Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Congestion-aware adaptive forwarding in datacenter networks

Jiao Zhang [a,b,*], Fengyuan Ren [c], Tao Huang [a], Li Tang [c], Yunjie Liu [a]

[a] State Key Laboratory of Networking and Switching Technology, BUPT, China
[b] School of Information and Communication Engineering, BUPT, China
[c] Dept. of Computer Science and Technology, Tsinghua University, China

ARTICLE INFO

Article history:
Received 20 June 2014
Received in revised form 24 January 2015
Accepted 28 January 2015
Available online 13 February 2015

Keywords:
Datacenters
Adaptive forwarding
Goodput

ABSTRACT

Datacenters employ the scale-out model to achieve scalability. This model requires parallelism in the underlying workload. Therefore, high bisection bandwidth is required to support intensive communications between servers. Several new datacenter architectures have been designed to provide redundant bandwidth. Currently, it is critical to design a mechanism to efficiently utilize the abundant bandwidth. In this paper, we propose a distributed Congestion-Aware Adaptive foRwording (CAAR) protocol to balance traffic load only depending on the local queue length information. CAAR allows flows to select under-utilized paths to forward packets. It is theoretically proved to be stable if the arrival rates are within the network throughput region. Simulation results under diverse datacenter topologies and communication patterns validate that CAAR achieves higher aggregate goodput compared with random and static routing protocols.

1. Introduction

Most datacenters employ the scale-out model to achieve scalability [1]. One task is usually completed by many servers together to achieve higher performance. It is stated that 80% of the services in datacenters require 10–100 servers to cooperate and 20% require more than 100 servers [2]. Therefore, intensive communications between servers exist in datacenter networks. Several architectures with high bisection bandwidth[1] have been designed to provide the bandwidth resource required by the intensive communications [4,5,3]. However, how to efficiently use the bandwidth resource is still an open problem.

Unbalanced traffic load across different links results in inefficient bandwidth utilization. It is possible that several paths are congested while some others are under-utilized or even idle. Such situation will lead to unfairness among different flows. Flows along under-utilized paths will finish earlier than other flows with the same flow size but along congested paths. Since a lot of services are cooperated by many servers together, the response time is decided by the slowest flow. In such kind of services, it is desirable that a set of flows are finished near the same time. For example, in the Partition/Aggregation workflow [6–8], the aggregators need to collect all the results from the lower-level workers and then generate the final results. Therefore, the latency of the final result will be dragged by the slowest flow, or the quality of the result degrades without aggregating the results sent from the slow workers.

There are mainly two research directions to utilize the redundant bandwidth in datacenters. One is designing multipath transport layer protocols. The typical work of this type is MPTCP [9]. MPTCP utilizes redundant bandwidth by splitting a flow into multiple subflows. Each subflow adjusts its own congestion window. However, it is an end-to-end protocol and thus can only response to congestion on the magnitude of Round Trip Time (RTT). While most of flows have quite short length in datacenters [7]. Therefore, balancing load across multiple paths at the transport layer is not fast enough.

The other direction is employing multipath forwarding protocols. Hop-by-hop forwarding mechanisms have the potential ability to quickly shift traffic bursts from congested paths to under-utilized ones and thus fully utilize the redundant bandwidth. Some work has been done to balance load in datacenters. ECMP [10] is a widely known flow-level random forwarding protocol that could utilize multipath bandwidth. However, since flow sizes are various and communication patterns are diverse in datacenters, possibly some flows will be randomly routed to some congested links even if there are some other idle links. To overcome the drawbacks of ECMP, Hedera [11] makes use of the feature that

* Corresponding author at: Office 738, 3rd Teaching Building, Beijing University of Posts and Telecommunications, 100876, China.

E-mail addresses: jiaozhang@bupt.edu.cn (J. Zhang), renfy@csnet1.cs.tsinghua.edu.cn (F. Ren), htao@bupt.edu.cn (T. Huang), tangli@csnet1.cs.tsinghua.edu.cn (L. Tang), liuyj@chinaunicom.cn (Y. Liu).

[1] Bisection bandwidth denotes the minimal total bandwidth of the links to be removed to partition a network into two parts of equal size [3].

http://dx.doi.org/10.1016/j.comcom.2015.01.020
0140-3664/© 2015 Elsevier B.V. All rights reserved.

the number of elephant flows in datacenters is small, and employs a centralized controller to reselect under-utilized paths for the long flows if they encounter congestion. However, in Hedera, the centralized controller has to collect flow-level information to find the elephant flows in real time, which brings large overhead. Recently, packet-level random forwarding is proposed to reduce the tail of the flow latency [12]. Since each packet can choose its own path, traffic load can be well balanced across multiple paths. However, packet-level random forwarding possibly leads to a lot of out-of-order packets. Larger memory at the end servers is required to accommodate the out-of-order packets [12].

In this work, we propose CAAR to purposely forward packets to balance the traffic with many short bursts across multiple links. The main idea is that each flow selects the most under-utilized path. If the selected path becomes congested during transmission, then the flow will be redirected to another under-utilized path. CAAR can responsively adapt to the traffic variation in datacenters and avoid packets reordering when no congestion happens.

Using theoretical analysis, it is proved that CAAR protocol is stable, and is throughput-optimal in terms of that it can fully utilize the redundant bandwidth. This theoretical result explores the possibility of making a tradeoff between protocol complexity and bandwidth utilization.

There are possibly a few out-of-order packets in CAAR since a flow will change its path if the current path becomes congested. In datacenters, the traffic is mixed of few long flows and a large number of short flows [13]. It is not necessary to redirect short flows since they can be finished quite quickly, and the probability of two long flows selecting the same path is quite small. Thus, we propose CAAR without reordering mechanism which does not reselect paths for flows during their transmission.

We implement both CAAR and CAAR without reordering on the ns-2 platform, and evaluate their performance in different datacenter topologies, including FatTree [4], VL2 [14], and single-rooted tree. The results show that CAAR performs much better than static routing and ECMP. Besides, CAAR without reordering performs close to CAAR.

The main contributions of this work have threefold.

(1) A congestion-aware adaptive forwarding protocol, CAAR, is proposed to fully utilize the redundant bandwidth in datacenters.
(2) Using theoretical analysis, CAAR is proved to be stable and throughput optimal.
(3) Considering the traffic characteristics, we propose CAAR without reordering mechanism and compares its performance with CAAR, static, and ECMP in various datacenter topologies.

The remainder of the paper is organized as follows. In Section 2, the related work and motivation is described. The network model is presented in Section 3. Section 4 presents the proposed scheme CAAR in detail. In Section 5, we prove that CAAR is stable when the arrival traffic is within the throughput region. Section 6 evaluates the proposed CAAR algorithm through simulations. Finally, the paper is concluded in Section 7.

2. Related work and motivation

2.1. Related work

Numerous forwarding/routing protocols have been developed for sorts of networks, including Internet, interconnection networks and datacenters. Here, some most related work in datacenters are summarized.

Static routing. Static routing is calculated in advance and keeps unchanged. In [4], Al-Fares et al. proposed a FAT-tree architecture for datacenters and designed a deterministic routing for FAT-tree topology. Guo et al. in [3] also proposed a deterministic source routing for their DCell architecture. In [15], Yuan et al. theoretically studied the deterministic routing for FatTree interconnection. They analyzed the lower bound of the oblivious performance ratio for different fat-trees and developed optimal deterministic single-path and multipath routing schemes in terms of oblivious performance ratio.

Random forwarding. Static routing is simple, but it cannot fully utilize the redundant links in datacenters. Greenberg et al. in [14] proposed balancing load by randomly spreading traffic across all the available links without considering any other factors. In [16], redundant paths are computed and then merged into a set of VLANs. Each packet is randomly sent to one of the VLANs that can reach the destination. However, random forwarding cannot perform well under some communication patterns and flow size distributions due to its blindness. In DeTail [12], packet-level randomized routing is proposed to reduce the tail of the flow latency. Traffic load can be well balanced across multiple paths since each packet can choose its own path. However, large memory at the end hosts is required to accommodate out-of-order packets [12].

Adaptive forwarding. Adaptive forwarding algorithms make a tradeoff between static routing and random forwarding. They employ some local information to change forwarding decisions to adapt to the traffic variation. The adaptive forwarding mechanisms can be classified into centralized and decentralized.

Hedera [11], MicroTE [17] and Fastpass [18] are typical centralized mechanisms. In Hedera [11]. short flows are randomly forwarded, while each large flow is assigned a under-utilized path computed by a heuristic algorithm to balance load. However, to differentiate whether a flow is large or not, switches need to record the size of every flow. Besides, the scheduler is fundamentally limited in its response time since it has to retrieve statistics, comput routing paths and install them. Whenever a flow's size exceeds a threshold or it persists for some time, it is considered to be a large flow. At last, Hedera assumes exponentially distributed flow sizes and Poisson arrival, which does not comply with the traffic characteristics of datacenters [19,14]. MicroTE [17] also employs a centralized controller as Hedera does. The centralized controller is used to track the predictable traffic between servers connected with the same ToR switch and route the traffic optimally. The remaining unpredictable traffic is then routed along weighted equal-cost multipath routes, where the weights reflect the available capacity after the predictable traffic has been routed. Various services are being developed in datacenters. It is difficult to predict traffic between servers. Also, the latency caused by the centralized controller [20] could not be neglected. Fastpass [18] uses a centralized arbiter to determine the time at which each packet should be transmitted as well as the path to use for that packet. It mainly focuses on guaranteeing zero-queue. The scalability of Fastpass is limited by the centralized arbiter that needs to deal with all the packets.

Many decentralized dynamic routing algorithms are designed for ISP, such as MATE [21], FLARE [22], and TeXCP [23]. MATE [21] converges slowly and works on the premise of knowing a global network information [23]. TeXCP [23] works in the granularity of a packet rather than a flow [24], which might lead to lots of packets reordering. FLARE [22] mainly solves the reordering problem when splitting a flow across multiple paths. It measures the delay of each path and set the maximum delay difference between the parallel paths as a threshold. Only if the interval of two packets exceeds the threshold, they can be transmitted along different paths. Thus, the packet-reordering problem can be avoided. However, since it is difficult to exactly measure the path delay of data center networks that is in the granularity of microseconds, FLARE is not proper for data centers. However, they are not proper for data center networks. There are also some distributed dynamic routing schemes designed for data center networks, such as DARD [24]. However, DARD [24] is a host-based dynamic forwarding scheme proposed for data center networks. In DARD, every end host requires to monitor the states of all paths to other hosts, which will cause large overhead especially in large data center networks. DiFS [25] states that switch-based dynamic routing mechanism is more proper for data center networks and proposed a dynamic forwarding mechanism by modifying the function of switches. However, DiFS needs to count the transmitting bytes of each flow, which will incur much overhead. The author states that OpenSketch can be used to reduce the cost of measurement. Yet OpenSketch is based on the centralized controller of SDN.

2.2. Motivation

Static routing fails to efficiently utilize the rich bandwidth in datacenters. With respect to random forwarding, whether it can well balance load will be affected by many factors, such as network topology, communication pattern and the distribution of flow sizes. Even the random forwarding, Valiant Load Balancing (VLB), which is similar to ECMP, is customized for VL2. It could not perform well under some traffic patterns and flow size distributions.

Fig. 1 illustrates a partial topology, which likely appears in some datacenters, such as VL2. Assume that under a communication pattern, six flows pass through switch A, including four flows with destination D1 denoted by solid lines and two flows with destination D2 denoted by dashed lines. First, assume that all flows have the same length. If a flow-level forwarding algorithm is employed, the path of each flow will be determined by its first packet. Fig. 1(a) shows a possible spatial distribution under the control of a random forwarding protocol. For the four solid flows, they have two options for next hops, i.e., B and C. Statistically, two of the solid flows will select B and the other two select C. Similarly, since there are two next hops, C and D, for dashed flows, one of them will select C and the other select D. If the capacity of each path is $\mathcal{P}$ and the rate of each flow is $\frac{\mathcal{P}}{2}$, then link (A,C) will suffer congestion and drop some packets, while link (A,D) is under-utilized. This simple example indicates that generally random forwarding mechanisms could not evenly balance traffic load under asymmetric communication pattern, namely, the number of flows with different destinations is different. The main reason is that the forwarding of different flows is somewhat blindfold. If some information reflecting networks state can be collected, and a new arrival flow is transmitted to the next hop with the minimum load as shown in Fig. 1(b), then the traffic can be evenly distributed across multiple links. Therefore, the goodput can be improved and the loss ratio will be reduced.

Second, even if the communication pattern is symmetric, such as all-to-all communication, different flow sizes will impose a negative impact on the random forwarding. Statistically, under symmetric communication patterns, the number of flows passing each switch is equal. However, if the random forwarding mechanism is flow-based, different flow size will lead to unbalanced traffic distribution.

In summary, random forwarding can be easily implemented, but it is somewhat blindfold. Fully utilizing redundant bandwidth plays a great role in improving the user experience of cloud services. It is necessary to explore whether we can make a tradeoff between protocol complexity and bandwidth utilization.

3. Network model

Datacenters are comprised of severs, links and switches. Switches and servers are modeled as buffers. Hence, a datacenter can be modeled as a graph with S buffers (switches or servers) and N links as shown in Fig. 2. The set of buffers and links are denoted as $\mathcal{B}$ and $\mathcal{N}$ respectively, $|\mathcal{B}| = S$ and $|\mathcal{N}| = N$. Each buffer can accommodate at most B packets. $\lambda_j(t)$ denotes the instantaneous rate at which external traffic comes to buffer j. It equals 0 if buffer j has no external arrivals.

A buffer can be served by several links simultaneously. Define $\mathcal{R}_j$ as the set of neighbors of buffer j, $\mathcal{R}_j^d$ as the set of next hops of packets destined for d at buffer j. $\mathcal{S}_j$ is the set of links which can serve buffer j. The packets with destination d in buffer j can be directed to any buffer in $\mathcal{R}_j^d$. A forwarding policy determines to which buffer in $\mathcal{R}_j^d$ the packets in buffer j are routed. There exists a sequence of buffers starting from buffer j through which the traffic in buffer j can be routed to the destination server, which is indicated by buffer 0. The service rate of each link is r. Let μ_j^i be the fraction of time that link i spends serving buffer j and $Q_j(t)$ be the length of buffer j at time t.

The proposed CAAR scheme operates in slotted time. Slots are normalized to integral units. Let T be the slot length. Time slot n refers to the time interval $(nT, (n+1)T], n \in \{0, 1, 2, \ldots\}$. If link i is assigned to transfer the traffic in buffer j to k continuously during a time slot n, then an amount of traffic rT will be transmitted

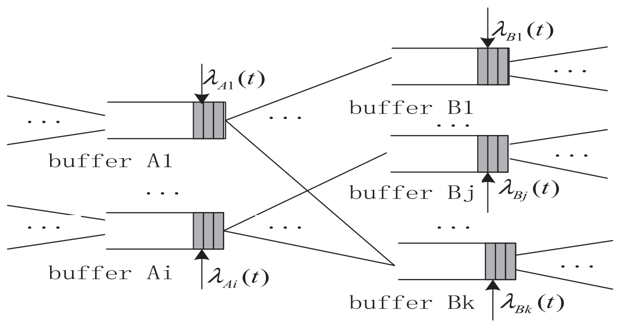

Fig. 2. Model of datacenter networks.

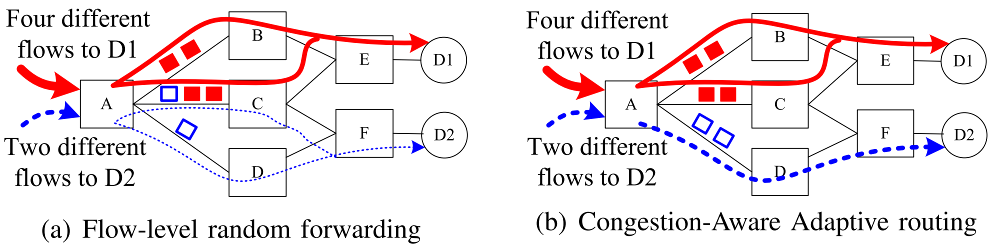
Four different flows to D1
Two different flows to D2
A
B
C
D
E
F
D1
D2
(a) Flow-level random forwarding
(b) Congestion-Aware Adaptive routing

Fig. 1. Motivation of designing adaptive routing for datacenters.

Table 1
The variables most-used in this paper.

Var	Definition
$Q_j(t)$	The queue length of buffer j at time t
B	Buffer size
$\mathcal{S}_j$	The set of links which can server buffer j
$\lambda_j(t)$	The external arrival rates at which the traffic comes to buffer j
$\mathcal{R}_j$	The set of neighbors of buffer j
$\mathcal{R}_j^d$	The set of next hops of packets destined for d at buffer j
μ_j^i	The fraction of time that link i spends serving buffer j
$A_j(n)$	The amount of external traffic to buffer j during time slot n
$W_{jk}^i(n)$	The amount of traffic transferred from buffer j to buffer k
a_j	The long-run external arrival rate to buffer j
M_1	The congestion threshold of the queue length difference between two adjacent buffers
M_2	The congestion threshold of the queue length of a buffer.

from buffer j to k. Denote $W_{jk}^i(n)$ as the amount of traffic transferred from buffer j to buffer k by link i during time slot n, and $A_j(n)$ as the amount of external coming traffic to buffer j during time slot n. Then the dynamic of queue length in buffer j is as follows:

$$Q_j((n+1)T) = Q_j(nT) + A_j(n) + \sum_{l:j\in\mathcal{R}_l}\sum_{i\in\mathcal{S}_j} W_{lj}^i(n) - \sum_{k\in\mathcal{R}_j}\sum_{i\in\mathcal{S}_j} W_{jk}^i(n), \quad j \in \mathcal{B} \qquad (1)$$

The definition of variables are summarized in Table 1 for easy reference.

4. Adaptive forwarding

In this section, the proposed CAAR scheme will be described in detail. We assume that multipaths are pre-configured in datacenters. Each flow selects the next hop with the minimum queue length.

4.1. Forwarding packets

Under CAAR, each switch makes forwarding decisions based on its neighbors' queue length, whose value is updated in each time slot. Consider a packet p destined for d during time slot n. If its flow identifier is new, then it will be transmitted to the buffer $\bar{k}^d(nT)$ with the minimum queue length in $\mathcal{R}_j^d$.

$$\bar{k}^d(nT) = \arg\min_{k\in\mathcal{R}_j^d} Q_k(nT) \qquad (2)$$

This new flow will be recorded in the flow table in buffer j.

If packet p belongs to an old flow, then usually it will be sent to the preassigned next hop to prevent packets from reordering. However, when any one of the two following situations happen,

- The queue length of buffer $j, Q_j(t)$, exceeds a threshold M_2, namely, congestion is going to occur in buffer j.
- The difference of queue length between buffer j and the preassigned buffer h is not larger than $-M_1$, which implies that the packets from buffer j to buffer h will possibly overwhelm h.

To keep the network stable, all packets in buffer j will be sent to the next hop $\bar{k}^d(t)$ which has the minimum queue length.

With regard to the flow identification, the end system can use a hashed value of the five tuples of a flow to generate the flow ID.

4.2. Local caching rule

As stated above, for a packet destined for d with a preassigned next hop h in buffer j, if $Q_j(t) - Q_h(t) \leqslant -M_1$, packets will be forwarded to $\bar{k}^d$ to avoid packets dropping in buffer h. However, if even $Q_j(t) - Q_{\bar{k}^d(t)}(t) \leqslant -M_1$, the packets from buffer j will likely overwhelm buffer $\bar{k}^d$ with the lightest load. In this situation, to alleviate potential congestion in buffer $\bar{k}^d$ and avoid packet loss, the best choice is that buffer j temporarily caches packets, which is referred to as Local Caching Rule.

4.3. Signaling

A buffer needs to know the queue length of its possible next hops to make forwarding decisions. How to get the information depends on the queue mechanism of switches. The queue mechanisms of current switches mainly include three classes: input queuing, output queuing and shared buffering [26]. If switches use output queuing, which is widely employed since it can easily support weighted fair queuing (WFQ)-based packet scheduling [27]. Then the switch can get the queue length in real time without consuming additional bandwidth. Otherwise, a signaling message is needed to update the queue length information of neighbors. Since the proposed CAAR scheme operates in slotted time, the queue length information will be updated every time slot T. But if the relative difference between current queue length and the length in last slot is not larger than a threshold $\Delta(0 < \Delta < 1)$, the queue length will not be updated to reduce overhead. At worst, if every time slot T, a switch sends its queue length to all of its neighbors, then each link needs to transmit two signaling packets every time slot T. Since the signaling packet do not need to deliver any payload, we could multiplex some packet header field to carry the queue length information. The size of the ethernet frame has to be larger than 64 KB. Thus, the signaling overhead is at most $\frac{64\ \text{KB}\times 2}{T\times C}$. The link capacity between a ToR (Top of Rack) switch and a server is usually 1 Gbps. But the link capacity between switches is generally 10 Gbps or even higher. Thus, if the signaling period is 500 us, then the overhead is at most $\frac{64\ \text{KB}\times 8\times 2}{500\ \text{us}\times 10\ \text{Gbs}} = 0.02\%$, which is acceptable.

4.4. Procedure of CAAR

The basic components included in CAAR have been introduced as above. To present a coherent procedure of CAAR, its pseudo-codes are illustrated in Figs. 3 and 4, respectively. In summary, CAAR includes packet forwarding module and message updating module. Fig. 3 presents packet forwarding procedure, line 2 is the local caching rule. Lines 4 and 5 aim to avoid potential packets loss when $Q_j(nT) > M_2$. Lines 8–15 are for old flows while lines 16–18 are for new flows. Fig. 4 depicts message updating, which provides two functions: UpdateTable() and UpdateLength(). In UpdateTable(), Lines 1–4 aim to update the neighbor ID with the minimum queue length. Lines 5–9 judge whether buffer j is about to be congested. And Lines 10–12 delete the out-dated routing items. In UpdateLength(), if the relative difference between current queue length and that in last slot exceeds a threshold Δ, then the buffer will send update messages to its neighbors.

4.5. CAAR Without reordering

In CAAR, a flow will change its path if the current path becomes congested, which possibly lead to out-of-order packets. The traffic measurement results show that the traffic in datacenters is mixed of a much small proportion of elephant flow and a large proportion of mice flows [28,19]. It is not quite necessary to redirect the mice

Packet forwarding

```
void packetForward(PACKET p)
1  Get the destination $d$ of $p$.
2  if $Q_j(nT) - Q_{\bar{k}^d}(nT) \leq -M_1$ then sleep($T$);
3  else
4     if $flag$[j]=1 then // node $j$ is congested
5        Forwarding packet p to $\bar{k}^d$;
6     else
7        Check the flow ID of packet $p$;
8        if the flow exists in the flow table then
9           Get next hop $h$;
10          if $Q_j(nT) - Q_h(nT) > -M_1$ then
11             Forwarding packet $p$ to $h$;
12          else
13             Forwarding packet $p$ to $\bar{k}^d$;
14             Update the routing table;
15          end if
16       else
17          Forwarding packet $p$ to $\bar{k}^d$;
18          Record this new flow in flow table;
19       end if
20    end if
21 end if
```

Fig. 3. Packet forwarding algorithm at buffer j in CAAR.

Message update

```
/*Update the routing table.*/
/*Call at the beginning of each time slot $n$ (n=0,1,...).*/
void UpdateTable(TABLE ft)
1  Compute $\hat{k}^d = \arg\max_{k \in \mathcal{R}_j^d} Q_k(nT)$;
2  if $\hat{k}^d \neq \bar{k}^d$ then
3     $\bar{k}^d \leftarrow \hat{k}^d$.
4  end if
5  if $Q_j(nT) > M_2$ then
6     $flag[j] \leftarrow 1$;
7  else
8     $flag[j] \leftarrow 0$;
9  end if
10 for all the items in the flow table at buffer $j$
11    Delete the outdated items;
12 end for

/*Update queue length information. */
void UpdateLength() /*$B$ is the buffer size.*/
1  if ($\frac{|Q_j(nT) - Q_j((n-1)T)|}{B} > \Delta$) then
2     Send $Q_j(nT)$ to all neighbors of $j$;
3  end if
```

Fig. 4. Message update algorithm at buffer j in CAAR.

flows since they can be finished quickly. For example, queries and their responses take only 1.6 KB to 2 KB data [7]. They can be finished in one or two RTTs. If two long flows select the same path, then congestion will happen [11]. However, only one percent of flows belong to long flows in datacenters [13]. Therefore, the possibility that two long flows select the same path is quite small. Based on this investigation, we propose CAAR without reordering mechanism and evaluate its performance in the evaluation section.

4.6. An illustrative example

In this subsection, we use a simple example shown in Fig. 5 to intuitively illustrates how CAAR works. Server 1 denotes the source and server 2 is the destination. The lines with arrow represent the pre-calculated multipaths which only include the paths with the same number of hops. The hollow packet belongs to a recorded flow, whose identifier is 1 and the solid packet belongs to a new flow. The preassigned next hop for packet of flow 1 has been recorded in the routing table in each buffer. Let $M_1 = 3$ and $M_2 = 8$. Now server 1 generates a hollow packet and a solid packet respectively. Considering the hollow packet first, buffer a_1 will select the next hop a_2 for the hollow packet according to the routing table since $Q_{a_1}(t)$ is not bigger than M_2 and $Q_{a_1}(t) - Q_{a_2}(t) = 4 - 5 = -1 > -3$. However, at buffer a_2, since the queue length of the preassigned next buffer 1 is too large such that $Q_{a_2}(t) - Q_{b_1}(t) = 5 - 8 = -3, a_2$ will select buffer $\bar{k}^d = b_2$ as the next hop and change the corresponding routing items, then the next hollow packet will select buffer b_2 as its next hop. While in buffer b_2, the only next hop is c_1, however the queue length of buffer c_1 is too large such that $Q_{b_2}(t) - Q_{c_1}(t) = 2 - 7 = -5 < -3$. Hence, buffer b_2 will wait for a time slot according to the local caching rule. And at the next time slot, if the queue length of buffer c_1 is still very large such that $Q_2(t) - Q_{c_1}(t) \leqslant -3$, then it waits for another time slot again until the queue length in buffer c_1 becomes smaller such that $Q_2(t) - Q_{c_1}(t) > -3$. On the other hand, the solid packet belonging to a new flow will be forwarded to the buffer with the minimum queue length in $\mathcal{R}_j^d$.

4.7. A word on complexity

Our CAAR does not need to compute the traffic distribution ratio of each link as some centralized routing algorithms do, it just selects the next hop with the minimum queue length from the next hop options in real time. Thus, its time complexity is rather low. In Fig. 4, the time complexity of Line 1 is $O(|\mathcal{R}_j|)$, $|\mathcal{R}_j|$ is the cardinality of set $\mathcal{R}_j$. Lines 2–9 take $O(1)$. And lines 10–12 take $|T_j|$, where T_j represents the flow table at buffer j and $|T_j|$ is the number of items in table T_j. Hence, the time complexity of the module UpdateTable(Table ft) takes $O(\max\{|\mathcal{R}_j|, |T_j|\})$. And UpdateLength() takes $O(1)$ runtime, but need $O(\frac{|\mathcal{R}_j| \times S_p}{T})$ bandwidth, where S_p is the updating message length. With respect to the packet forwarding module in Fig. 3, the time complexity depends on the algorithm of looking up routing table. At worst it is $O(|T_j|)$.

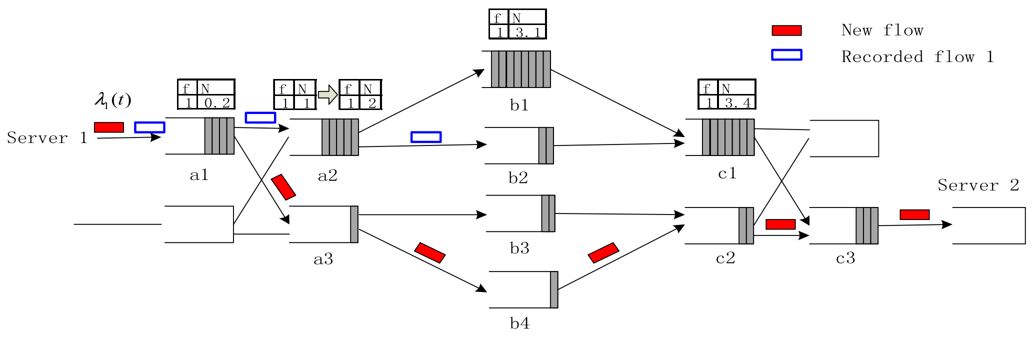

Fig. 5. The communication procedure between server 1 and server 2 under the proposed scheme CAAR.

5. Theoretical analysis

Stability is an important feature desired by the decentralized CAAR scheme. In this section, we will first introduce some definitions and present some constraint conditions, and then theoretically prove that the proposed CAAR is stable.

5.1. Definition and constraint

Definition 1. The network is *stable* under a policy $\mathcal{P}$ if the sum of the number of backlogged packets in the network has an upper bound Q_{max}, that is

$$\limsup_{t\to+\infty}\sum_{j=1}^{B}Q_j(t)\leqslant Q_{max} \tag{3}$$

where Q_{max} may only depend on the service rate r, the arrival rates $\vec{a}=(a_j)_{j\in\mathcal{B}}$, and the burst parameter $\vec{b}=(b_j)_{j\in\mathcal{B}}$, but not on the initial condition.

According to this stability criterion, if a datacenter is stable under the control of our CAAR, the queue length in all switch buffers is bounded, which is beneficial to avoiding packet losses.

Definition 2. All the external arrival matrices $\vec{\lambda}(t)=(\lambda_j(t))_{j\in\mathcal{B}}$ under which the network can be stable, compose the *network throughput region* Λ.

Similar definition of throughput region can be seen in [29]. The throughput region can be formalized as the set of the external arrival rates which satisfy the following Constraint 1.

Constraint 1: Let $f_{jk}(j,k\in\mathcal{B})$ be the long-run average rate with which the traffic in buffer j is transferred to buffer k. We say $\lambda(\vec{t})\in\Lambda$ if there exists f_{jk} such that the following conditions hold:

$$f_{jk}\geqslant 0, f_{jj}=0,\quad \text{for all } j,k\in\mathcal{B} \tag{4}$$

$$a_j+\sum_{l:j\in\mathcal{R}_l}f_{lj}=\sum_{k\in\mathcal{R}_j}f_{jk},\quad \text{for all } j\in\mathcal{B} \tag{5}$$

$$\sum_{k\in\mathcal{R}_j}f_{jk}\leqslant\sum_{i\in\mathcal{S}_j}\mu_j^i r,\quad \text{for all } j\in\mathcal{B},\ i\in\mathcal{N} \tag{6}$$

The conditions above are explained as follows:

- Eq. (4) is the flow efficiency constraint. The amount of traffic should not be negative and a buffer will not send packets to itself.
- Eq. (5) is the flow conservation constraint, which states that the sum of the external arrival traffic and the incoming traffic to buffer j from other buffers equals the outgoing traffic from buffer j.
- Eq. (6) is the capacity constraint. μ_j^i represents the time fraction link i spends for buffer j, then $r\mu_j^i$ denotes the traffic transmitted by link i from buffer j. Obviously, the outgoing traffic from buffer j should not be larger than the service all the links $i\in\mathcal{S}_j$ provide.

Obviously, if given the external traffic $\lambda(\vec{t})$, the network can be stabilized, then the above three conditions: flow efficiency, flow conservation and capacity constraint must satisfy, that is, there must exist f_{jk} such that the above conditions hold.

Constraint 2: We consider a fluid model for the external arrivals. During the interval $[t_1,t_2]$, the sum of the traffic coming into buffer j is $\int_{t_1}^{t_2}\lambda_j(t)dt$. Assume there are nonnegative numbers $a_j, b_j, j\in\mathcal{B}$, such that for all $0\leqslant t_1\leqslant t_2$, we have

$$\int_{t_1}^{t_2}\lambda_j(t)dt\leqslant a_j(t_2-t_1)+b_j \tag{7}$$

Actually this is a burst constraint for input traffic. Besides, assume that the traffic enters into buffer j with a long-run average rate a_j.

$$\lim_{t\to+\infty}\frac{1}{t}\int_0^t\lambda_j(t')dt'=a_j, j\in\mathcal{B} \tag{8}$$

Note that as long as the instantaneous rate of the external arrival is finite, then it will satisfy the burst constraint. The measurement results presented in [19] indicate that the traffic in datacenters exhibits on–off pattern and there is many short-lived bursts, but the instantaneous rate will not be infinite. Hence, the burst constraint of the fluid model is reasonable.

5.2. Stability of CAAR

Before proving our CAAR mechanism can guarantee that the network is stable given any $\vec{\lambda}(t)$ that satisfies Constraint 1, we first introduce 3 lemmas.

Lemma 1. *For nonnegative T, if the system satisfies that when* $\sum_{j=1}^{B}Q_j^2(nT)>Q_m$ *(Q_m is a constant value),*

$$\sum_{j=1}^{B}Q_j^2((n+1)T)-\sum_{j=1}^{B}Q_j^2(nT)<-\epsilon \tag{9}$$

Then the system is stable.

Proof. See Appendix A. □

Lemma 2. *There is a constant* $c>0$ *which depends only on the topology of the network so that*

$$\max_{j\in\mathcal{B},k\in\mathcal{R}_j^d}\{Q_j(t)-Q_k(t)\}\geqslant c\sqrt{\sum_{j=1}^{S}Q_j^2(t)} \tag{10}$$

Proof. See Appendix B. □

Lemma 3. *For buffer j which satisfies* $Q_j(nT)>M_2$ *and* $\max_{k\in\mathcal{R}_j^d}\{Q_j(nT)-Q_k(nT)\}>2MT-M_1$, *where M is the maximum rate with which any queue may vary. Let* $I=\{(j,\bar{k}^d)|d$ is the destination of sent packets during time slot n at buffer $j\}$. *We have for buffer j*

$$\sum_{k\in\mathcal{R}_j}\sum_{i\in I}W_{jk}^i(n)\{Q_j(nT)-Q_k(nT)\}\geqslant rcT\sqrt{\sum_{j=1}^{S}Q_j(t)^2}$$

Proof. See Appendix C. □

We compute the equation $\sum_{j=1}^{S}Q_j^2((n+1)T)-\sum_{j=1}^{S}Q_j^2(nT)$ according to the queue dynamics in Eq. (1) and the proposed algorithm AR, obtaining that $\exists Q_m$, when $Q>Q_m, \sum_{j=1}^{S}Q_j^2((n+1)T)-\sum_{j=1}^{S}Q_j^2(nT)<-\epsilon$, that is, the data center network controlled by CAAR is stable.

Theorem 1. *If the arrival rates are within the throughput region, then the system is stable under AR.*

Proof. See Appendix D. □

If a network can keep stable under any possible injected traffic rates that are with in the throughput region Λ, then we can infer that the network is throughput-optimal.

Note that in our theoretical proof, we assume that a switch can obtain accurate queue length information. However, in practice, there maybe some queue length deviation between the measured queue length value and the real value. Intuitively, if the arrival traffic still obeys the traffic model described in Constraint 2, the proof result is still valid. Of course, to prove the performance of CAAR with deviated queue length information, more attempts and strict proof are required to be made. This is also a possible future direction of CAAR.

6. Evaluation

In this section, the performance of CAAR and CAAR without reordering will be compared with static routing and ECMP forwarding mechanisms.

6.1. Simulation setup

We implement the protocols on the NS2 platform. We extend the Class *Node* in NS2, including Routing module, Classifier, Queue and so on, to a *layer 2 switch*, and properly implement CAAR with or without reordering, oblivious random algorithm ECMP and static routing algorithms. The transport protocol in all servers is configured as TCP SACK1. On one hand, the rate of incoming traffic is adjusted by the in-built flow regulation mechanism in TCP to approach to the network throughput region. On the other hand, a unbiased comparison can be conducted since TCP is also employed in the experiments of evaluating the random forwarding performance in [14].

Next, we will describe the topology, communication pattern, traffic generation and parameter setting in our simulations in detail.

(1) *Topology:* VL2 architecture is an example of a folded Clos network as shown in Fig. 6. This fabric is configured in simulations to eliminate the possible negative impact of other topology on the performance of oblivious random forwarding. There is a capacity gap between the server line cards and intermediate network links in VL2. In the simulation, the rate of server line cards is 1 Gbps and the rate of intermediate network links is 10 Gbps. Each ToR connects to 20 servers and 2 Aggregation switches, and each Aggregation switch connects to all the Intermediate switches. Multipaths that include all the paths for each pair of servers are preconfigured. CAAR only selects one of them to forward packets.

We will first compare the performance of CAAR with the other two protocols with VL2 in detail. In addition, to prove that the performance of our CAAR is independent on topology, we also conduct simulations with Tree and Fat-tree topologies in Section 6.2.2.

(2) *Communication pattern:* The simulation experiments are conducted under three types of communication patterns:

- *Stride(x) pattern*: Some servers i transmit packets to the servers $x + i$.
- *Hot spot pattern:* Some servers transmit traffic to the same destination.
- *Random pattern:* A server generates some flows and sends each flow to a random destination.

(3) *Traffic generation:* To make the simulation results more convinced, except from long flows, we also generate flows with random size based on the measurement results in real datacenters in [14]. The sizes of 99% of flows are smaller than 100 MB. We discretize the flow size distribution as shown in Fig. 7. In our simulations, a uniformly random value $f \in U[0,1]$ is generated, according to the value of f, we can get the corresponding flow size. For example, if $0.6 < f \leqslant 0.8$, a flow with flow size 10 KB will be generated.

With respect to the number of flows, to keep fair when comparing the goodput and loss ratio under different protocols, we generated the same number of flows for all the algorithms. If one flow ends, another flow will be generated.

(4) *Parameter setting:* In default, the switch buffer size B is 50 packets. The key parameters in our algorithm, M_1, M_2, is set to 0.5 B and 0.9 B. The parameters of our algorithm in simulations are summarized in Table 2.

6.2. Results

(1) *Small scale topology:* First, the proposed CAAR algorithm is validated in a small topology with $N_I = 5, N_A = 10, N_T = 10$.

Stride(100) communication pattern: Let servers $i \in [0, 39]$ send packets to servers $100 + i$. To be fair to all the four mechanisms, we first let each sender generate a long flow. In this way, the external traffic is absolutely equal for each mechanism. Furthermore, to avoid all the flows starting at the same time, which is a impractical

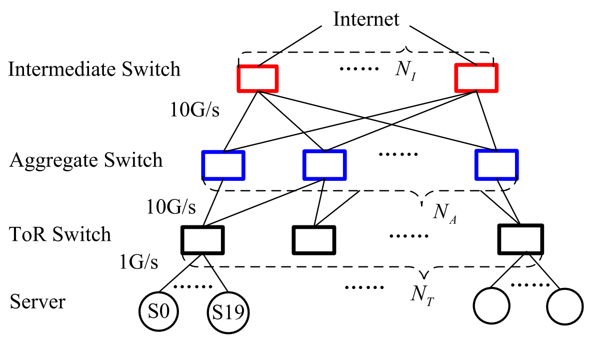

Fig. 6. VL2 topology with N_I Intermediate Switches, N_A Aggregate Switches and N_T ToR Switches.

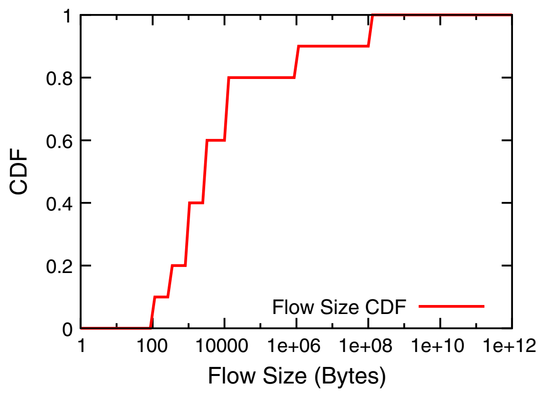

Fig. 7. The distribution of flow size in our simulation.

Table 2
Parameters setting.

Parameter	Value	Parameter	Value
Pkt Size	1.4 KB	B	50 packets
M_1	0.5 B	M_2	0.9 B
Δ	0.05 B	T	500 us

case, we let each flow orderly starts with interval 1 s. Hence, the last flows should be active at 39 s. Fig. 8 shows the comparison of the goodput under the four protocols. All the goodput values keep increasing before 45 s and become stable after that. The proposed CAAR algorithm with reordering performs approximately 9% better than CAAR without reordering. The goodput of CAAR is 20% larger than ECMP and about 33% larger than Static routing. Note that the total goodput for 40 flows in our CAAR is approximately 36 Gbps, which indicates CAAR can efficiently utilize the bandwidth.

To further compare the performance under more practical traffic patterns. We randomize each flow size according to Fig. 7. When a flow ends, a new flow with random size becomes active. Hence, the external traffic will be nearly identical for all the four routing protocols. Note that all the next simulations generate flows with random flow size. Fig. 9 shows the aggregate goodput results. We can see that CAAR with reordering still performs better than CAAR without reordering, ECMP and static routing protocols. But the goodput fluctuates. The reason is that the flow size is random. When a flow finishes, a new flow will be generated and begin with TCP slow start.

Hot spot communication pattern: Servers 0–80 sends a flow with random size to server 81. Figs. 10 and 11 shows the goodput and loss ratio results, respectively. The goodput of the four protocols is about 0.8 Gbps since the capacity of the bottleneck link, which connects to server 81, is 1 Gbps. ECMP and static routing have relatively higher goodput than CAAR. This is because CAAR will temporarily cache packets to avoid packet loss when congestion happens, which possibly wastes some bandwidth. While static and ECMP routing inject packets into the bottleneck link ceaselessly to fill the bottleneck link. Besides, in hot spot communication pattern, the single bottleneck link connecting to the receiver limits the aggregated throughput. The advantage of CAAR, better balancing traffic across multiple links, cannot be exhibited in this communication pattern. But with respect to loss ratio, the static and random forwarding both have about 7% loss ratio as shown in Fig. 11. While CAAR has no packet dropping, which is quite important for applications that are sensitive to packet loss.

Random communication pattern: Fig. 12 illustrates the goodput in random communication pattern. The number of flows is 50. Since only one server generates traffic, the NIC of server becomes the bottleneck. We can observe that the goodput under the four protocols is about 0.93 Gbps, close to.

1 Gbps. And there is small fluctuation since the flow size is random. Under this scenario, the sender link is the bottleneck and the goodput is quite relevant to the upper layer protocol.

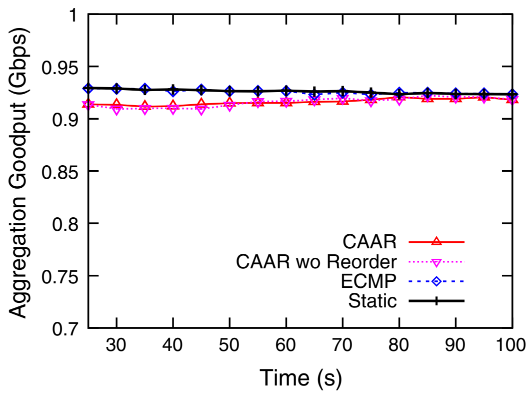

Fig. 10. Goodput under Hot spot communication pattern.

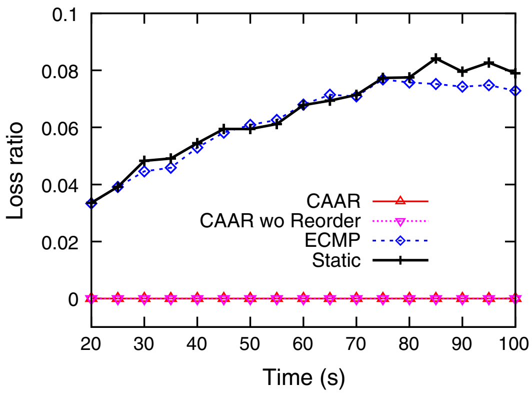

Fig. 11. Loss ratio under Hot spot communication pattern.

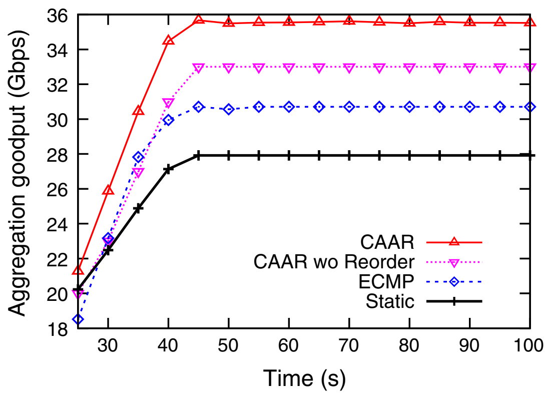

Fig. 8. Goodput under Stride communication pattern with fixed flow size.

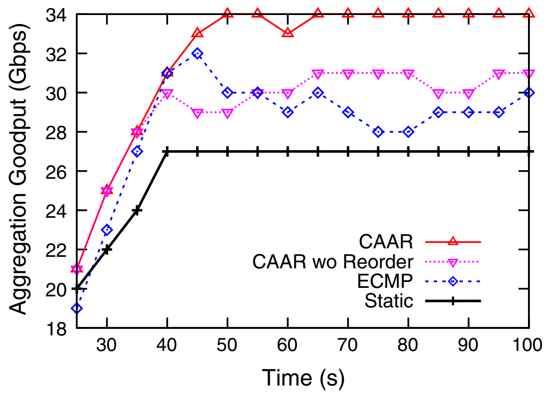

Fig. 9. Goodput under Stride communication pattern with random flow size.

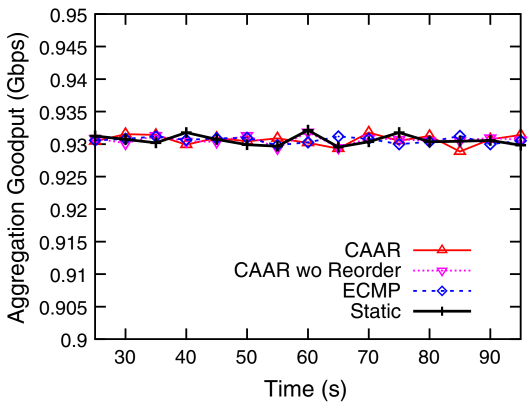

Fig. 12. Goodput under random communication pattern.

(2) *Large scale topology:* To validate the scalability of the proposed CAAR, we conduct the stride scenario under the topology with $N_I = 25, N_A = 40, N_T = 100$, which can support 2000 servers and the topology with $N_I = 10, N_A = 100, N_T = 500$ that can support 10,000 servers. The goodput results are shown in Figs. 13 and 14. Similar to the results in small topology, the advantage of CAAR is still obvious, which indicates that our CAAR has good scalability.

6.2.1. Impact of parameters

To assess the impact of M_1 and M_2 on the performance of CAAR, ten pairs of them are selected as shown in Table 3. We refer to them as CAAR1–10. Fig. 15 shows the goodput results of them under the stride scenario with random flow size in small topology. CAAR1 performs worst and CAAR2-7 perform better than CAAR8-10. Hence, the goodput will not benefit from too small M_2 (Small M_2 indicates that the packets will be forwarded to the buffer with the minimum queue length even if the current buffer is not congested). Possibly the reason is that small M_2 will cause a large amount of reordering packets which reduces the TCP performance. In contrast to CAAR0, larger M_1 in CAAR8-10 will reduce the chance that the recorded flows select new better paths, hence the goodput is lower than CAAR2-7. Therefore, the parameters in CAAR5-7 are good choice for implementation of CAAR.

Fig. 16 shows the reordering packets under CAAR1-10 in the stride scenario with random flow size per second for a flow. Obviously the number of the normalized reordering packets is very small, which will introduce little burden to the end system.

Figs. 17 and 18 plot the aggregate goodput and the number of reordering packets per switch per second with different switch port buffer sizes, respectively. We can see that as the buffer size increases, the aggregate goodput of CAAR keeps stable. However, the number of reordering packets decreases dramatically.

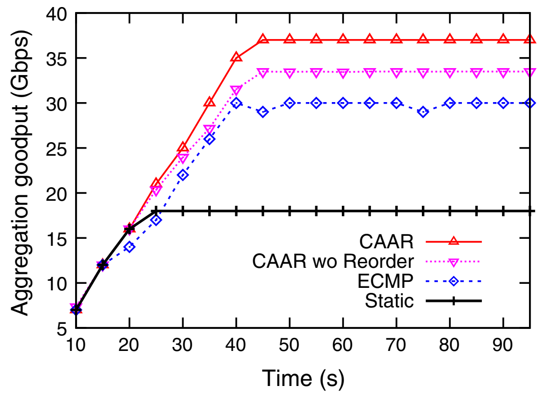

Fig. 13. Goodput under random communication pattern in datacenter that can support 2000 servers.

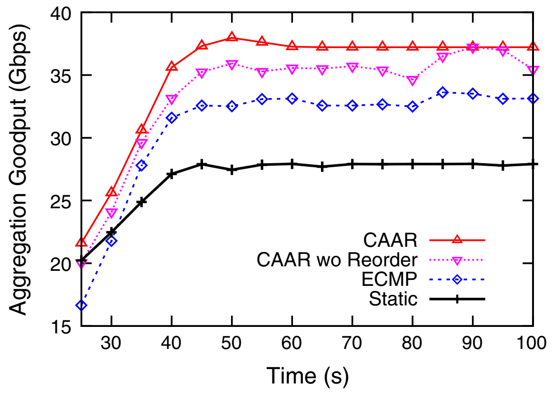

Fig. 14. Goodput under stride communication pattern in large datacenters that can support 10 K servers.

Table 3
Parameter pairs.

CAAR#	1	2	3	4	5	6	7	8	9	10
$\frac{M_1}{B}$	0.5	0.5	0.5	0.5	0.5	0.45	0.55	0.6	0.65	0.7
$\frac{M_2}{B}$	0.5	0.6	0.7	0.8	0.9	0.9	0.9	0.9	0.9	0.9

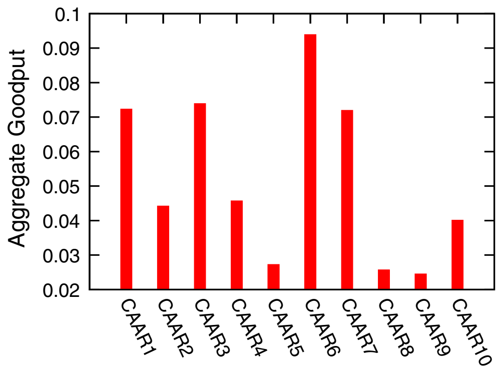

Fig. 15. Goodput with different parameters.

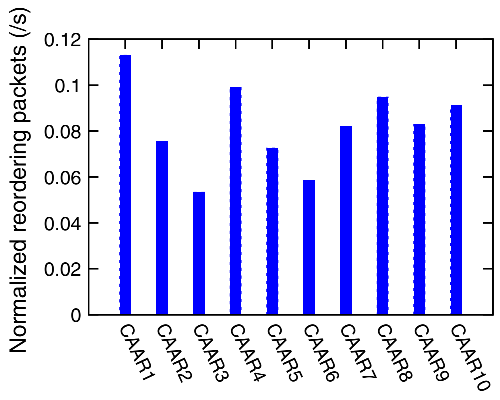

Fig. 16. The reordering packets of one flow every second.

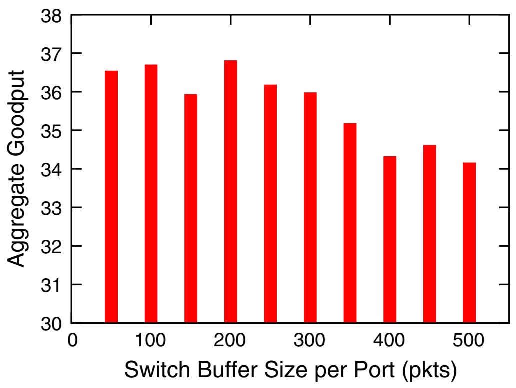

Fig. 17. The aggregated goodput with different switch buffer size.

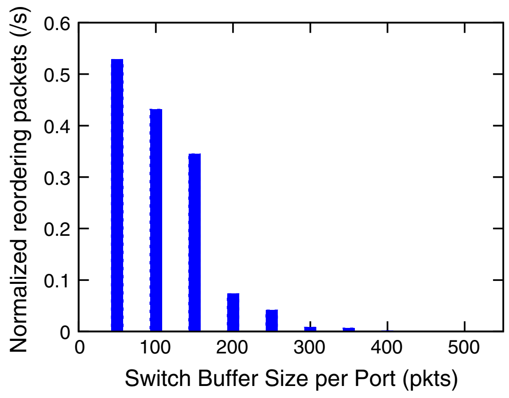

Fig. 18. The reordering packets of one flow every second with different switch buffer size.

Especially when the buffer size is equal or larger than 400 packets, the number of reordering packets equals zero. This is because larger switch buffer can accommodate more packets. The threshold of changing the path of a flow increases. Thus, the event of packets reordering rarely happens.

6.2.2. Other topologies

To evaluate whether our protocol CAAR works well in other topologies besides from VL2, we conduct simulations in two other kinds of datacenter topologies. One is multi-rooted tree shown in Fig. 19, which is widely used in industry. The other one is Fat-tree (Fig. 20), another new architecture proposed for datacenters except from VL2. In our simulations, the Tree topology has 3 roots (intermediate switches), 25 aggregation switches and 50 ToRs. There is only one path except from the paths from the aggregation switches to the intermediate switches. The link capacity of ToR switches is 1 Gbps. All the aggregation and intermediate switches have links with 10 Gbps rate.

Fig. 21 shows the goodput results with Tree topology. Servers (0–39) begin to communicate with the other (100–139) servers at the start of the simulation. We can see that under the four protocols, the goodput results are almost the same, approximate 18.6 Gbps. This is because each ToR connects with 20 servers and each aggregation switch connects with 2 ToRs. Thus, servers (0–39) will go through the same aggregation switch. The link between the two ToRs and their connected aggregation switch becomes the bottleneck. Since each link capacity of the aggregation switch is 10 Gbps, the maximum bandwidth is 20 Gbps. Thus, the aggregate goodput is about 18.6 Gbps. In static routing, we use the dst%3 to select the intermediate switch, thus it also achieves almost the same aggregate bandwidth as the other protocols.

Fig. 22 plots the goodput in a Fat-tree topology. In our simulations, we let $k = 16$, that is, the number of ports of each switch is 16. Half of the 16 ports connect switches in the down layer and the other half connect with the switches in the up layer. For example, each ToR switch connects 8 servers and 8 aggregation switches. Since Fat-tree aims to achieve high bisection bandwidth by employing commodity switches. The link bandwidth of all the switches is 1 Gbps. We let all the servers in the first pod, server (0–63), communicate with all the servers in the third pod from 0 s. The static routing in our simulations is implemented as follows: If there are multiple choices, then the switch sends out the packet with destination i from the $(i\%\frac{k}{2})$-th port. The goodput results in

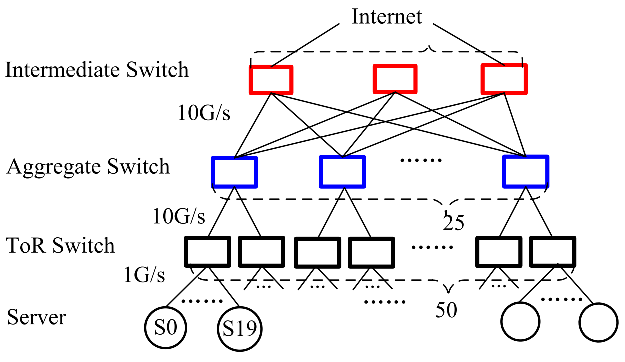

Fig. 19. Tree topology with multiple roots.

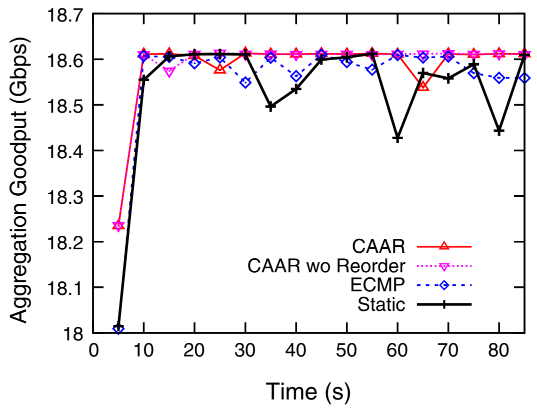

Fig. 21. The aggregated goodput with Tree topology.

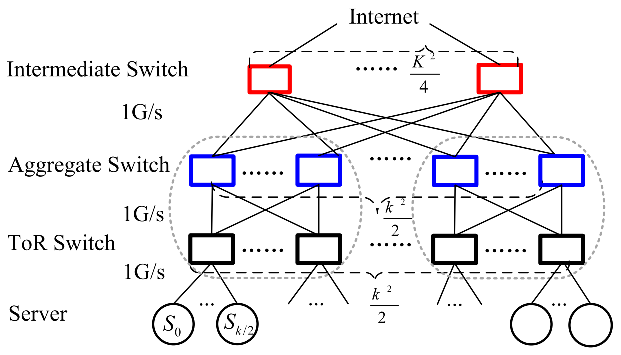

Fig. 20. Fat-tree topology.

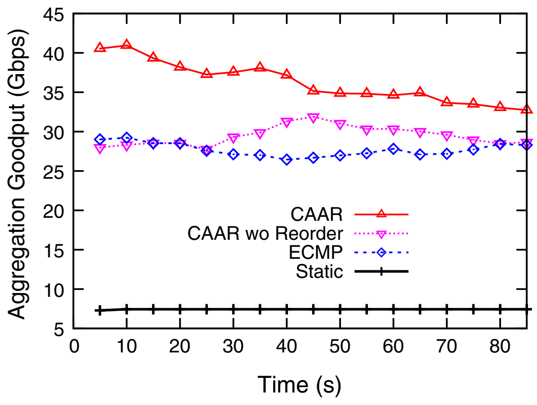

Fig. 22. The aggregated goodput with Fat-tree topology.

Fig. 22 show that CAAR with reordering achieves the maximum goodput. ECMP achieves a little lower goodput compared with CAAR with reordering. And static routing only achieves about 7.7 Gbps. This is because this protocol transmits packets only along 8 paths.

The above results show that CAAR performs well in different kinds of topologies.

(5) *Benchmark Traffic:* To evaluate the performance of CAAR with realistic data center traffic, we generate data center traffic based on the traffic measurement in [7]. The traffic lasts for 10 min. Totally 12,871 flows are generated, which are consisted of query flows with 2000 Bytes length and background flows with 1 K–100 MB length. End hosts employ TCP NewReno protocol.

Figs. 23 and 24 shows the average, 95th, 99th and 99.9th percentile of the flow competition time for query and background flows, respectively. We can see that CAAR exhibits dramatic benefits compared with ECMP and Static mechanisms. This is because in ECMP and Static mechanisms, long flows are possibly transmitted along the same path and thus cause congestion. This not only leads to long flow completion time of the corresponding long flows, but also negatively impact the flow completion time of query flows due to long queuing delay and packets drops and so on. While CAAR can effectively avoids the congestion as long as there are underutilized paths. Therefore, CAAR dramatically reduces the flow completion time of query and background flows.

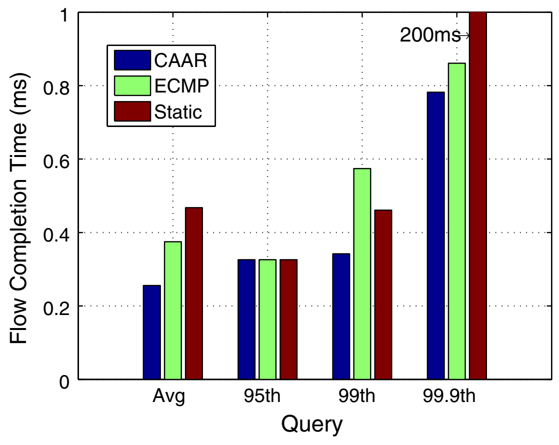

Fig. 23. Flow completion time of queries with the benchmark workload.

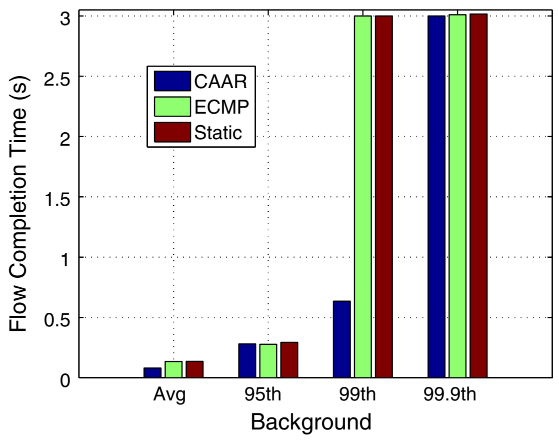

Fig. 24. Flow completion time of background flows with the benchmark workload.

7. Conclusions

In this paper, we propose a congestion-aware adaptive routing scheme, CAAR, to balance traffic across multiple paths in datacenters. CAAR can responsively adapt to bursts and rapidly varying traffic patterns in datacenters, avoid reordering packets when no congestion happens, and keep stable. It is decentralized and uses only local information. It can work well in various communication patterns. The basic idea of CAAR is to send packets to the next hop with the lightest load. We define the concept of throughput region and prove that CAAR can keep the network stable whenever the arrival rates are in the throughput region. The performance of CAAR is evaluated on the ns-2 platform. The simulation results validate that the proposed CAAR algorithm can more efficiently utilize the redundant bandwidth compared with static and ECMP protocols.

Acknowledgement

The authors gratefully acknowledge the anonymous reviewers for their constructive comments. This work is supported in part by National Basic Research Program of China (973 Program) under Grant No. 2012CB315803, and National Natural Science Foundation of China (NSFC) under Grant No. 61225011.

Appendix A. Proof of Lemma 1

Proof. First, let η be the smallest integer such that $\sum_{j=1}^{B} Q_j^2(\eta T) \leqslant Q_m \eta$ must exist. Because if $\sum_{j=1}^{B} Q_j^2(0) \leqslant Q_m$, then let $\eta = 0$. Else if $\sum_{j=1}^{B} Q_j^2(0) > Q_m$, then according to the condition of this lemma, $\sum_{j=1}^{B} Q_j^2((0+1)T) < \sum_{j=1}^{B} Q_j^2(0) - \epsilon$. Iteratively, there must exist an integer η such that inequality $\sum_{j=1}^{B} Q_j^2(\eta T) \leqslant Q_m$ satisfies.

From the queue dynamics and burst constraints, we have

$$\sum_{j=1}^{B} Q_j((n+1)T) \leqslant \sum_{j=1}^{B} Q_j(nT) + a_j T + b_j + BrT \tag{11}$$

Hence

$$\sum_{j=1}^{B} Q_j^2((n+1)T) \leqslant \sum_{j=1}^{B} Q_j^2(nT) + 2B\sqrt{\sum_{j=1}^{B} Q_j^2(nT)}(a_j T + b_j + BrT) + (a_j T + b_j + BrT)^2 \tag{12}$$

Define $\overline{Q} = Q_m + 2(a_j T + b_j + rBT)B\sqrt{Q_m} + (a_j T + b_j + rBT)^2$, we can readily prove that $\sum_{j=1}^{B} Q_j^2(mT) \leqslant \overline{Q}$whenm $\geqslant \eta$.

If $\sum_{j=1}^{B} Q_j^2(mT) \leqslant Q_m$, then according to Eq. (12), $\sum_{j=1}^{B} Q_j^2((m+1)T) \leqslant \overline{Q}$. Else if $Q_m < \sum_{j=1}^{B} Q_j^2(mT) \leqslant \overline{Q}$, then $\sum_{j=1}^{S} Q_j^2((m+1)T) - \sum_{j=1}^{S} Q_j^2(mT) < -\epsilon$. Therefore $\sum_{j=1}^{B} Q_j^2((m+1)T) \leqslant \overline{Q}$.

Hence, the system is stable if it satisfies that when $Q_j(nT) > Q_m, \sum_{j=1}^{S} Q_j^2((n+1)T) - \sum_{j=1}^{S} Q_j^2(nT) < -\epsilon$. □

Appendix B. Proof of Lemma 2

Proof. Consider the buffer j_0 with the maximum length, obviously $Q_{j_0}(t) \geqslant \sqrt{(\sum_{j=1}^{S} Q_j^2(t))/S}$.

There exists a sequence of queues through which the packet destined for d can be routed to the destination. Define D as the maximum hops on the routing path. For the packet destined for d

at buffer j_0, there exist a sequence $j_1, j_2, \dots, j_n, n < D, j_{k+1} \in \mathcal{R}^d_{j_k}, k = 0, 1, 2, \dots, (n-1)$ such that $\sum_{k=0}^{n-1}(Q_{j_k}(t) - Q_{j_{k+1}}(t)) = Q_{j_0}(t)$. Hence, we can obtain $\max_{k=0,1,\dots,n-1}(Q_{j_k}(t) - Q_{j_{k+1}}(t)) \geqslant \frac{Q_{j_0}(t)}{D} \geqslant \frac{\sqrt{\sum_{j=1}^{S} Q_j^2(t)}}{D\sqrt{S}}$. Let $c = \frac{1}{D\sqrt{S}}$, then we have $\max_{j\in\mathcal{B},k\in\mathcal{R}_j^d}\{Q_j(t) - Q_k(t)\} \geqslant c\sqrt{\sum_{j=1}^{S} Q_j(t)^2}$. □

Appendix C. Proof of Lemma 3

Proof. First, we show that buffer j is served during the whole time slot n. At time $t, Q_j(nT) - Q_{\bar{k}^d(nT)}(nT) > 2MT - M_1$, Because any queue can vary not larger than M. Consider the worst case, the length of buffer j decreases M while buffer $\bar{k}^d(nT)$ increases M per unit time during the time slot n. Then at time $(n+1)T, Q_j((n+1)T) - Q_{\bar{k}^d(nT)}((n+1)T) > -M_1$. Hence buffer j does not satisfy the condition of local caching rule defined in Section 4, namely buffer j will transmit packets during the whole time slot n.

Then according to the proposed CAAR mechanism, since $Q_j(nT) > M_2$, we have $flag[j] = 1$, namely, during the time slot n, the packets destined for d at buffer j will select $\bar{k}^d(nT)$ as the next hop. According to Lemma 2, $\max_{j\in\mathcal{B},k\in\mathcal{R}_j^d}\{Q_j(t) - Q_k(t)\} \geqslant c\sqrt{\sum_{j=1}^{S} Q_j(t)^2}$ and the assumption that the capacity of all the links equals to r. We can obtain

$$\sum_{k\in\mathcal{R}_j}\sum_{i\in I} W^i_{jk}(n)\{Q_j(nT) - Q_k(nT)\} \geqslant rcT\sqrt{\sum_{j=1}^{S} Q_j(t)^2} \tag{13}$$

Lemma 3 is proved. □

Appendix D. Proof of Theorem 1

Proof. Define $\triangle Q \triangleq \sum_{j=1}^{S} Q_j^2(n+1)T - \sum_{j=1}^{S} Q_j^2(nT)$. With simple calculations, we get that

$$\triangle Q = 2\sum_{j=1}^{S}(Q_j((n+1)T) - Q_j(nT))Q_j(nT) + \sum_{j=1}^{S}(Q_j((n+1)T) - Q_j(nT))^2 \tag{14}$$

Because we assume that one buffer can vary with the rate no more than M, we have

$$\sum_{j=1}^{S}(Q_j((n+1)T) - Q_j(nT))^2 \leqslant M^2T^2 \tag{15}$$

According to the queue dynamics in Eq. (1), we get that

$$\begin{aligned}\sum_{j=1}^{S}(Q_j((n+1)T) - Q_j(nT))Q_j(nT) &= \sum_{j=1}^{S} A_j(n)Q_j(nT) + \sum_{j=1}^{S}\sum_{l:j\in\mathcal{R}_l}\sum_{i=1}^{L} W^i_{lj}(n)Q_j(nT) - \sum_{j=1}^{S}\sum_{k\in\mathcal{R}_j}\sum_{i\in\mathcal{S}_j} W^i_{jk}(n)Q_j(nT) \\ &= \sum_{j=1}^{S}\sum_{k\in\mathcal{R}_j}\sum_{i\in\mathcal{S}_j} W^i_{jk}(n)(Q_k(nT) - Q_j(nT)) + \sum_{j=1}^{S} A_j(n)Q_j(nT)\end{aligned} \tag{16}$$

By interchanging the order of the summations, we yield

$$\sum_{j=1}^{S}\sum_{k\in\mathcal{R}_j}\sum_{i\in\mathcal{S}_j} W^i_{jk}(n)(Q_k(nT) - Q_j(nT)) = -\sum_{i=1}^{L}\sum_{j:i\in\mathcal{S}_j}\sum_{k\in\mathcal{R}_j} W^i_{jk}(n)(Q_j(nT) - Q_k(nT)) \tag{17}$$

For any buffer j and k, we have

$$W^i_{jk}(n)(Q_j(nT) - Q_k(nT)) > -(MM_1T + 2M^2T^2) \tag{18}$$

This is because first $0 \leqslant W^i_{jk}(n) \leqslant MT$, and second if $Q_j(nT) - Q_k(nT) \leqslant -2MT - M_1$, then $Q_j(\bar{t}) - Q_k(\bar{t}) \leqslant -M_1$ for all $\bar{t} \in (nT, (n+1)T]$, which indicates $W^i_{jk}(n)$ will be 0 under the proposed scheme CAAR since the packet of buffer j will not be transferred to k during time interval $(nT, (n+1)T]$.

Let $\mathcal{F} = \{i \in I | I$ satisfies the condition of LEMMA 3$\}$. From Eq. (13) (17) and (18), we can obtain

$$\begin{aligned}\sum_{j=1}^{S}\sum_{k\in\mathcal{R}_j}\sum_{i\in\mathcal{S}_j} W^i_{jk}(n)(Q_k(nT) - Q_j(nT)) &\leq -\sum_{i\in\mathcal{F}} Trc\sqrt{Q} + NB^2(MM_1T + 2M^2T^2) \\ &= -\sum_{i\in\mathcal{F},j:i\in\mathcal{S}_j}(1-\mu^i_j)Trc\sqrt{Q} - \sum_{i\in\mathcal{F},j:i\in\mathcal{S}_j}\mu^i_j Trc\sqrt{Q} + NB^2(MM_1T \\ &\quad + 2M^2T^2)\end{aligned} \tag{19}$$

Define $\mu \triangleq \min_{j\in\mathcal{B},i\in\mathcal{S}_j}\{1 - \sum_{j:i\in\mathcal{S}_j}\mu^i_j\}$. Although possibly at some time the external traffic will completely consume the network capacity, however, in the long run, the injected traffic will not always equal to the network capacity. Hence, the utilization ratio of the link will be less than 100%, namely, $\mu^i_j < 1$. Therefore, we have $\mu > 0$. When Q is large enough, there must exist at least a buffer which satisfies the condition of Lemma 3, that is, $|\mathcal{F}| \geqslant 1$. Hence we can get

$$\sum_{i\in\mathcal{F},j:i\in\mathcal{S}_j}(1-\mu^i_j)Trc\sqrt{Q} \geqslant \mu rcT\sqrt{Q}$$

Then from Eq. (19), we can obtain that

$$\sum_{j=1}^{S}\sum_{k\in\mathcal{R}_j}\sum_{i\in\mathcal{S}_j} W^i_{jk}(n)(Q_k(nT) - Q_j(nT)) \leqslant -\mu Trc\sqrt{Q} - \sum_{i\in\mathcal{F},j:i\in\mathcal{S}_j}\mu^i_j Trc\sqrt{Q} + NB^2(MM_1T + 2M^2T^2) \tag{20}$$

From the burst constraint of the arrival process, we have

$$A_j(n) \leqslant a_jT + b_j = \left(\sum_{k\in\mathcal{R}_j} f_{jk} - \sum_{l:j\in\mathcal{R}_l} f_{lj}\right)T + b_j \tag{21}$$

Combining with Eq. (5) and (6), we have

$$\begin{aligned}\sum_{j=1}^{S} A_j(n)Q_j(nT) &\leqslant T\sum_{j=1}^{S}\sum_{k\in\mathcal{R}_j} f_{jk}(Q_j(nT) - Q_k(nT)) + \sum_{j=1}^{S} b_jQ_j(nT) \\ &\leqslant T\sum_{j=1}^{S}\sum_{i\in\mathcal{S}_j}\mu^i_j r(Q_j(nT) - Q_k(nT)) + \sum_{j=1}^{S} b_jQ_j(nT) \\ &= T\sum_{i\in\mathcal{F},j:i\in\mathcal{S}_j}\mu^i_j r(Q_j(nT) - Q_k(nT)) + \sum_{j=1}^{S} b_jQ_j(nT) \\ &\quad + T\sum_{i\notin\mathcal{F},j:i\in\mathcal{S}_j}\mu^i_j r(Q_j(nT) - Q_k(nT))\end{aligned} \tag{22}$$

Obviously $\sqrt{Q} \geqslant Q_j(nT), j = 1, 2, \ldots, B$, hence

$$\sum_{j=1}^{S} b_j Q_j(nT) \leqslant \sum_{j=1}^{S} b_j \sqrt{Q} = \sqrt{Q} \sum_{j=1}^{S} b_j \tag{23}$$

According to Lemma 3, if $i \notin \mathcal{F}$, we can get

$$Q_j(nT) - Q_k(nT) \leqslant max\{(2MT - M_1), M_2\} \tag{24}$$

Since μ_j^i is the utilization ratio of link i, we also have

$$T \sum_{i \notin \mathcal{F}, j: i \in \mathcal{S}_j} \mu_j^i r(Q_j(nT) - Q_k(nT)) \leqslant rTN \max\{(2MT - M_1), M_2\} \tag{25}$$

Substituting Eq. (23) and (24) into Eq. (22), we yield

$$\sum_{j=1}^{S} A_j(n) Q_j(nT) \leqslant T \sum_{i \in \mathcal{F}, j: i \in \mathcal{S}_j} \mu_j^i r(Q_j(nT) - Q_k(nT)) + \sqrt{Q} \sum_{j=1}^{S} b_j + rTN \max\{(2MT - M_1), M_2\} \tag{26}$$

From Eq. (16) (20) and (26), we have

$$\sum_{j=1}^{S} (Q_j((n+1)T) - Q_j(nT)) Q_j(nT) \leqslant -\mu Trc \sqrt{Q} + NB^2(MM_1T + 2M^2T^2) + TNr \max\{(2MT - M_1), M_2\} + \sqrt{Q} \sum_{j=1}^{S} b_j \tag{27}$$

Substituting Eq. (15) and (27) into (14), we get

$$\triangle Q \leqslant -c_1 T \sqrt{Q} + c_2 T + c_3 T^2 + c_4 \sqrt{Q} \tag{28}$$

Let $T > c_4/c_1$, namely, $T > \sum_{j=1}^{S} b_j / (\mu rc)$, we can obtain

$$\triangle Q \leqslant -\hat{c} Q^{\frac{1}{2}} + c_2 T + c_3 T^2$$

where $\hat{c} = c_1 T - c_4$.

Therefore, $\exists Q_m$, when $Q > Q_m, \sum_{j=1}^{S} Q_j^2((n+1)T) - \sum_{j=1}^{S} Q_j^2(nT) < -\epsilon$, that is, the data center network controlled by CAAR is stable. □

References

[1] A. Vahdat, M. Al-Fares, N. Farrington, R.N. Mysore, G. Porter, S. Radhakrishnan, Scale-Out Networking in the Data Center, IEEE Micro 30 (4) (2010) 29–41.
[2] P. Bodík, I. Menache, M. Chowdhury, P. Mani, D.A. Maltz, I. Stoica, Surviving failures in bandwidth-constrained datacenters, in: ACM SIGCOMM, 2012, pp. 431–442.
[3] C. Guo, H. Wu, K. Tan, L. Shiy, Y. Zhang, S. Luz, Dcell: a scalable and fault-tolerant network structure for data centers, in: ACM SIGCOMM, 2008, pp. 75–86.
[4] M. Al-Fares, A. Loukissas, A. Vahdat, A scalable, commodity data center network architecture, in: ACM SIGCOMM, 2008, pp. 63–74.
[5] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, S. Lu, BCube: a high performance, server-centric network architecture for modular data centers, in: ACM SIGCOMM, 2009, pp. 63–74.
[6] J. Dean, S. Ghemawat, G. Inc, MapReduce: simplified data processing on large clusters, in: USENIX OSDI, 2004.
[7] M. Alizadeh, A. Greenberg, D.A. Maltz, J. Padhye, Data center TCP (DCTCP), in: ACM SIGCOMM, 2010, pp. 63–74.
[8] B. Vamanan, J. Hasan, T. Vijaykumar, Deadline-Aware datacenter TCP (D2TCP), in: ACM SIGCOMM, 2012.
[9] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, M. Handley, Improving datacenter performance and robustness with multipath TCP, in: ACM SIGCOMM, 2011, pp. 265–276.
[10] C. Hopps, Analysis of an equal-cost multi-path algorithm. RFC 2992, Internet Engineering Task Force, 2000.
[11] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, A. Vahdat, Hedera: dynamic flow scheduling for data center networks, in: USENIX NSDI, 2010.
[12] D. Zats, T. Das, P. Mohan, D. Borthakur, R. Katz, DeTail: reducing the flow completion time tail in Datacenter networks, in: ACM SIGCOMM, 2012.
[13] D. Abts, B. Felderman, A guided tour through data-center networking, ACM Queue 10 (5) (2012) 10.
[14] A. Greenberg, J.R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D.A. Maltz, P. Patel, S. Sengupta, VL2: a scalable and flexible data center network, in: ACM SIGCOMM, 2009.
[15] X. Yuan, W. Nienaber, Z. Duan, R. Melhem, Oblivious routing in fat-tree based system area networks with uncertain traffic demands, IEEE/ACM Trans. Netw. (TON) 17 (5) (2009) 1439–1452.
[16] J. Mudigonda, P. Yalagandula, M. Al-Fares, J.C. Mogul, SPAIN: COTS data-center ethernet for multipathing over arbitrary topologies, in: USENIX NSDI, 2010.
[17] T. Benson, A. Anand, A. Akella, M. Zhang, Microte: fine grained traffic engineering for data centers, in: ACM CoNext, 2011, p. 8.
[18] Hari Balakrishnan, Devavrat Shah, Hans Fugal, Jonathan Perry, Amy Ousterhout, Fastpass: a centralized zero-queue datacenter network, in: ACM SIGCOMM, 2014, and correct if necessary.
[19] T.A. Benson, A. Anand, A. Akella, M. Zhang, Understanding data center traffic characteristics, in: The 1st ACM Workshop on Research on ENterprise networking, 2009, pp. 65–72.
[20] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, R. Sherwood, On controller performance in software-defined networks, in: USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services, 2012, pp. 1–10.
[21] A. Elwalid, C. Jin, S. Low, I. Widjaja, MATE: MPLS adaptive traffic engineering, in: IEEE INFOCOM, 2001, pp. 1300–1309.
[22] S. Kandula, D. Katabi, S. Sinha, A. Berger, Dynamic load balancing without packet reordering, ACM SIGCOMM Comput. Commun. Rev. 37 (2) (2007) 51–62.
[23] S. Kandula, D. Katabi, B. Davie, A. Charny, Walking the tightrope: responsive yet stable traffic engineering, ACM SIGCOMM Comput. Commun. Rev. 35 (4) (2005) 253–264. ACM.
[24] X. Wu, X. Yang, Dard: distributed adaptive routing for datacenter networks, in: ICDCS, IEEE, 2012, pp. 32–41.
[25] W. Cui, C. Qian, DiFS: distributed flow scheduling for adaptive routing in hierarchical data center networks, in: ACM/IEEE Symposium on Architectures for Networking and Communications Systems, 2014.
[26] U. Cummings, D. Daly, R. Collins, V. Agarwal, F. Microsystems, F.Petrini, M. Perrone, Fulcrums FocalPoint FM4000: a scalable, low-latency 10 GigE switch for high-performance data centers, in: 17th IEEE Symposium on High Performance Interconnects, 2009, pp. 42–51.
[27] S.-T. Chuang, A. Goel, N. McKeown, B. Prabhakar, Matching output queueing with a combined input/output-queued switch, IEEE J. Sel. Areas Commun. (JSAC) 17 (6) (1999) 1030–1039.
[28] T. Benson, A. Akella, D.A. Maltz, Network traffic characteristics of data centers in the wild, in: ACM IMC, 2010, pp. 267–280.
[29] L. Ying, S. Shakkottai, A. Reddy, On combining shortest-path and back-pressure routing over multihop wireless networks, in: IEEE INFOCOM, 2009, pp. 1674–1682.