Journal of Network and Systems Management (2021) 29:26
https://doi.org/10.1007/s10922-021-09590-z

AAC: Adaptively Adjusting Concurrency by Exploiting Path Diversity in Datacenter Networks

Weimin Gao[1,2] · **Jiawei Huang**[1] · **Shaojun Zou**[1] · **Weihe Li**[1] · **Jianxin Wang**[1] · **Jianer Chen**[3]

Received: 6 September 2020 / Revised: 21 December 2020 / Accepted: 6 February 2021 /
Published online: 5 March 2021

Abstract
Recent datacenter load balancing designs make full use of all available multiple paths to achieve high bisection bandwidth and support the increasing traffic demands. However, a multitude of uncertainties, such as congestion and asymmetry, easily leads to long tailed latencies for unlucky flows on bad paths. In this paper, we aim at adjusting the maximum number of multiple paths used by existing load balancing designs to achieve good tradeoff between the tailed latency and link utilization. Specifically, we propose a packet-level load balancing called scheme Adaptively Adjusting Concurrency (AAC) to spread packets across the multiple paths, which are adaptively selected according to path diversity. AAC is deployed at switch, without any modifications on end-hosts. The experimental results of NS2 simulation and Mininet implementation show that AAC significantly reduces the flow completion time by ~21–56% over the state-of-the-art datacenter load balancing designs including MPTCP, LetFlow and RPS.

Keywords Datacenters · Path diversity · Flow completion times · Concurrency

1 Introduction

Modern data centers with thousands of servers host a variety of large-scale data processing applications and services such as web search, cloud storage, and big-data analytics. A rich body of datacenter load balancing schemes has emerged to improve transmission performance for the increasing traffic over the multiple paths in multi-rooted tree topologies such as Fat-tree [1] and Clos [2].

Equal Cost MultiPath (ECMP) [3] is the standard load balancing scheme used in data center network. However, as randomly assigning flows across available paths

✉ Jiawei Huang
jiaweihuang@csu.edu.cn

Extended author information available on the last page of the article

using flow hashing, ECMP performs poorly due to hash collisions and low link utilization. To resolve the problem, many congestion-aware load balancing schemes are proposed. For example, MPTCP [4] uses parallel subflows to minimize packet reordering and obtain high link utilization. Random Packet Spraying (RPS) [5], DRILL [6] and Hermes [7] flexibly split flows at packet-level to make full use of all available paths.

Unfortunately, these state-of-the-art load balancing schemes do not consider the important feature that a multitude of uncertainties widely exist in production data centers. The traffic dynamic, topology asymmetry and switch failure arise as data center operate over time [7, 8]. Under such uncertainties, the multiple paths become diverse or asymmetric. When load balancing schemes scatter packets on the bad or congested paths, the unlucky flows unavoidably experience unpredicted congestion and packet disordering, resulting in long tailed latency and suboptimal network goodput.

In this paper, we explore a self-adjusting approach AAC that selects the multiple paths used by existing load balancing designs to achieve both low tailed latency and high link utilization. To mitigate the impact of uncertainties under high path asymmetry, AAC shrinks the number of multiple paths to avoid high tailed latency and packet reordering. On the contrary, when the path asymmetry is low, AAC uses more paths to spread packets to achieve high utilization and network goodput. Moreover, AAC only needs to be deployed on switch, while making no modifications on existing TCP/IP protocols at end hosts.

In summary, our major contributions are:

- We conduct an extensive simulation-based study to analyze the impact of path asymmetry on the load balancing performance. We demonstrate experimentally and theoretically why controlling number of paths is effective in avoiding large tailed flow completion time (FCT) and packet reordering under large path asymmetry.
- We propose a packet-level load balancing scheme AAC to spread packets across the multiple paths, which are adaptively selected according to path diversity. AAC rationally adjusts the number of paths to improve link utilization under small path asymmetry and reduce tailed latency under large path asymmetry.
- By using both Mininet testbed and large-scale NS2 simulations, we demonstrate that AAC performs remarkably better than the state-of-the-art load balancing designs under different realistic traffic workloads. Especially, AAC greatly reduces the tailed FCT by ~21–56% over MPTCP, LetFlow and RPS.

The remainder of this paper is structured as follows. In Sect. 2, we describe our design motivation. The design detail of AAC is presented in Sect. 3. In Sects. 4 and 5, we show the experimental results of NS2 simulation and Mininet implementation, respectively. In Sect. 6, we demonstrate existing approaches. Finally, we conclude the paper in Sect. 7.

2 Background and Motivation

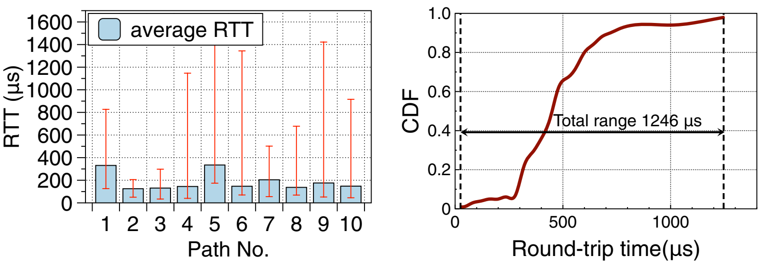
In this section, we present empirical studies to show the path asymmetry is very common in the modern data centers. Then, we analyze the impact of path asymmetry on load balancing performance and demonstrate that controlling number of paths is effective in reducing latency under large path asymmetry.

2.1 Path Asymmetry in Production Data Center

Modern data center networks use multi-rooted tree topologies such as Fat-tree and Clos to enable multiple paths between host pairs. However, the multiple paths become asymmetric under network uncertainties, such as traffic dynamic and switch failure [9–12]. In the following, we measure the round trip time of multiple paths in a production data center to show the wide existence of path asymmetry.

To investigate the path asymmetry, we analyze the network trace of an university data center in a leaf-spine architecture with 26 switches and 120 servers. There are 10 spine switches(Huawei Quidway S9306) and 16 leaf switches(Ruijie RGS-2928G), respectively. The uplink and downlink bandwidth of switches are 1Gbps. The university data center provides a variety of services including web service, distributed database system and E-mail service.In the test, one host sends 1000 ping packets to other 10 application servers under different leaf switches. Each application server responds with 100 pong messages. Then the host measures the average RTT of each path to application server. RTT distribution is shown in Fig. 1a. The path RTT varies from 25 to 1271 μs.

In the production network, the RTT distribution is effected by the locations of source and destination hosts [13, 14]. The RTT between servers in the same rack is usually low. However, the network traffic changes greatly between racks, easily forming a long-tailed distribution. Figure 1b shows the RTT's CDF. Though the median RTT is 408 μs, there are also about 10% cases where the RTT value becomes larger than 1 *m*s.



(a) RTT distribution **(b)** RTT CDF

Fig. 1 RTT measurement results in production data center

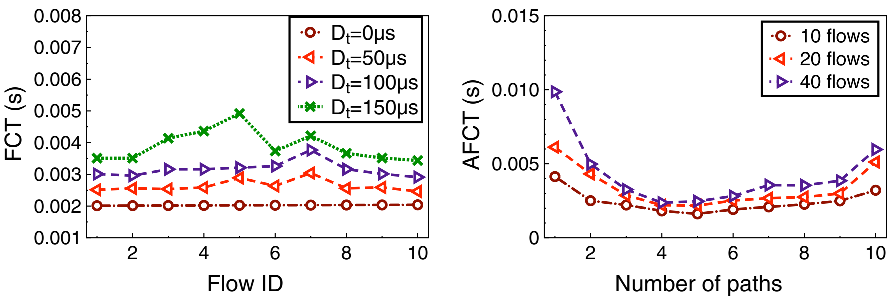
2.2 Impact of Path Asymmetry on Load Balancing Performances

In order to explore the impact of path asymmetry on load balancing performances, we use the NS2 simulation to test the performance of RPS, which is the typical datacenter load balancing design already implemented on the commodity switches. The test topology is a leaf-spine topology [15] with 10 spine switches and 2 leaf switches. The bandwidth of each path and buffer size of each switch are 1Gbps and 250 packets, respectively. Each sender sends a DCTCP flow to a single receiver via leaf switches with RPS scheme, which randomly spreads the arriving packets to all 10 paths. To produce the path asymmetry, we change the round trip propagation delay of each path.

Firstly, we measure the flow completion time of 10 short flows respectively transferring 20 packets (1500 Byte per packet). We set the round trip propagation delay of one path as 100 μs and increase RTT by RTT difference D_t. The degree of path asymmetry increases with larger RTT difference. Figure 2a shows the flow completion time of each flow under different degree of path asymmetry. When the RTT difference is 0, all flows experience almost the same flow completion time. With larger degrees of path asymmetry, however, more packets are blocked on the slow paths, resulting in larger flow completion time.

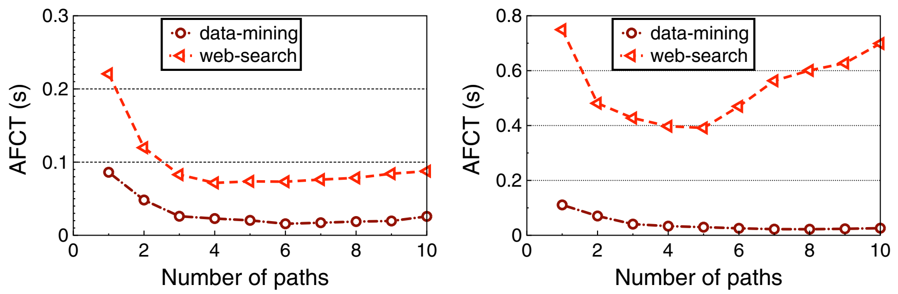
Next, we set the RTT difference to 100 μs and change the number of paths used by RPS to test the average flow completion time (AFCT). Here, the used paths are randomly selected from 10 paths with different round trip propagation delay. The experiments are repeated for 20 times to measure AFCT. Figure 2b shows the test results with different number of flows. When the number of paths is small (i.e.,$<$ 5), AFCT decreases with more paths because of the larger link utilization. However, when the number of paths is larger than 5, AFCT becomes larger with more paths, because the probability of packets scattered on slow paths also increases, eliminating the benefit of larger link utilization.

Finally, we test the impact of path asymmetry in the realistic workloads of web search and data mining. In the web search workload, about 30% flows are larger than 1MB, while in data mining less than 5% flows are larger than 35MB [16, 17]. Figure 3 shows the average and 99th percentile flow completion time with increasing



(a) Change the difference of paths **(b)** Change the number of flows

Fig. 2 FCT under different path asymmetry



(a) AFCT of real workload **(b)** 99th percentile AFCT

Fig. 3 FCT under realistic workloads

number of paths. The RTT difference is 100 μs. Figure 3a shows that AFCT of both workloads firstly declines and then arises with the increasing number of paths. In Fig. 3b, the 99th percentile FCT shows the similar trend. This result shows that, more paths reduce the tailed delay under small path asymmetry. However, under large path asymmetry, more packets experience large tailed delay with more used paths, resulting in large 99th percentile FCT.

2.3 Summary

Based on the above analysis, we draw the following conclusions that (i) more paths provide higher link utilization to reduce flow completion time under small path asymmetry, (ii) more paths easily increase the tailed delay under large path asymmetry. These conclusions motivate us to design a novel load balancing scheme that adjusts the maximum number of multiple paths to achieve good tradeoff between the tailed latency and link utilization. In the rest of this paper, we present our AAC scheme as well as a prototype implementation in real testbed system.

3 Adaptively Adjusting Concurrency

In this section, we will firstly describe the design overview of AAC. Then, we present the details on how to measure the path delay at switch. Moreover, we theoretically analyze how to obtain the optimal number of concurrent paths and give the details of adaptive adjusting the number of paths according to network congestion state. Finally we evaluate the accuracy of model analysis by comparing the results of theoretical analyze and simulation test.

3.1 Design Overview

In Fig. 4, we plot the architecture of AAC, which includes the congestion detection module and path adjustment module. Firstly, AAC measures the delay between the source and destination leaf switch (i.e., leaf-to-leaf delay) to reflect the real-time

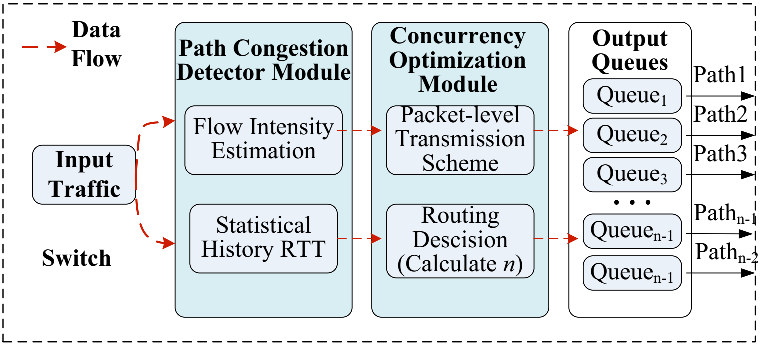


Fig. 4 The architecture of AAC

congestion state of end-to-end path. According to current and history delay information, the paths are divided into congested paths and uncongested ones. With the path states, AAC makes packet-level forwarding decisions for each arrival packet. Specifically, AAC calculates the optimal number of paths n based on the path diversity. Then AAC spreads packets on n paths to balance the long tailed delay and high link utilization.

AAC design involves several key challenges. Firstly, AAC needs to gather the leaf-to-leaf delay to distinguish the congested and uncongested paths. Secondly, the path adjustment strategy should cope with the rapid changes in network dynamics with limited overhead. Finally, AAC should be compatible with existing transport layer protocols for large-scale deployment in production data centers.

3.2 Leaf-to-leaf Delay Measurement

Modern data center networks are usually organized in multi-rooted tree topologies, in which the load balancing schemes split flows across multiple paths between the source leaf switch and the destination one. To obtain the path congestion state, AAC should measure the leaf-to-leaf delay between the source and destination switch. However, though it is not hard for end host to measure the round-trip-time (RTT), leaf-to-leaf delay measurement at switch is still a challenging task.

Traditional, delay measurement can be divided into active and passive measurement. In the active measurement, the probe packets are proactively injected into network. Though the active measurement can obtain accurate results, it inevitably introduces additional traffic overhead. On the other hand, the passive measurement only monitors the network traffic and measures the delay without any traffic overhead. Due to its passive feature, however, the passive measurement potentially suffers from the accuracy loss [18].

In our design, we propose an effective yet simple scheme on the leaf switch to accurately measure the leaf-to-leaf delay without additional traffic overhead. Specifically, the source leaf switch firstly records the sequence number and departure time of a data packet. Then, once receiving the corresponding ACK packet of the data packet, this switch calculates the path delay by subtracting the departure time of the data packet from the receiving time of the ACK packet. To limit the computing and memory overhead, the leaf switch measures the path delay every 100 µs. Furthermore, since the congestion on the reverse path has negative effect on the measurement accuracy, AAC lets ACK packets have higher priority than data packets to reduce the queueing delay of ACK packets [19].

Figure 5 illustrates how to measure the leaf-to-leaf delay at switch [20]. The leaf switch L_1 scatters packets across multiple paths to the destination switch L_2. The 1st data packet is forwarded to the core switch C_2, while the 2nd data packet is sent to C_1. If L_1 needs to measure the real-time delay of a path, it selects a data packet being forwarded to the path as the probe packet. For example, when the 2nd data packet arrives at L_1, it is chosen as probe packet to measure the delay of path $L_1 \rightarrow C_1 \rightarrow L_2$. Then L_1 records the sequence number and departure time of the 2nd data packet. When the ACK packet of the 2nd data packet is received by L_1, L_1 get the real-time delay of path $L_1 \rightarrow C_1 \rightarrow L_2$ by subtracting the departure time of the data packet from the receiving time of the ACK packet. It is worth noting that the data packet and its corresponding ACK may be transmitted on different paths. Fortunately, whether data packet and its ACK are transmitted on the same path or not, our design can measure global path delay.

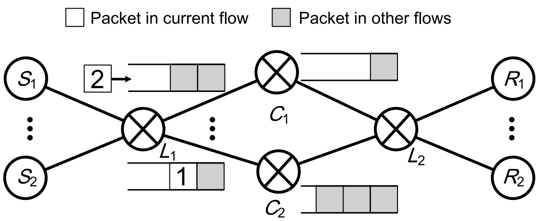
On the other hand, though L_2/L_3 switch cannot track transport layer header information like sequence number, fortunately, the commonly used switch can obtain these information in current production data center network. For example, the default load balancing scheme at datacenter switch is ECMP, which hashes each flow to one path according to 5-tuple information in TCP and IP header. Therefore, AAC also utilizes the information of TCP header (i.e., sequence number) to measure the leaf-to-leaf delay.

3.3 Tuning the Flow Concurrency

In the multipath transmission, a large number of concurrent paths provide high link utilization to reduce flow completion time under small path asymmetry, while easily suffering from the long tailed delay under large path asymmetries. Therefore, AAC elaborately tunes the flow concurrency under different degrees of path asymmetry. In this section, we use the continuous-time absorption birth-death Markov model to analyze the optimal flow concurrency [21–23].

We consider the typical case that m concurrent TCP flows are transferred on m paths, which include δ bad paths and m-δ good ones. For simplicity, we assume that the round trip time of good and bad paths are *BaseRTT* and *MaxRTT*

Fig. 5 RTT measurement on switch



, respectively. According to reference [24, 25], the empirical threshold for the *MaxRTT* is set as 2x average *RTT* of all paths. We summarize the key notations in Table 1.

In data center, most TCP flows are very small (normally less than 100KB [26]) and usually can be finished in their slow-start phase. For example, the transfer of a 40 KB flow only needs 5 rounds of RTT. During slow-start phase, the TCP congestion window exponentially increases in every round of RTT. Given the initial congestion window of k packets, the congestion window in the k-th round of RTT is 2^k [27, 28]. Then, for a flow with its size as F_{size} packets, the total number of RTTs r to finish transmission is

$$r = log_2\left(\frac{F_{size}}{k} + 1\right). \tag{1}$$

We assume that there are m TCP flows arriving at the same output of source leaf switch and their arrival times follow a Poisson process. Given the average packet arrival rate λ and the service rate μ, we get the flow intensity ρ as $\rho = \frac{\lambda}{\mu}$.

According to the history data trace, AAC can randomly select multiple paths, and the probability that each path becomes bad path is different. For example, according to reference [19], each path has a less than 10% probability of becoming a bad path. To simplify the analysis, we assume that there are m paths from the source leaf switch to the destination leaf switch, and only one path is a bad path. When the switch selects n paths from m paths, and the probability that the switch selects bad paths is $p = \frac{n}{m}$.

We assume that the link capacity of one good path and bad one are C and $\frac{C\times BaseRTT}{MaxRTT}$, respectively. For n parallel paths, we get the average service rate μ as

Table 1 Notations definition

Notations	Implication
F_{size}	The size of flow
p	Probability of selecting hotspot path from m paths
$BaseRTT$	The round trip time of good paths
$MaxRTT$	The round trip time of bad paths
FCT	Flow completion time
$\bar{\lambda}$	Average arrival rate
μ	Average service rate
ρ	Traffic intensity of each path
C	Link bandwidth
k	Initial window size in slow-start
λ_i	Average arrival rate when packet amount is i in the stationary state
μ_i	Average service rate when packet amount is i in the stationary state
ρ_i	The traffic intensity when packet amount is i in the stationary state
p_i	State distribution after the system reaches equilibrium
X	Degree of path diversity

$$\mu = (1-p) \times n \times C + p \times n \times C \times \frac{BaseRTT}{MaxRTT}. \tag{2}$$

We define the degree of path diversity X as $X = \frac{MaxRTT}{BaseRTT}$, according to the birth and death process of queuing theory, the equilibrium equation in any state can be calculated as

$$\begin{cases} \mu_1 \cdot p_1 = \lambda_0 \cdot p_0 & i = 0 \\ \lambda_{i-1} \cdot p_{i-1} + \mu_{i+1} \cdot p_{i+1} = (\lambda_i + \mu_i) \cdot p_i & i \geq 1, \end{cases} \tag{3}$$

where i and p_i denotes the number of packets and the state distribution after the system reaches equilibrium, respectively [29].

According Eq. (3), we have

$$\begin{cases} p_1 = \frac{\lambda_0}{\mu_0} \times p_0 & i = 0 \\ p_{i+1} = \frac{\lambda_i \lambda_{i-1} \cdots \lambda_0}{\mu_{i+1} \mu_i \cdots \mu_1} \times p_0 & i \geq 1. \end{cases} \tag{4}$$

For the sake of description, we use C_i to denote $\frac{\lambda_i \lambda_{i-1} \cdots \lambda_0}{\mu_{i+1} \mu_i \cdots \mu_1}$. Then distribution of the stationary state can be expressed as

$$p_i = C_i \times p_0, i \geq 1. \tag{5}$$

According to the probability distribution, we have

$$\sum_{i=0}^{\infty} p_i = \left(1 + \sum_{i=1}^{\infty} C_i\right) \times p_0 = 1. \tag{6}$$

Then we have

$$p_0 = \frac{1}{1 + \sum_{i=1}^{\infty} C_i}. \tag{7}$$

For one queue, it follows M/M/1/∞. Then we have $\lambda_i = \lambda$ and $\mu_i = \mu$ $(i \geq 0)$. Besides, the traffic intensity is $\rho = \frac{\lambda}{\mu}$ and C_i can be expressed as $(\frac{\lambda}{\mu})^i$. Therefore, Eq.(7) and (5) can be rewritten as

$$p_0 = \frac{1}{1 + \sum_{i=1}^{\infty} \rho^i} = \left(\sum_{i=0}^{\infty} \rho^i\right)^{-1} = \left(\frac{1}{1-\rho}\right)^{-1} = 1 - \rho, \tag{8}$$

$$p_i = \rho^i \times p_0 = (1-\rho) \times \rho^i, i \geq 1. \tag{9}$$

According to the distribution of queue length in the stationary state, the average queue length $\bar{L}$ can be caculated as

$$\bar{L} = \sum_{i=0}^{\infty} i \times p_i = \sum_{i=1}^{\infty} i \times (1-p) \times \rho^i = \frac{\rho}{1-\rho}. \tag{10}$$

Furthermore, we can leverage the Little formula to calculate the queueing delay t_q of packets as

$$t_q = \frac{\bar{L}}{\lambda} = \left(\frac{\rho}{1-\rho}\right) \times \frac{1}{\lambda} = \frac{1}{\mu - \lambda}. \tag{11}$$

Combining Eq. (1) and (11), we can obtain the average flow completion time ($\overline{FCT}$) as

$$\overline{FCT} = r \times t_q = \frac{log_2(\frac{F_{size}}{k} + 1)}{\mu - \lambda}. \tag{12}$$

To get the optimal flow completion time, we calculate the derivative of $\overline{FCT}$ with respect to n as

$$\begin{aligned} \frac{\mathrm{d}\overline{FCT}}{\mathrm{d}n} &= \frac{log_2(\frac{F_{size}}{k} + 1)}{(1-\rho)} \times \frac{1}{\frac{\mathrm{d}\mu}{\mathrm{d}n}} \\ &= \frac{log_2(\frac{F_{size}}{k} + 1)}{(\rho - 1) \times \mu^2} \times \frac{\mathrm{d}\mu}{\mathrm{d}n}. \end{aligned} \tag{13}$$

With Eq. (2), we calculate the derivative of μ with respect to n as

$$\begin{aligned} \frac{\mathrm{d}\mu}{\mathrm{d}n} &= \frac{C}{m \times X} \times \frac{\mathrm{d}((1 - X \cdot C) \cdot n^2 + X \cdot C \cdot n)}{\mathrm{d}n} \\ &= \frac{C}{m \times X} \times (2 \times n \times (1 - X) + m \times X). \end{aligned} \tag{14}$$

Then we let $\frac{\mathrm{d}\left(\overline{\mathrm{FCT}}\right)}{\mathrm{d}n} = 0$. In the right side of Eq. (13), only $\frac{\mathrm{d}\mu}{\mathrm{d}n} = 0$ can satisfy the condition. It is very easy to deduce $(2 \times n \times (1 - X) + m \times X) = 0$. Therefore, the optimal concurrency for AAC is given by:

$$n = \begin{cases} \left\lfloor \frac{m \times X}{2 \times (X-1)} \right\rfloor & X > 2; \\ m & 1 \le X \le 2. \end{cases} \tag{15}$$

We evaluate the correctness of the theoretical analysis by NS2 simulations. In this test, the sender leverages TCP as the underlying transport protocol. Besides, the flow size, *MaxRTT* and *BaseRTT* are 200 packets, 1000 µs and 100 µs, respectively. Other parameters are the same as the experimental scenario in Sect. II-B.

When the number of paths n increases from 1 to 10, we calculate the theoretical completion time FCT for a flow with size of 200 packets. The experimental value of FCT is consistent with the varying trend in NS2 simulation test (Fig. 6).

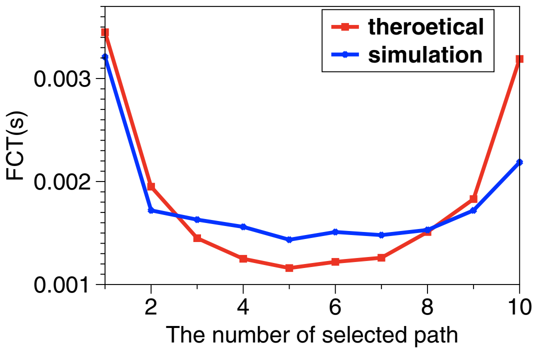


Fig. 6 Theoretical numeric and simulation comparison

3.4 AAC Algorithm

Based on above analysis, we obtain the optimal number of transmission paths. However, under the dynamic network traffic, RTT may change dynamically. It is unreasonable to adopt a fixed flow concurrency. Thus, we design the adjustment strategy shown in Algorithm 1, which consists of path congestion detector module and concurrency optimization module.

Algorithm 1: Adaptive Adjusting Concurrency Algorithm

Input: m /*The number of paths in DCN between leaf switches*/
Output: n /*The number of paths will be selected*/
Initialization;
$p_i.mode \leftarrow \emptyset$; $path[] \leftarrow allpaths$;
$n \leftarrow m$; $\delta \leftarrow 100\mu s$; $\delta t \leftarrow 40\mu s$;
/* Path congestion detection module */ ;
if *the timer is timeout* **then**
    $T_{BaseRTT} = +\infty$;$T_{MaxRTT} = 0; Sum = 0$;
    for *each path* $p_i \epsilon path[]$ **do**
        $T_{BaseRTT} = MIN(T_{BaseRTT}, p_i.rtt)$;
        $T_{MaxRTT} = MAX(T_{MaxRTT}, p_i.rtt)$;
        $Sum += p_i.rtt$;
    $T_{avg} = \frac{Sum}{m}$;
    $X = \frac{T_{MaxRTT}}{T_{BaseRTT}}$;
    for *each path* $p_i \epsilon path[]$ **do**
        if $p_i.rtt > 2 \times T_{avg}$ **then**
            $p_i.mode = congested$;
        else
            $p_i.mode = uncongested$;
/* Concurrency Optimization Module */ ;
if *the path state changes* **then**
    $n = \left\lfloor \frac{m \times X}{2 \times (X-1)} \right\rfloor$;
return n

3.4.1 Path Congestion Detector Module

The path congestion detector at the sender side of AAC periodically (e.g.100 µs) sends probe packets to measure the congestion state. If a path has the RTT larger than 2X average RTT of all paths, this path is deemed abnormal. Then, the abnormal path is deleted from the available path set, and AAC recalculates the optimal number of transmission path.

3.4.2 Concurrency Optimization Module

When the path state changes, AAC recalculates the optimal number of paths based on the current number of available paths, the congested paths and uncongested paths.

As a typical flow-based multi-path algorithm, ECMP randomly hashes the flow to one of the equivalent paths [30]. LetFlow uses flowlet as the switching granularity to randomly send flowlets to available paths. Thus, the time complexity of ECMP and Letflow is O(1). Since AAC needs to probe all m paths to adjust the flow concurrency, its time complexity is O(m). Fortunately, in the typical leaf-spine topology, the number of leaf-to-leaf paths m is the number of leaf switches, which is not very large.

4 Simulation Experiment

In this section, we conduct large-scale NS2 simulations to evaluate AAC performance. Firstly, we conduct the feature test to evaluate the basic performance of AAC. Secondly, we test AAC performance under symmetric and asymmetric topologies. Finally, we compare performance of AAC with the other state-of-the-art load balancing schemes such as ECMP, RPS, MPTCP and LetFlow in realistic datacenter workloads.

4.1 Feature Test

We firstly conduct NS2 simulations to test how AAC and RPS deal with path diversity under the network uncertainty. The experimental topology is shown in Fig. 7 [31]. There are 4 available paths path1 path4 from the one leaf switch to another leaf switch. The RTT of each path is 100 μs.The downlink and uplink bandwidth of leaf switch is 1 Gbps.

In this test, one sending host under one leaf switch sends a flow to a receiving host under another leaf switch. $Path_1$ is hotspot path with latency of 500 μs. Here, we compare the performances of RPS and AAC on the source leaf switch.

Figure 8 shows the instantaneous throughputs on path1–path4. As shown in the Fig. 8a, RPS randomly spreads packets across all paths. Thus, the instantaneous throughputs of 4 paths are almost equal. However, though path1 experiences

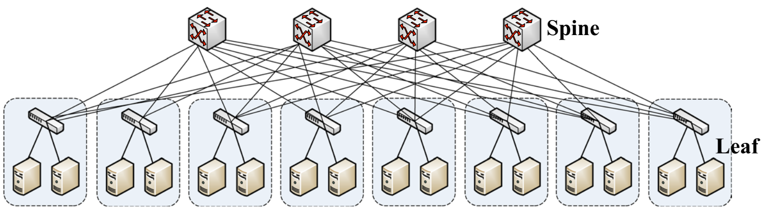


Fig. 7 Simple test topology

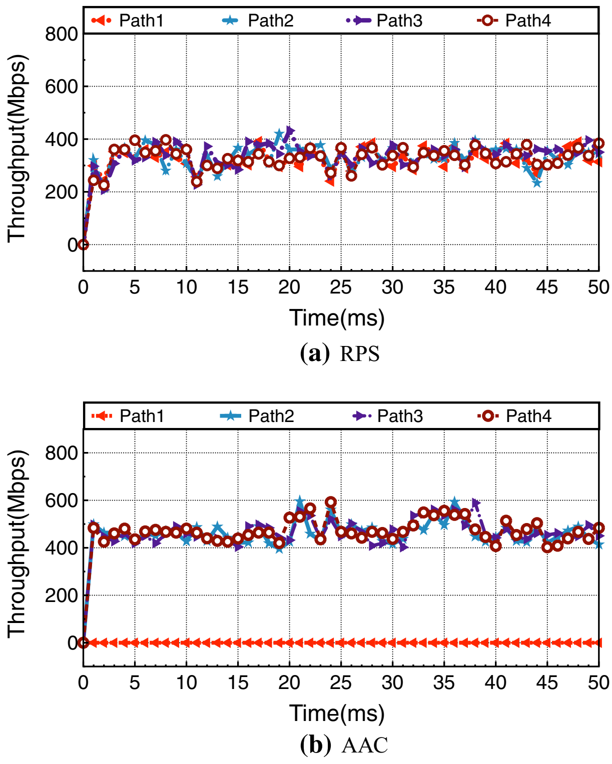
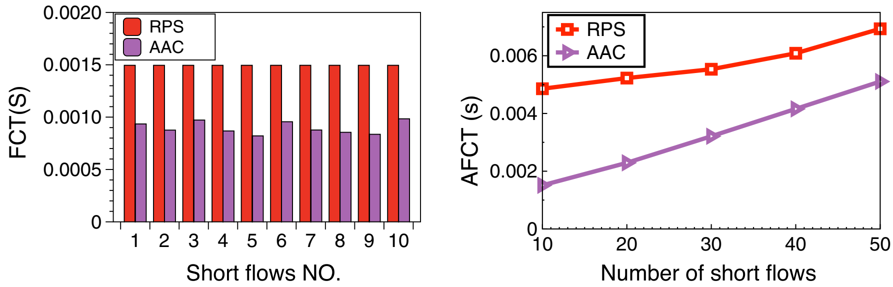


Fig. 8 RPS vs. AAC path throughput comparisons

link failure, RPS still sends packets on all paths, resulting in throughputs loss on all paths. The reason is that, when some packets are transferred on the slow path by RPS, these unlucky packets may lead to out-of-order issue at the receiver and thus unnecessary reduction of TCP congestion window at the sender side. As a result, even when only one path deteriorates, the instantaneous throughputs of four paths are decreased.

In contrast, as shown in Fig. 8b, AAC adaptively decreases the flow concurrency once a hotspot path occurs. After detecting the RTT diversity, AAC decreases the number of paths to 3 until it finally selects the fast paths. Since the packets automatically avoids the slow path, the instantaneous throughputs of the remaining three paths increase, showing good adaptability in load balancing.

Next, since most flows are short ones in data center traffic, we compare the performances of short flows under AAC and RPS. We test 10 short flows with 100 packets in a leaf-spine topology, which has 10 paths between two leaf switches. The round trip propagation delay of 9 paths and one hotspot path are 70 μs and 700 μs, respectively. Figure 9a shows that, since the data packets of each flow are evenly scattered on 10 paths under RPS, the completion time of each flow is almost same. Under AAC scheme, since the numbers of data packets passing



(a) 10 short-flows FCT **(b)** change number of short-flows

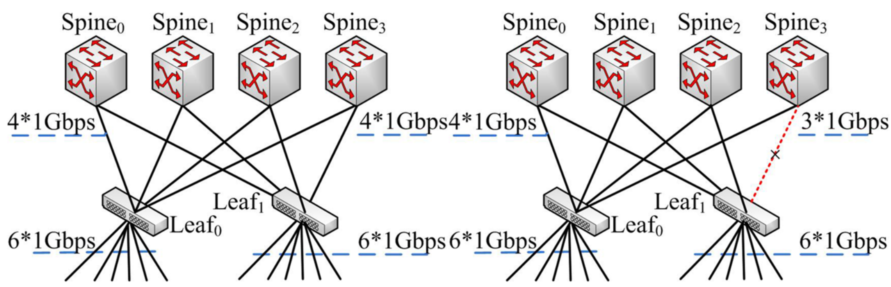
Fig. 9 RPS vs. AAC comparisons

through the hotspot path are different in each flow, the completion time of each flow is different. Figure 9b shows, with the increasing number of flows, AFCT of RPS and AAC increases accordingly. Nonetheless, AAC performs much better than RPS.

4.2 Performance Under Symmetric and Asymmetric Topologies

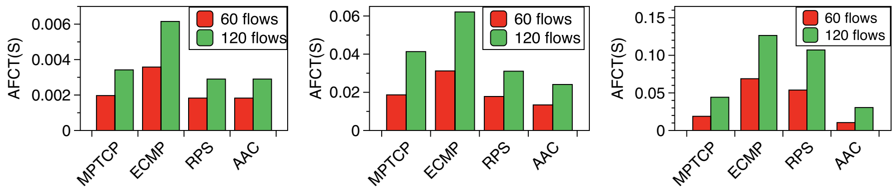
We test ACC performance under symmetric and asymmetric topologies. We use the leaf-spine topology with four paths between any pair of hosts. Figure 10a shows the symmetric topology, i.e., all paths have 20 µs latency and 1Gbps bandwidth. Figure 10b shows the asymmetric topology due to latency or bandwidth difference between paths. Under the RTT asymmetry, one path has the large RTT of 800 µs, while the others have 20 µs. For bandwidth asymmetry, only one path experiences the link failure and its bandwidth is decreased to 200Mbps. The switch buffer size is 100 packets.

We compare AAC with ECMP, MPTCP and RPS. ECMP is the standard flow-level load balancing mechanism in data center. Based on the hash result of the five-tuple in packet header, an outgoing port is selected for each flow. MPTCP [3]



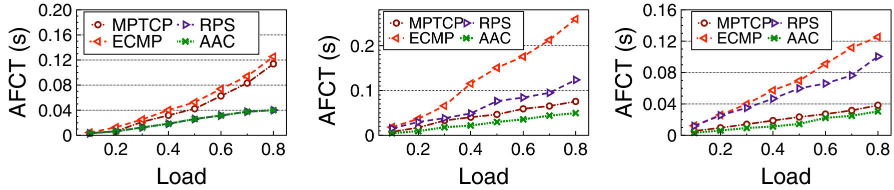
(a) Symmetric topology **(b)** Asymmetric topology

Fig. 10 Leaf-spine topology



(a) Symmetric topology **(b)** Link failure **(c)** RTT difference

Fig. 11 Comparison of AFCT under different scenarios



(a) Symmetric topology **(b)** Link failure **(c)** RTT difference

Fig. 12 AFCT of different workload

divides a TCP flow into 2 subflows. Each subflow has its own congestion window and uses ECMP to select its path independently.

In Figs. 11 and 12, we increase the number of flows and network load to test AFCTs of different protocols, respectively. Figures 11a and 12a shows the test results under symmetric topology. The average flow completion times (AFCTs) of all schemes increase with larger flow amount and network load. Since MPTCP and ECMP are flow-level load balancing schemes, they are hard to effectively make full use of the available paths when some flows finish their transmissions, resulting in low link utilizations and large AFCT. As the packet-level load balancing schemes, RPS and AAC fully utilize the link resources and achieve low AFCT under the symmetric topology.

Figures11b and 12b show the test results under bandwidth asymmetry while Figs. 11c and 12c evaluate performance of AAC under RTT asymmetry. Due to the out-of-order problem under bandwidth and RTT asymmetry, RPS experiences long AFCT. MPTCP obtains lower AFCT than RPS and ECMP, because the sublfows adjust their congestion windows according to congestion state on each path, thus achieving traffic balance without out-of-order issue. Since AAC adjusts the flow concurrency according to path diversity, it mitigates the impact of slow path and achieves the lowest AFCT.

4.3 Performance of TCP Friendliness

When load balancing schemes use multiple paths to transfer flows, it is important to ensure TCP friendliness. Here, we compare the performances of TCP friendliness

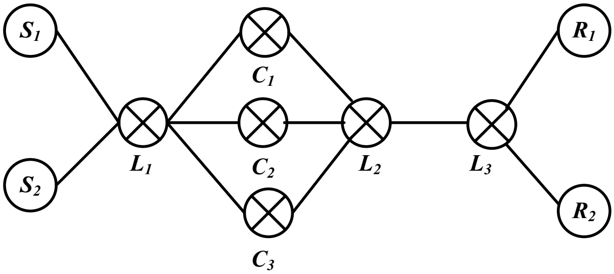
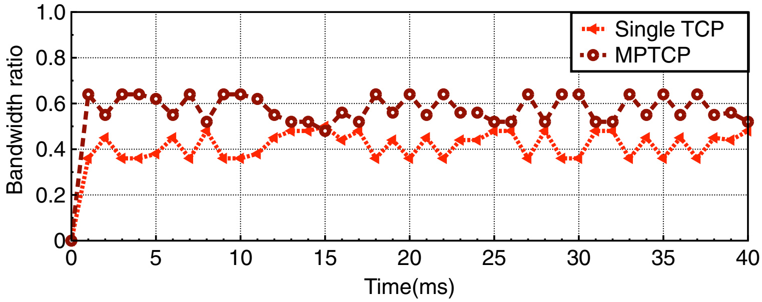
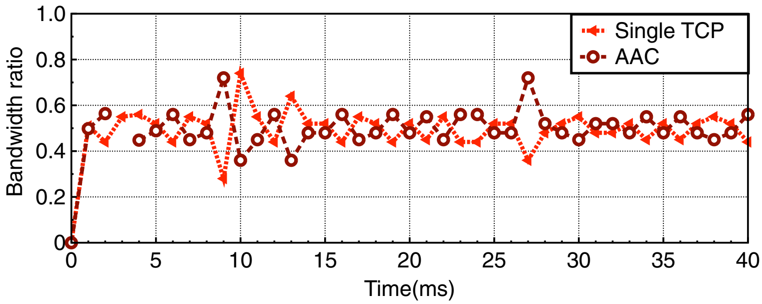


Fig. 13 Topology for testing fairness



(a) MPTCP VS. Single-TCP



(b) AAC VS. Single-TCP

Fig. 14 Performance comparison of TCP friendliness

under AAC and MPTCP+OLIA [32]. We build the topology with the shared bottleneck shown in Fig. 13. The path delay is set to 100 μs, and the link bandwidth is 1 Gbps. Firstly, two MPTCP subflows with 1000 packets are sent from S_1 to R_1. A single TCP flow is sent from S_2 to R_2.

The simulation results are shown in Fig. 14a. Though two subflows share the bottleneck link $L_2 \rightarrow L_3$ with the single TCP flow, with the help of OLIA, MPTCP senders couples the two subflows after detecting the shared bottleneck link. Thus, the total throughput of two subflows is not much larger than of the single TCP flow. Secondly, two TCP flows f_1 and f_2 are respectively sent from S_1 and S_2, competing at the bottleneck link $L_2 \rightarrow L_3$. AAC works at L_1 to balance the traffic from f_1 and f_2. Figure 14b shows that, on the bottleneck link of $L_2 \rightarrow L_3$, the throughput of f_1 is almost equal to f_2, exhibiting good TCP friendliness.

4.4 Large-Scale Simulation Test

We conduct large-scale simulation test to evaluate AAC's performance under realistic datacenter workloads. We use a leaf-spine topology with 16 core switches and 16 leaf switches. Each leaf connects 20 end-hosts. The link capacity, the round trip propagation delay and buffer size of switches are 1Gbps, 100 μs and 200 packets, respectively. We set the round trip propagation delay of one randomly selected path as 1000 μs to produce delay asymmetry. In this test, we also test the performance of LetFlow, which uses the flowlet as the switching unit. In LetFlow, the switch randomly assigns a available path to a new flowlet when time interval between two adjacent packets belonging to the same flow is larger than a threshold, which is set as 500 μs [8].

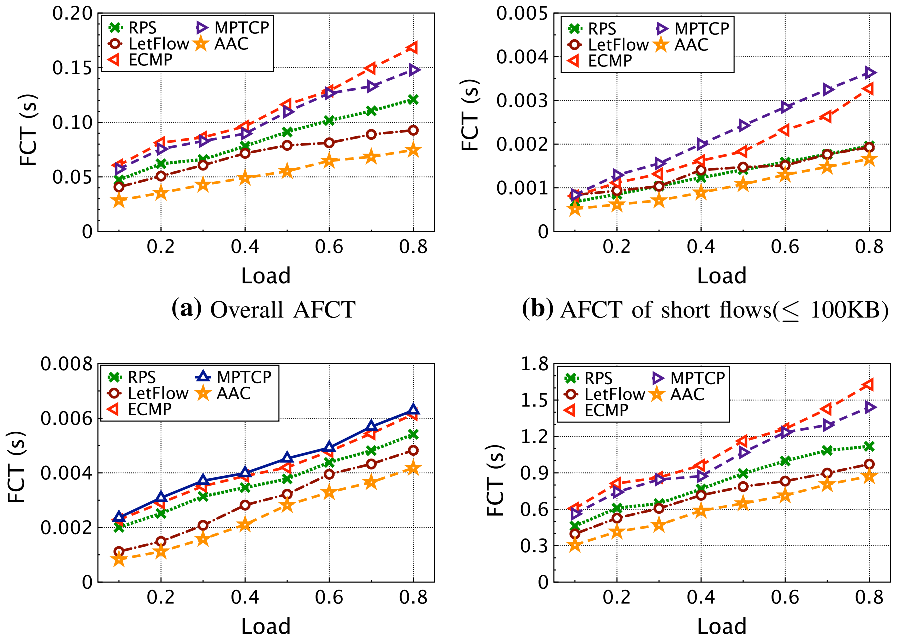
We use realistic workloads in production data centers. We consider the web search [24] and data mining [15] workloads, both of which exhibit heavy-tailed characteristics with a mixture of small and long flows. In the web search workload, over 95% of the bytes are from 30% of flows larger than 1MB. In the data mining workload, 95% of all bytes are from 3.6% flows that are larger than 35MB, while more than 80% of flows are less than 10KB. Flows are generated between random pairs of hosts following a Poisson process with load varying from 0.1 to 0.8 to thoroughly evaluate AACs performance in different traffic conditions.

Similar to previous work, we use flow completion time (FCT) as the primary performance metric. In addition to the overall average FCT, we also take the FCT for small flows (< 100 KB) and large flows (> 10 MB) into consideration for better understanding of performance. The 99th percentile FCT of small flows is also an important performance metric for tailed latency.

Figures 15 and 16 show the FCT of small and large flows in web search and data mining workloads, respectively. As shown in Figs. 15a and 16a, for short flows, AAC reduces the average FCT by 40–45% compared with the other schemes when the load increases from 0.1 to 0.8. The results demonstrate the advantage of AAC in adjusting flow concurrency under path diversity. ECMP obtains larger AFCT than the other solutions, especially at high load, since some flows suffer from hash collisions and cause low link utilization.

Figures 15b and 16b show the average FCT of short flows with varying workload. Under the web search workload, AAC reduces AFCT by 21–56%. Under the data mining workload, AAC reduces AFCT by 40–66%. We observe that, compared with the other load balancing schemes, MPTCP performs poorly. The reason is that, when MPTCP makes full use of equal-cost multipath between end hosts to achieve high throughput, the short flows may be blocked by long flows, resulting in large AFCT for short flows.

Figures 15c and 16c show that tailed FCT of short flows. AAC significantly reduces the tail FCT by around 16–51% and 7–56% under data mining and web



(a) Overall AFCT **(b)** AFCT of short flows($\leq$ 100KB)

(c) 99^{th}-ile AFCT of short flows **(d)** AFCT of long flows($\geq$ 1MB)

Fig. 15 FCT statistics of web search workload

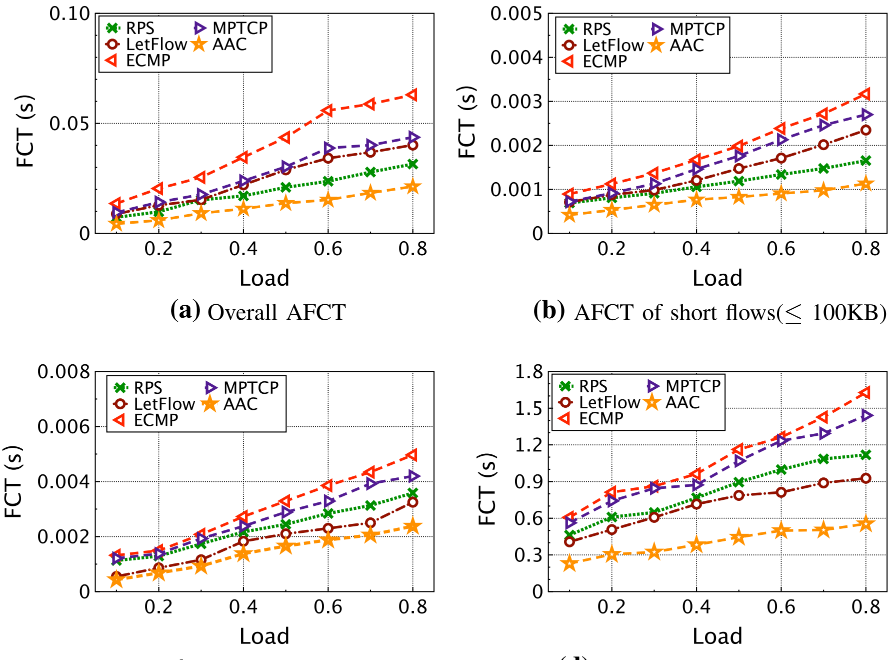
search load, respectively. Since the percent of long flows in web search is larger than data mining, the probability that short flows in web search are blocked by long flows is higher. Therefore, tailed FCT of short flows in web search is larger than that in data mining.

Finally, we plot the AFCT of long flows in Figs. 15d and 16d. The results show that, by adjusting the flow concurrency, AAC effectively avoids the packet reordering and low link utilization, therefore achieving low FCT for long flows.

4.5 Comparison with State-of-the-Art Approaches

In recent years, researchers have proposed many load balancing solutions, such as AG [33], Intflow [34], CAPS [25], Luopan [35]. Here, we conduct test to compare AAC with these schemes. The experiment topology is the same as that in Sect. IV-C. The state-of-the-art approaches are as follows.

- Hermes: Hermes [36] makes the switching decisions at packet level for long flows according to congestion conditions, and reroutes the short flows at a flow level. Due to the inflexibility feature, however, the flow-based mechanisms may lead to congestion and load imbalance.



(a) Overall AFCT **(b)** AFCT of short flows($\leq$ 100KB)

(c) 99^{th}-ile AFCT of short flows **(d)** AFCT of long flows($\geq$ 1MB)

Fig. 16 FCT statistics of data mining workload

- CAF: CAF [37] proactively measures available bandwidth at the end-hosts. Based on the measurement result, CAF sets a proper congestion window that matches the state of the new path.
- AG: AG [33] adaptively adjusts switching granularity under different degree of topology asymmetry. To achieve flexibility and resiliency to asymmetry, AG is sensitive to path latency and periodically adjusts its switching granularity. AG randomly selects paths for each switching unit.
- SPLB: SPLB [38] employs a dual channel architecture, which logically partitions a physical link into control and data channels. SPLB determines the appropriate transmission route of each packet immediately, without introducing any control loop durations.
- Intflow: Intflow [34] integrates per-packet and per-flowlet switching. IntFlow reacts to congestion and failures timely based on flow status to achieve proactive rerouting, while performing cautious rerouting for flowlet switching.
- HMMLB: HMMLB [39] utilizes the hidden Markov Model to select paths for data flows with small time cost, and approximates the same network throughput rate as a traditional centralized load balancing algorithm.

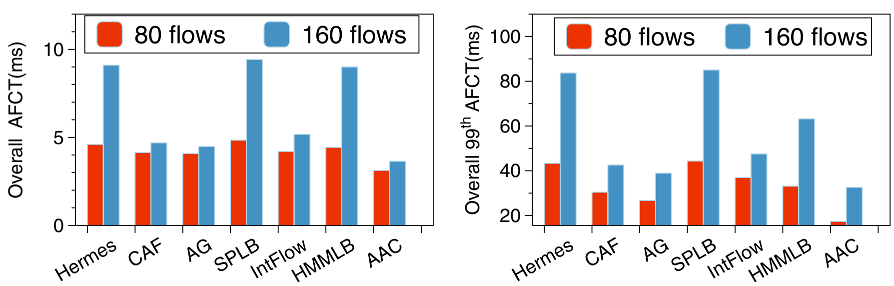
We use the 8×8 leaf-spine network topology to test FCT of different number of flows, which are generated between random pairs of hosts following a Poisson process. We evaluate the performance of recent proposed approaches and AAC.

Figure 17a shows that, AAC obtains 18–57% AFCT improvements compared with the other schemes. The overall performance SPLB is relatively poor, since it needs stand-in packets to probe the congestion state. As shown in Fig. 17b, AAC also obtains the lowest tail FCT. The reason is two-fold. On the one hand, AAC makes switching decisions according to the global network congestion. On the other hand, AAC adjusts the number of paths according to congestion conditions to effectively alleviate packet reordering and link under-utilization.

5 Mininet Implementation

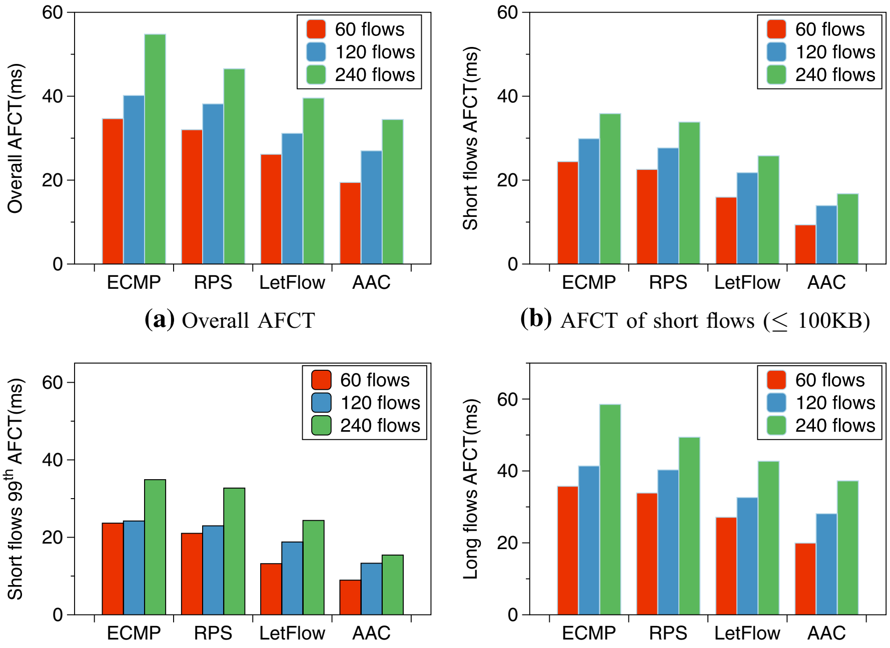
We implement AAC on Mininet, a high-fidelity network emulation framework built on Linux container based virtualization [36]. Mininet creates a virtual network, running real Linux kernel, switch and application code on a single machine. We use Mininet 2.3.0 to create the leaf-spine topology with 20 equal cost paths between the leaf and spine switches. We respectively set the link bandwidth to 20Mbps and delay to 5ms as recommended in [40]. BMv2 is installed as the software programmable switch with the buffer size of 256 packets. The overall traffic obeys the heavy-tailed distribution in web server workload as illustrated in [40]. Besides, We set the delay of two paths as 100ms to produce asymmetric topology.

Figure 18a shows that, with the increasing number of flows, AAC reduces the AFCT of all flows by 32–46%, 26–42%, 13–29% over ECMP, RPS and LetFlow, respectively. As shown in Fig. 17b and c, AAC effectively improves the AFCT and tailed FCT of short flows over ECMP, RPS and LetFlow. Figure 18d shows that AAC improves the FCT of long flows by 32–44%, 25–41%, 12–26% over ECMP, RPS and LetFlow, respectively. With more flows, AAC achieves better performance compared with the other schemes by adaptively adjusting the flow concurrency. ECMP and LetFlow suffer from the long-tailed delay and low link utilization because of the inability to flexibly reroute flows. The packet reordering issue degrades RPS's performance.



(a) Overall AFCT **(b)** 99^{th}-ile AFCT of overall flows

Fig. 17 Comparison with recent approaches



(a) Overall AFCT **(b)** AFCT of short flows ($\leq$ 100KB)

(c) 99^{th}-ile AFCT of short flows **(d)** AFCT of long flows($\geq$ 1MB)

Fig. 18 FCT on mininet implementation

6 Releated Work

With the rapid increase of traffic, load balancing has become a hot research issue in data center networks. To provide large bisection bandwidth and achieve good performance, a wide range of solutions are proposed to balance traffic across multiple paths. The existing load balancing schemes are based on flow granularity, flowlet granularity and packet granularity.

The representative of per-packet load balancing scheme is RPS [5], which splits flows at packet level to make full use of all available paths. Data packets are randomly scattered on the switch to all equal cost paths. RPS has fewer out-of-order packets, good load balancing performance, and high link utilization under symmetrical topology. However, RPS is prone to experience packet disorder under an asymmetric topology, leading to suboptimal performance.

DRILL [6] makes per-packet decisions to distribute load at each switch based on local queue occupancies. However, under asymmetric topology, DRILL is prone to experience packet reordering due to the difference between the local and end-to-end congestion states [34]. MMPTCP [26] uses a combined transmission method of RPS and MPTCP to improve network utilization. It dynamically adjusts the fast retransmission threshold according to the topology information to prevent false retransmissions caused by disorder. Detail [13] is a cross-layer scheme that considers packet priority and load balancing to reduce tail delay. QDAPS [41] is a packet-level load balancing mechanism based on queuing delay. FastPass [42] uses a central controller to allocate transmission time slot and transmission path for each packet. As a per-packet load balancing scheme, Hermes also easily leads to reordering especially under the asymmetric topology [43].

In a word, these packet-based multipath schemes potentially make short flows experience queueing delay and easily lead to reordering especially under the asymmetric topology[5, 26, 44]. Unlike aforementioned solutions, when rerouting event occurs due to path congestion, our approach AAC proactively measures path congestion information at the sending leaf swtiches. According to path diversity, AAC rationally adjusts the number of paths to improve link utilization under small path asymmetry and reduce tail latency under large path asymmetry.

7 Conclusion

We proposed a novel asymmetry-aware load balancing scheme AAC that reduces flow completion time and simultaneously improves link utilization. Specifically, AAC measures the leaf-to-leaf delay at leaf switch to adjust the flow concurrency according to the degree of path asymmetry. Moreover, AAC is deployed only at the leaf switches, without modification at thousands of servers.

Acknowledgements This work is supported by the Natural Science Foundation of Hunan Province, China (No. 2018JJ2084), National Natural Science Foundation of China (No. 61872387), and Project of Foreign Cultural and Educational Expert (No. G20190018003).

References

1. Huang, Q., Jin, X., Lee, P.P.C., et al.: Setchvisor: Robust network measurement for software packet processing. In: Proc. ACM Special Interest Group on Data Communication, pp. 113–126 (2015)
2. Bredel, M., Bozakov, Z., Barczyk, A., et al.: low-based load balancing in multipathed layer-2 networks using OpenFlow and multipath-TCP. In: Proc. Hot Topics in Software Defined Networking, pp. 213–214 (2014)
3. Hopps, C.: Analysis of an equal-cost multi-path algorithm. RFC 2992, November, (2000). http://www.ietf.org/rfc/rfc2992.txt
4. Raiciu, C., Barre, S., Pluntke, C., et al.: Improving datacenter performance and robustness with multipath TCP. ACM SIGCOMM Comput. Commun. Rev. **41**(4), 266–277 (2011)
5. Dixit, A., Prakash, P., Hu, Y. C., et al.: On the impact of packet spraying in data center networks. In Proc. IEEE INFOCOM, pp. 2130–2138 (2013)
6. Ghorbani, S., Yang, Z., Godfrey, P. B., et al.: Drill: Micro load balancing for low-latency data center networks. In: Proc. ACM Special Interest Group on Data Communication, pp. 225–238 (2017)
7. Zhang, H., Zhang, J., Bai, W., et al.: Resilient datacenter load balancing in the wild. In: Proc. ACM Special Interest Group on Data Communication, pp. 253–266 (2017)
8. Vanini, E., Pan, R., Alizadeh, M., et al.: Let it flow: Resilient asymmetric load balancing with flowlet switching. In: Proc. USENIX Symposium on Networked Systems Design and Implementation, pp. 407–420 (2017)
9. Chen, G., Lu, Y., Meng, Y., et al.: Fast and cautious: Leveraging multi-path diversity for transport loss recovery in data centers. In: Proc. USENIX Annual Technical Conference, pp. 29–42 (2016)

10. Tso, F. P., Hamilton, G., Weber, R., et al.: Longer is better: exploiting path diversity in data center networks. In: Proc. International Conference on Distributed Computing Systems, pp. 430–439 (2013)
11. Alizadeh, M., Edsall, T., Dharmapurikar, S., et al.: CONGA: distributed congestion-aware load balancing for datacenters. In: Proc. ACM Conference on SIGCOMM, pp. 503–514 (2014)
12. Cao, Y., Xu, M., Fu, X., et al.: Explicit multipath congestion control for data center networks. In Proc. ACM Conference on Emerging Networking Experiments and Technologies, pp. 73–84 (2013)
13. Zats, D., Das, T., Mohan, P., et al.: DeTail: reducing the flow completion time tail in datacenter networks. In: Proc. ACM SIGCOMM on Applications, Technologies, Architectures, and Protocols for Computer Communication, pp. 139–150 (2012)
14. Zhang, J., Zhang, D., Huang, K.: Improving datacenter throughput and robustness with Lazy TCP over packet spraying. Comput. Commun. **62**, 23–33 (2015)
15. Wang, W., Sun, Y., Salamatian, K., et al.: Adaptive path isolation for elephant and mice flows by exploiting path diversity in datacenters. IEEE Trans. Netw. Serv. Manag. **13**(1), 5–18 (2016)
16. Alizadeh, M., Kabbani, A., Edsall, T., et al.: Less is more: trading a little bandwidth for ultra-low latency in the data center. In: Proc. USENIX Symposium on Networked Systems Design and Implementation, pp. 253–266 (2012)
17. Carpio, F., Engelmann, A., Jukan, A.: DiffFlow: differentiating short and long flows for load balancing in data center networks. In: Proc. IEEE Global Communications Conference, pp. 1–6 (2016)
18. Lee, C., Park, C., Jang, K., et al.: Accurate latency-based congestion feedback for datacenters. In: Proc. USENIX, pp. 403–415 (2015)
19. Mittal, R., Lam, V.T., Dukkipati, N., et al.: TIMELY: RTT-based congestion control for the datacenter. ACM SIGCOMM Comput. Commun. Rev. **45**(4), 537–550 (2015)
20. Zou, S., Huang, J., Wang, J., et al.: Improving TCP Robustness over asymmetry with reordering marking and coding in data centers. In: Proc. International Conference on Distributed Computing Systems, pp. 57–67 (2019)
21. Alizadeh, M., Yang, S., Sharif, M., et al.: Pfabric: minimal near-optimal datacenter transport. ACM SIGCOMM Comput. Commun. Rev. **43**(4), 435–446 (2013)
22. Munir, A., Qazi, I.A., Uzmi, Z.A., et al.: Minimizing flow completion times in data centers. In: Proc. IEEE INFOCOM, pp. 2157–2165 (2013)
23. Noormohammadpour, M., Raghavendra, C.S.: Datacenter traffic control: understanding techniques and tradeoffs. IEEE Commun. Surv. Tutor. **20**(2), 1492–1525 (2017)
24. Alizadeh, M., Greenberg, A., Maltz, D. A., et al.: Data center TCP (DCTCP). In: Proc. ACM SIGCOMM, pp. 63–74 (2010)
25. Hu, J., Huang, J., Lv, W., et al.: CAPS: coding-based adaptive packet spraying to reduce flow completion time in data center. IEEE/ACM Trans. Netw. **27**(6), 2338–2353 (2019)
26. Kheirkhah, M., Wakeman, I., Parisis, G.: MMPTCP: a multipath transport protocol for data centers. In: Proc. IEEE INFOCOM, pp.1–9 (2016)
27. Zhangy, W., Lingy, D., Zhangy, Y., et al.: Achieving optimal edge-based congestion-aware load balancing in data center networks. In: Proc. IEEE Networking Conference, pp. 109–117 (2020)
28. Sen, S., Shue, D., Ihm, S., et al.: Scalable, "Optimal flow routing in datacenters via local link balancing". In: Proc. ACM Conference on Emerging Networking Experiments and Technologies, pp. 151–162 (2013)
29. Serfozo, R.F.: An equivalence between continuous and discrete time Markov decision processes. Oper. Res. **27**(3), 616–620 (1979)
30. Kabbani, A., Sharif, M.: Flier: flow-level congestion-aware routing for direct-connect data centers. In: Proc. IEEE INFOCOM, pp. 1–9 (2017)
31. Katta, N., Hira, M., Ghag, A., et al.: CLOVE: How I learned to stop worrying about the core and love the edge. In: Proc. the 15th ACM Workshop on Hot Topics in Networks, pp. 155–161 (2016)
32. Huang, J., Li, W., Li, Q., et al.: Tuning high flow concurrency for MPTCP in data center networks. J. Cloud Comput. **9**(1), 1–15 (2020)
33. Liu, J., Huang, J., Li, W., et al.: AG: adaptive switching granularity for load balancing with asymmetric topology in data center network. In: Proc. International Conference on Network Protocols, pp. 1–11 (2019)
34. Shi, Q., Wang, F., Feng, D.: IntFlow: integrating per-packet and per-flowlet switching strategy for load balancing in datacenter networks. IEEE Trans. Netw. Serv. Manag. **17**(3), 1377–1388 (2020)
35. Wang, P., Trimponias, G., Xu, H., et al.: Luopan: sampling-based load balancing in data center networks. IEEE Trans. Parallel Distrib. Syst. **30**(1), 133–145 (2018)

36. Floyd, S., Jacobson, V.: Random early detection gateways for congestion avoidance. IEEE/ACM Trans. Netw. **1**(4), 397–413 (1993)
37. Zou, S., Huang, J., Jiang, W., et al.: Achieving high utilization of flowlet-based load balancing in data center networks. Future Gener. Comput. Syst. **108**, 546–559 (2020)
38. Xu, C., Yuan, T., Zhang, H., et al.: Dual channel per-packet load balancing for datacenters. In: Proc. IEEE INFOCOM, pp. 157–164 (2020)
39. He, B., Zhang, D., Zhao, C.: Hidden Markov Model-based Load Balancing in Data Center Networks. Comput. J. **63**(10), 449–1462 (2020)
40. Xu, H., Li, B.: RepFlow: minimizing flow completion times with replicated flows in data centers. In: Proc. IEEE INFOCOM on Computer Communications, pp. 1581–1589 (2014)
41. Huang, J., Lv, W., Li, W., et al.: QDAPS: queueing delay aware packet spraying for load balancing in data center. In: Proc. International Conference on Network Protocols, pp. 66–76 (2018)
42. Perry, J., Ousterhout, A., Balakrishnan, H., et al.: "Fastpass: a centralized" zero-queue "datacenter network". In: Proc. SIGCOMM, pp. 307–318 (2014)
43. Hu, J., Huang, J., Lv, W., et al.: TLB: Traffic-aware load balancing with adaptive granularity in data center networks. In: ICPP 2019: Proceedings of the 48th International Conference on Parallel Processing, pp. 1–10 (2019)
44. Katta, N., Ghag, A., Hira, M., et al.: Clove: congestion-aware load balancing at the virtual edge. In: Proc. the 13th International Conference on Emerging Networking Experiments and Technologies (2017)

Weimin Gao received the B.E. degrees from Nanhua University, China, in 1999 and the master's degrees from the School of Information Science and Engineering, Hunan University, China, He is currently pursuing the Ph.D. degree with the School of Computer Science and Engineering, Central South University, China. His current research interest is data center network.

Jiawei Huang (M'07) received the bachelor's degree from the School of Computer Science, Hunan University, in 1999, and the master's and Ph.D. degrees from the School of Computer Science and Engineering, Central South University, China, in 2004 and 2008, respectively. He is currently a Professor with the School of Computer Science and Engineering, Central South University. His research interests include performance modeling, analysis, and optimization for data center networks.

Shaojun Zou is currently working toward the Ph.D. degree in the Department of Computer Science and Engineering, Central South University, Changsha, China. His current research interests include congestion control and data center networks.

Weihe Li is a Master Student in the School of Computer Science and Engineering, Central South University, China. His research interests include video streaming and data center networks.

Jianxin Wang (SM'12) received the B.E. and M.E. degrees in computer engineering and the Ph.D. degree in computer science from Central South University, Changsha, China, in 1992, 1996, and 2001 respectively. He is currently a Professor with the School of Computer Science and Engineering, Central South University. His current research interests include algorithm analysis and optimization, parameterized algorithm, bioinformatics, and computer network.

Jianer Chen (Senior Member, IEEE) received the Ph.D. degree in computer science from the Courant Institute, New York University, in 1987, and the Ph.D. degree in mathematics from Columbia University in 1990. He is currently a Professor of computer science with Texas A&M University at College Station. His main research interest includes computer algorithms and their applications. His current research projects include exact and parameterized algorithms, computer graphics, computer networks, and computational biology.

Authors and Affiliations

Weimin Gao[1,2] · **Jiawei Huang**[1] · **Shaojun Zou**[1] · **Weihe Li**[1] · **Jianxin Wang**[1] · **Jianer Chen**[3]

Weimin Gao
gwm@hnit.edu.cn

Shaojun Zou
zoushj@csu.edu.cn

Weihe Li
weiheli@csu.edu.cn

Jianxin Wang
jxwang@csu.edu.cn

Jianer Chen
chen@cse.tamu.edu

1 School of Computer Science and Engineering, Central South University, Changsha 410083, China

2 Department of Computer and Information Science, Hunan Institute of Technology, Hengyang 421002, China

3 Department of Computer Science and Engineering, Texas A&M University, College Station, TX 77843, USA