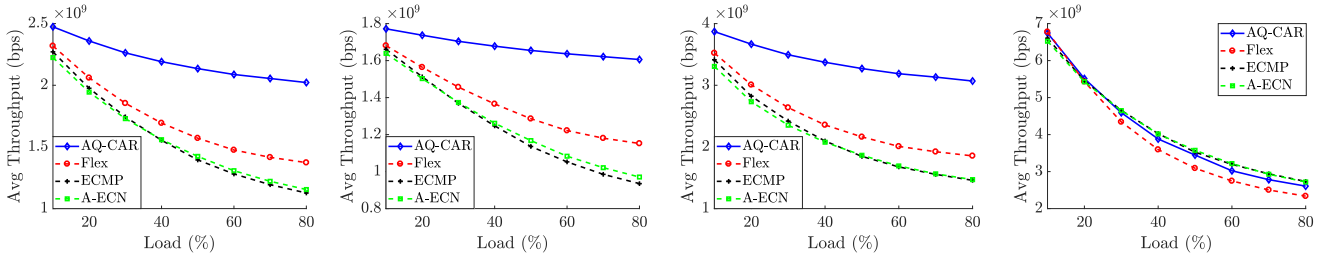
(a) Overall (b) Small (c) Medium (d) Large

Fig. 11: Throughput statistics across different flow sizes for Cache workload



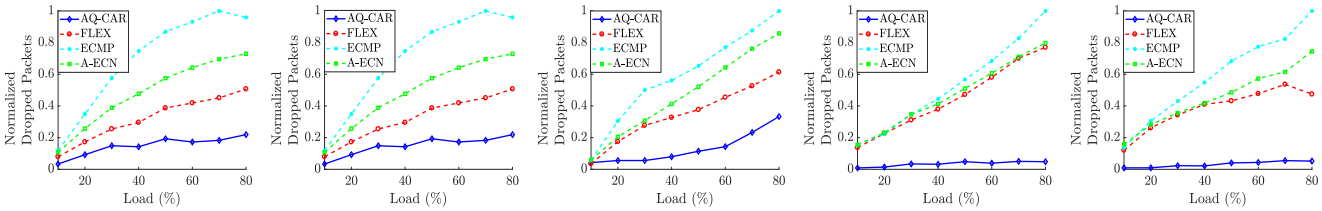
(a) Overall (b) Small (c) Medium (d) Large

Fig. 12: Throughput statistics across different flow sizes for Data mining workload



(a) Overall (b) Small (c) Medium (d) Large

Fig. 13: Throughput statistics across different flow sizes for Hadoop workload



(a) Mix (b) Web search (c) Cache (d) Data mining (e) Hadoop

Fig. 14: Dropped packets comparison across different workloads

IV. Conclusion

We propose AQ-CAR for multi-rooted multi-queue DCNs to attain both high throughput and low latency. AQ-CAR separates small, medium, and large flows into their separate queues with a different number of ECN thresholds. Afterward, it performs congestion-aware routing and rerouting based on flow types. Simulation results show the superiority of AQ-CAR in reducing FCT when compared to existing approaches.

References

[1] Andrew R Curtis, Wonho Kim, and Praveen Yalagandula, "Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection," in *Proc. of IEEE INFOCOM*, 2011, pp. 1629–1637.

[2] Ting Wang, Lu Wang, and Mounir Hamdi, "A cost-effective low-latency overlaid torus-based data center network architecture," *Computer Communications*, vol. 129, pp. 89–100, 2018.

[3] Keon Jang, Justine Sherry, Hitesh Ballani, and Toby Moncaster, "Silo: Predictable message latency in the cloud," in *Proc. of ACM Conference on SIG on Data Communication*, 2015, pp. 435–448.

[4] Tao Zhang, Qianqiang Zhang, Yasi Lei, Shaojun Zou, Juan Huang, and Fangmin Li, "Load balancing with traffic isolation in data center networks," *Future Gen. Computer Systems*, vol. 127, pp. 126–141, 2022.

[5] Advait Dixit, Pawan Prakash, Y Charlie Hu, and Ramana Rao Kompella, "On the impact of packet spraying in data center networks," in *2013 Proceedings IEEE INFOCOM*. IEEE, 2013, pp. 2130–2138.

[6] Soudeh Ghorbani, Zibin Yang, P Brighten Godfrey, Yashar Ganjali, and Amin Firoozshahian, "Drill: Micro load balancing for low-latency data center networks," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 2017, pp. 225–238.

[7] Xinglong Diao, Huaxi Gu, Xiaoshan Yu, Liang Qin, and Changyun Luo, "Flex: A flowlet-level load balancing based on load-adaptive timeout in dcn," *Future Generation Computer Systems*, 2022.

[8] Erico Vanini, Rong Pan, Mohammad Alizadeh, Parvin Taheri, and Tom Edsall, "Let it flow: Resilient asymmetric load balancing with flowlet switching," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, 2017, pp. 407–420.

[9] Kadangode Ramakrishnan, Sally Floyd, David Black, et al., "The addition of explicit congestion notification (ecn) to ip," 2001.

[10] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan, "Data center tcp (dctcp)," in *Proceedings of the ACM SIGCOMM 2010 Conference*, 2010, pp. 63–74.

[11] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang, "Congestion control for large-scale rdma deployments," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 523–536, 2015.

[12] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker, "pfabric: Minimal near-optimal datacenter transport," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 435–446, 2013.

[13] Wei Bai, Li Chen, Kai Chen, and Haitao Wu, "Enabling {ECN} in multi-service multi-queue data centers," in *13th {USENIX} Symposium on Networked Systems Design and Implementation*, 2016, pp. 537–549.

[14] Shuo Wang, Jiao Zhang, Tao Huang, Tian Pan, Jiang Liu, and Yunjie Liu, "A-ecn minimizing queue length for datacenter networks," *IEEE Access*, vol. 8, pp. 49100–49111, 2020.

[15] Theophilus Benson, Aditya Akella, and David A Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. of ACM SIGCOMM Conf. on Internet Measurement*, 2010, pp. 267–280.

[16] Sultan Alanazi and Bechir Hamdaoui, "Ml-ecn: Multi-level ecn marking for fair datacenter traffic forwarding," in *ICC 2022-IEEE International Conference on Communications*. IEEE, 2022, pp. 2726–2731.

[17] C Hopps, "Rfc2992: analysis of an equal-cost multi-path algorithm," 2000.

[18] Albert Greenberg, James R Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta, "Vl2: A scalable and flexible data center network," in *Proc. of the ACM SIGCOMM Conference on Data communication*, 2009, pp. 51–62.

[19] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C Snoeren, "Inside the social network's (datacenter) network," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, 2015, pp. 123–137.

[20] Jinbin Hu, Jiawei Huang, Wenjun Lyu, Weihe Li, Zhaoyi Li, Wenchao Jiang, Jianxin Wang, and Tian He, "Adjusting switching granularity of load balancing for heterogeneous datacenter traffic," *IEEE/ACM Transactions on Networking*, vol. 29, no. 5, pp. 2367–2384, 2021.