CAPS: Coding-Based Adaptive Packet Spraying to Reduce Flow Completion Time in Data Center

Jinbin Hu, Jiawei Huang, *Member, IEEE*, Wenjun Lv, Yutao Zhou, Jianxin Wang, *Senior Member, IEEE*, and Tian He, *Fellow, IEEE, ACM*

***Abstract*— Modern data-center applications generate a diverse mix of short and long flows with different performance requirements and weaknesses. The short flows are typically delay-sensitive but to suffer the head-of-line blocking and out-of-order problems. Recent solutions prioritize the short flows to meet their latency requirements, while damaging the throughput-sensitive long flows. To solve these problems, we design a Coding-based Adaptive Packet Spraying (CAPS) that effectively mitigates the negative impact of short and long flows on each other. To exploit the availability of multiple paths and avoid the head-of-line blocking, CAPS spreads the packets of short flows to all paths, while the long flows are limited to a few paths with Equal Cost Multi Path (ECMP). Meanwhile, to resolve the out-of-order problem with low overhead, CAPS encodes the short flows using forward error correction (FEC) technology and adjusts the coding redundancy according to the blocking probability. Moreover, since the coding efficiency decreases when the coding unit is too small or large, we demonstrate how to obtain the optimal size of coding unit. The coding layer is deployed between the TCP and IP layers, without any modifications on the existing TCP/IP protocols. The test results of NS2 simulation and small-scale testbed experiments show that CAPS significantly reduces the average flow completion time of short flows by ∼30%-70% over the state-of-the-art multipath transmission schemes and achieves the high throughput for long flows with negligible traffic overhead.**

***Index Terms*— Data center, TCP, packet spray, multipath.**

I. Introduction

IN MODERN data centers, a variety of cloud-based applications such as web search and social networking are deployed by a large number of online service providers like Google and Facebook. To obtain better user experience and financial revenue, how to achieve the low latency and high throughput becomes a crucially important issue.

Manuscript received June 2, 2018; revised February 26, 2019 and June 17, 2019; accepted September 25, 2019; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor N. Hegde. Date of publication October 28, 2019; date of current version December 17, 2019. This work was supported in part by the National Natural Science Foundation of China under Grant 61872387, Grant 61572530, and Grant 61872403 and in part by the CERNET Innovation Project under Grant NGII20170107. A preliminary version of this article appeared in IEEE INFOCOM [25], Honolulu, HI, USA, April 2018. *(Corresponding author: Jiawei Huang.)*

J. Hu, J. Huang, W. Lv, and J. Wang are with the School of Computer Science and Engineering, Central South University, Changsha 410083, China (e-mail: jiaweihuang@csu.edu.cn).

Y. Zhou is with the College of Information and Computer Sciences, University of Massachusetts Amherst, Amherst, MA 01002 USA.

T. He is with the Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455 USA.

Digital Object Identifier 10.1109/TNET.2019.2945863

Random Packet Spraying (RPS) [1] splits each flow into packets and spreads the packets to all available paths to achieve high efficiency and easy deployment in the multipath topologies such as FatTree and Clos, which are widely used in the large-scale data center networks (DCNs). In recent years, the packet spraying technique has already been implemented on the commodity switches (e.g. Cisco [2]).

However, RPS neglects the important traffic characteristic that the short and long Transmission Control Protocol (TCP) flows are mixed in DCNs. The data center traffic can be characterized by heavy-tailed distribution [3], [4], that is, around 90% of data is provided by only around 10% of TCP flows, and about 90% of TCP flows provide only about 10% of data. Furthermore, most short flows belong to the delay-sensitive applications, while the majority of long flows are throughput-sensitive. Since RPS does not distinguish between short and long flows, it inevitably leads to the negative interaction between the two kinds of flows.

The following two issues emerge when the packets of short and long flows are mixed over the multiple paths by RPS scheme. The first one is the head-of-line blocking. When the switch buffer is occupied by the packets of long flows, the short flows have to experience large queueing delay, leading to the long-tailed flow completion time (FCT). Secondly, RPS randomly spreads the packets into different paths, possibly resulting in the TCP out-of-order problem.

Coding is a very powerful scheme to address these issues. At the sender, the source packets are encoded and scattered to multiple paths. Though some packets experience the head-of-line blocking or out-of-order, once sufficient encoded packets arrive at the receiver, the original packets can be recovered immediately. In this paper, we propose a coding-based adaptive packet spraying (CAPS), which successfully integrates coding into packet spraying and efficiently avoids head-of-line blocking and packet reordering. To improve the coding efficiency, we provide quantitative analysis of the number of redundant packets and the size of coding unit. Moreover, the transparent coding layer only needs to be deployed between the TCP and Internet Protocol (IP) layers at the end hosts, while making no modifications on the existing TCP/IP protocols.

In summary, our major contributions are as follows:

- We conduct an extensive simulation-based study to analyze two key issues with multipath transmission: the short flows experience large FCT due to the head-of-line blocking caused by the long flows, and the coexisting of short and long flows leads to the reordering problem.

- We propose a multipath transport scheme CAPS, which randomly scatters the encoded packets of short flows to all paths and transmits the long flows to a few paths. Specifically, we design the coding layer, which rationally adjusts the number of redundant packets to resolve the out-of-order and head-of-line blocking problems. To improve the link utilization, CAPS swiftly scatters the packets of long flows to the unused paths by short flows.
- To achieve high coding efficiency, we firstly demonstrate experimentally and theoretically why the FCT performance is impacted by the size of coding unit. Then we give the mathematical analysis on how to obtain the optimal size of coding unit and verify the accuracy of our analysis results.
- By using both Network Simulator 2 (NS2) simulations and small-scale testbed experiments, we demonstrate that CAPS performs remarkably better than the state-of-the-art multipath transmission schemes. Especially, CAPS greatly reduces the average FCT (AFCT) of short flows by $\sim$30%-70% under high workload. Meanwhile, CAPS improves throughput of long flows by up to $\sim$45% and $\sim$35% over RepFlow [5] and Freeway [6], respectively.

The rest of the paper is organized as following. In Section II, we present the related works. In Section III and IV, we respectively describe our design motivation and overview. In Section V and VI, we introduce packet spraying and coding of CAPS, respectively. We discuss the implementation in Section VII. In Section VIII and IX, we show the testbed experimental and NS2 simulation results, respectively. We conclude the paper in Section X. The appendixes provide supplementary experiments related to Incast scenarios and priority queues.

II. Related Works

Recent data center architecture uses a large number of commodity switches organized as multi-rooted tree with multiple equal cost paths from the sender to the receiver. Many existing solutions have been proposed to leverage multiple paths to reduce FCT and improve throughput. As a typical representative of the flow-level transmission scheme, ECMP is extensively used in current fabrics due to its simplicity. However, when there are a few long flows, the hash collision becomes the main cause for TCP performance degradation. That is, multiple long flows are easily transferred through the same path, resulting in queueing delay or packet loss.

There have been lots of efforts to address the shortcoming of ECMP. RPS randomly spreads all packets along different paths, achieving better load balancing and network utilization. However, RPS easily brings about the TCP out-of-order problem, potentially triggering the suboptimal performance. CONGA [7] uses the global congestion information to decide how to route flowlets. However, as CONGA only reroutes when flowlets emerge, it cannot always timely react to congestion, resulting in long-tailed FCT. Presto [8] randomly routes every flowcell with fixed size (i.e., 64KB) among parallel paths. Since Presto is unaware of path conditions, it may suffer from reordering between flowcells in asymmetric environments. The above schemes are flow-size agnostic solutions and are not able to help both short and long flows to obtain high performances at the same time.

Some other schemes are proposed to schedule the short and long flows in different ways to address the problem of performance degradation. The centralized flow scheduling architecture Hedera [9] uses the global first fit algorithm to schedule elephant flows to non-congested paths. Although Hedera avoids the collision between long flows, it still leads to the traffic imbalance between long and short flows without flexibly splitting traffic. Freeway [6] adaptively partitions the parallel paths into low latency paths and high throughput paths and isolates the short and long flows on these two different transmission paths to reduce the impact of two types of flows on each other. Freeway not only guarantees the low latency for real-time traffic but also meets the demand of throughput hungry flows. Based on the RPS and MPTCP [10], MMPTCP [11] randomly scatters packets at the initial time to reduce the FCT of short flows. After a specific amount of data is sent, MMPTCP switches to a regular MultiPath TCP (MPTCP) mode to improve the throughputs of long flows. Karuna [4] schedules a mix of flows with and without deadlines to balance the interests of different kinds of flow. However, these flow-based scheduling schemes are not able to make full use of all available multiple paths to achieve high throughput. As a packet level edge-based transmission scheme, Hermes [12] uses comprehensive sensing to detect the network uncertainties. It considers rerouting only when it will be profitable rather than vigorous path changing. Even the timely yet cautious rerouting in Hermes can mitigate the congestion mismatch problem, it is still hard to avoid the reordering problem.

Based on the multipath diversity, many other multiple paths transmission schemes are proposed to improve the performance of short flows. RepFlow [5] simply replicates each short flow to exploit multiple paths diversity. Then the receiver uses the first flow that finishes its transfer to minimize flow completion times. Although it is simple and effective in reducing the FCT of short flows, the collisions between both kinds of flows still exist in heavy load scenarios. FMTCP [13] employs fountain code to encode transmission data to alleviate the impact of path heterogeneity in MPTCP. FMTCP recovers original data through the subflows from the good paths to avoid retransmission. L^2DCT [14] achieves the LAS scheduling discipline at the sender in a distributed way. According to the flow size that has been sent, L^2DCT distinguishes the short and long flows and assigns higher bandwidth to the short ones, thereby effectively reducing the average flow completion time. However, the improvements on the short flows come at a cost of throughput degradation of the long ones.

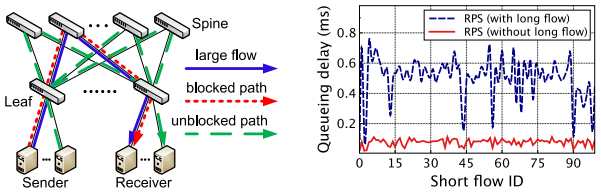
Compared with the above works, our solution CAPS works through a different perspective: we isolate the two kinds of short and long flows on the packet-level to avoid the head-of-line blocking problem and introduce the FEC coding technology to solve the TCP out-of-order issue for short flows. Meanwhile, CAPS flexibly switches the packets of long flows to idle paths to obtain high throughput.

III. Motivation

To motivate our design, we investigate the impact of the long flows on the short ones with existing RPS scheme.

A. Head-of-Line Blocking

In data center, almost 90% of TCP flows are less than 100KB [3]. Based on this characteristic, a flow with its size less than 100KB is considered as a short flow, otherwise, that is a long flow. Since RPS does not isolate short flows from



(a) Leaf-Spine topology (b) Average queueing delay

Fig. 1. Topology and queueing delay.

long ones, the packets of short flows may be spread to the paths occupied by long flows, and have to be queued behind the long flows. Therefore, the short flows may suffer from heavy head-of-line blocking and long-tailed FCT [15], [16].

We conduct NS2 simulation test to analyze the head-of-line blocking problem with the leaf-spine topology as shown in Fig. 1 (a). Each sender sends a flow to a single receiver via multiple switches. The buffer size of each switch is 256 packets. There are 40 equal cost paths between the source and destination nodes. The bandwidth of each path is 1Gbps and the round-trip propagation delay is 100μs. In addition, according to the measurement results in [17], the 75th and 95th percentile of number of concurrent large flows sharing a Top of Rack (ToR) switch are 2 and 4, respectively. In this test, the mixture of 100 short TCP flows (less than 100KB) and 4 long-lived background TCP flows are generated in heavy-tail distribution.

We compare the average queueing delay of short flows under two cases. Firstly, the ToR switch uses RPS to spread all packets of short and long flows to all paths. Secondly, without long flows, only the packets of short flows are spread to all paths. Fig. 1 (b) demonstrates the average queueing delay for the short flows. In the case without the long flows, the short flows do not experience the head-of-line blocking, and thus the average queueing delay of the short flows is reduced significantly by up to around 80%.

B. Out-of-Order

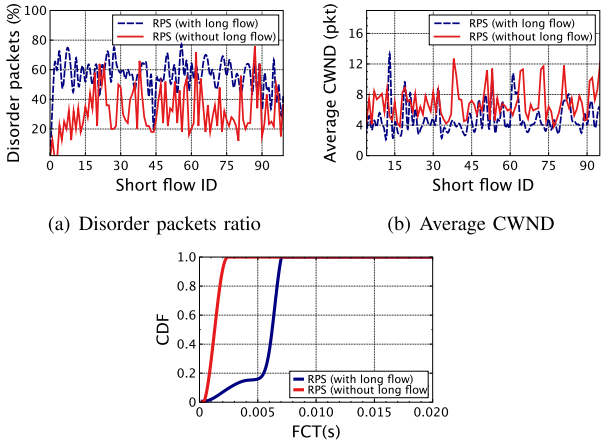
Since RPS randomly spreads the packets to all paths, the long flows potentially make the short flows suffer from the out-of-order problem, which means the later-sent packets may be received ahead of the earlier-sent ones. Fig. 2 (a) shows the ratio of the number of disorder packets to all packets. Compared with scenario without the long flows, the ratio of disorder packets of short flows becomes larger when both kinds of flows are mixed.

When the out-of-order event happens, the TCP sender assumes the packets are lost and then cuts its congestion window, resulting in spurious retransmission and even timeout. In Fig. 2 (b), the average congestion windows of the short flows are about 50% smaller than that in the scenario without the long flows. As a result, the throughput degrades significantly due to the constrained congestion window.

Without the long flows, both the queueing delay and the number of disorder packets in short flows are significantly reduced. Therefore, as shown in Fig. 2 (c), the FCT of short flows is greatly reduced without the impact of long flows.

C. Summary

Our analysis of the coexisting short and long flows leads us to conclude that (i) the short flows experience increasing



(a) Disorder packets ratio (b) Average CWND

(c) CDF of FCT

Fig. 2. Disorder of short flows due to long flows.

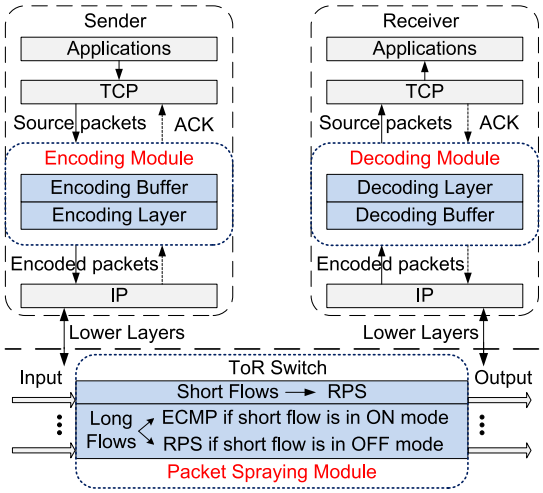


Fig. 3. CAPS architecture.

delay due to the head-of-line blocking once all flows are treated as the same, (ii) the packet reordering due to the long flows seriously enlarges the FCT of short flows. These conclusions motivate us to tackle the above problems by designing and implementing a coding-based adaptive packet spraying scheme.

IV. Design Overview

In this section, we present an overview of CAPS. The two key points of CAPS are using packet spraying and FEC coding to solve the head-of-line blocking and out-of-order problem, respectively. Specifically, on the one hand, when the long flows are limited to a few paths, most packets of the short flows are spread on the paths without blocking and therefore achieve the lower queueing delay. On the other hand, FEC coding eliminates the impact of packets reordering. Even if some encoded packets of the short flows are blocked by the long flows, the original packets can be recovered from the other non-blocked encoded packets. CAPS consists of three modules, as shown in Fig. 3.

(1) **Encoding Module:** At the sender, the encoding module accepts source packets from the transport layer and caches them into an encoding buffer. Then the sender generates $k+r$ encoded packets from k original packets (i.e., a coding unit) in the encoding buffer, and delivers the $k+r$ encoded

packets to the network layer. During the encoding procession, the number of redundant packets r is dynamically adjusted according to real-time state of network traffic. When receiving the Acknowledgement (ACK) packets, the sender removes the ACKed packets from the coding buffer and delivers the ACK packets to the transport layer.

(2) **Packet Spraying Module:** At the switch, the packets of the short flows are spread to all output ports using RPS technique [11], which have already been implemented in many commodity switches. On the other hand, the long flows are transmitted by ECMP [18] or RPS respectively according to the ON or OFF mode of short flows.

(3) **Decoding Module:** At the receiver, the encoded packets from the network layer are cached into the decoding buffer. The original k packets can be decoded from any $k+r$ encoded packets accurately and then handed over to the upper layer.

V. Packet Spraying

To reduce the negative impact of short and long flows on each other, CAPS uses different strategies to transmit the two kinds of flows. When the short and long flows are transmitted at the same time, the long flows are limited to only a few paths by ECMP strategy, while the short flows are simply spread to all paths by RPS to make full use of the multiple paths between any given pair of hosts. The details of CAPS operation on ToR switch are described as following.

(1) **Long flows:** For a long flow arriving at the ToR switch with the short flows existing at the same time, it is forwarded to the next hop by ECMP, which is extensively used as the $de\ facto$ routing algorithm. As identified by the TCP 5-tuple, the TCP flows are randomly hashed to their respective paths. Thus, the small number of long flows are limited to a few paths and the head-of-line blocking problem due to the long flows is avoided for the short flows on the other paths. Moreover, since ECMP is a flow-level scheme, the out-of-order problem of long flows is also resolved.

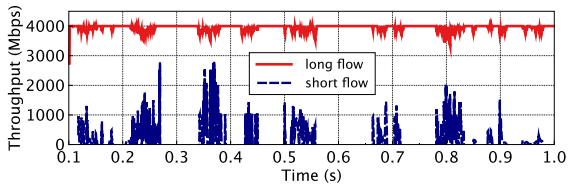
(2) **Short flows:** Once arriving at the ToR switch, the packet of short flows is routed by RPS. Since the packets of short flows are randomly scattered to all available paths to the destination on the packet-level, RPS utilizes all available bandwidth more efficiently than ECMP in terms of the throughput and flow completion time. However, RPS potentially results in the significant packet reordering due to the large queueing delay on the paths with long flows. To overcome the out-of-order problem, we use FEC coding technology to encode the packets of short flows at the sender as illustrated in the following section.

VI. Encoding and Decoding

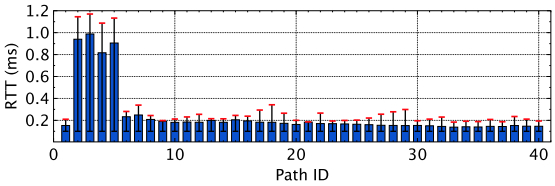
In this section, we firstly give our rational selection for coding algorithm. Then the key point of redundancy and coding unit optimization is discussed. Finally, we analyze the delay improvement and traffic overhead.

A. Coding Algorithm

Forward Error Correction (FEC) technology [13], [19], [20] effectively mitigates the negative impact of packets blocking and reordering. The reason is that FEC only cares about how many, rather than which encoded packets have been received. FEC codes are mainly divided into two categories, called fixed-rate codes [21] and rateless codes [22]. For rateless codes,



(a) The traffic mode of short and long flows



(b) The Round Trip Time (RTT) of each path

Fig. 4. Traffic mode and RTT distribution.

redundancy should be adjusted in real-time according to the varying packet loss or blocking probability. In order to avoid unnecessary redundant packets, the receiver sends the feedback information to the sender to stop encoding for the current coding unit [23], [24], unavoidably increasing the latency and also reducing the robustness of data transmission.

In our CAPS, we use the fixed-rate codes due to the following reasons. Firstly, the fixed-rate coding scheme works well when the blocking rate does not change rapidly. Here, we investigate the probability of a short flow being blocked by long flows (i.e., blocking probability). We conduct a simulation in NS2 with the same settings as described in Section III. As shown in Fig. 4 (a), the traffic of short flows shows ON/OFF mode, in which the packets of short flows start and finish their transmissions during the ON periods. Both the ON and OFF periods follow exponential distribution and the OFF period is much longer than the bursty ON period [3]. During the ON periods of the short flows, the long flows are always existing because the long flows have much larger flow size. This phenomenon means that, for short flows, the blocking probability is fixed during their lifetime.

Secondly, the blocking probability can be estimated in advance at the sender based on the measurement of RTT. As shown in Fig. 4 (b), 4 paths with long flows have much larger RTT than the other paths with only short flows. The blocking probability by long flows can be estimated as 0.1, which is the ratio of the number of paths with large RTT to the total number of paths. However, since CAPS randomly spreads packets of the short flows to all paths on the ToR switch, the sender is unable to directly obtain the exact RTT for each path. Here, we utilize the TCP congestion control mechanism. Specifically, when the sender receives the ACK packets, based on the corresponding RTT for each ACK, the blocking probability is calculated as the ratio of the number of ACK packets with large RTT to the total number of received ACK packets. According to the RTT statistics, the empirical threshold for the large RTT is set as 2x average RTT of all packets.

The classical fixed-rate codes include Reed-solomon codes (RS) and Low Density Parity Check Codes (LDPC). Since RS is suitable for the short code unit with total number of bits less than 1000, we choose LDPC because it is more practical to combine multiple packets (i.e., 1500 Bytes for each packets) into a code unit [21]. As shown in Fig. 5, a code unit

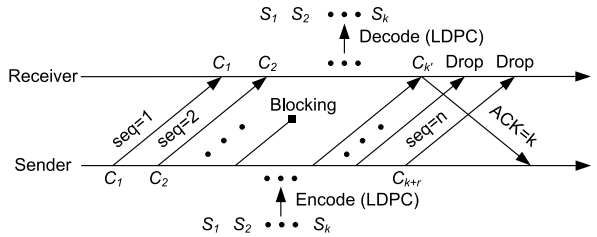


Fig. 5. Data transmission for a coding unit.

containing k source packets (S_1, S_2, ..., S_k) is encoded into $k+r$ encoded packets (C_1, C_2, ..., C_{k+r}) at the sender. The value of r can be changed at each encoding unit according to the blocking probability. The receiver performs decoding process to reconstruct the original packets from any set of k' encoded packets, for k' slightly larger than k. Here, k is set to the congestion window size of short flows. Since k' can be approximated by k when the number of bits of a code unit is large (i.e., >1000 bits) [20], for simplicity, we substitute k for k' in the following sections. At the receiver, the subsequent received encoded packets belonging to the same coding unit are dropped directly.

B. Redundancy Optimization

In the encoding operation, the coding redundancy affects both the decoding delay and traffic overhead. It is hard to balance these two aspects. For instance, in order to speed up the decoding operation, the sender should send more redundant packets, which unavoidably brings about unnecessary traffic overhead and limits the transmission rate of the sender by itself. However, if too less redundant packets are transferred, the decoding speed is limited because the receiver has to wait for enough encoded packets for decoding operation. In brief, the coding redundancy should be elaborately adjusted to achieve good tradeoff between the decoding delay and traffic overhead. We give the redundancy optimization as following.

Let n_L and n denote the number of ACK packets with large RTT and the total number of received ACK packets, respectively. Then we get the blocking probability p_B of a short flow blocked by long flows as $p_B = \frac{n_L}{n}$.

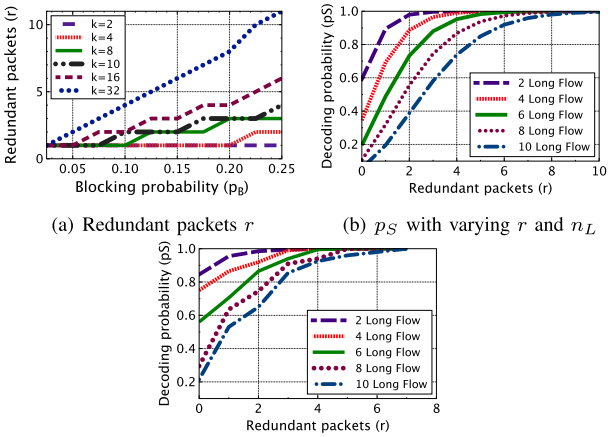
A code unit has k source packets and r redundant packets. That is, k source packets are encoded into $k+r$ encoded packets. To guarantee that at least k encoded packets reach the receiver without blocking, the following equation $(1-p_B)\times(k+r)\geq k$ should be satisfied.

To reduce the traffic overhead, the number of redundant packets r for each k source packets is set as $r=\lceil\frac{k}{1-p_B}-k\rceil$. Fig.6 (a) shows the number of redundant packets r with increasing p_B. For the higher blocking probability or larger coding unit, the sender uses more redundant packets to compensate the blocked packets and achieve the high successful decoding probability.

Unfortunately, since the short flows are randomly spread to all paths, the decoding probability of short flows is not 100%, because some unlucky packets may be blocked or even dropped on the paths with long flows. Here, we analyze the successful decoding probability p_S of short flows.

Within $k+r$ encoded packets, supposing the number of blocked packets by long flows is j, we obtain the probability of any blocked j encoded packets out of $k+r$ packets as

$$p_B(j) = C_{k+r}^{j} \times {p_B}^{j} \times (1-p_B)^{k+r-j}. \quad (1)$$



(a) Redundant packets r (b) p_S with varying r and n_L

(c) NS2 simulation for p_S

Fig. 6. Numeric comparison and NS2 simulation.

The receiver can successfully decode the original packets only when the number of blocked packets is no larger than the number of redundant packets r. Then the successful decoding probability p_S is computed as

$$p_S = \sum_{j=0}^{r} p_B(j) = \sum_{j=0}^{r} C_{k+r}^{j} \times {p_B}^{j} \times (1-p_B)^{k+r-j}. \quad (2)$$

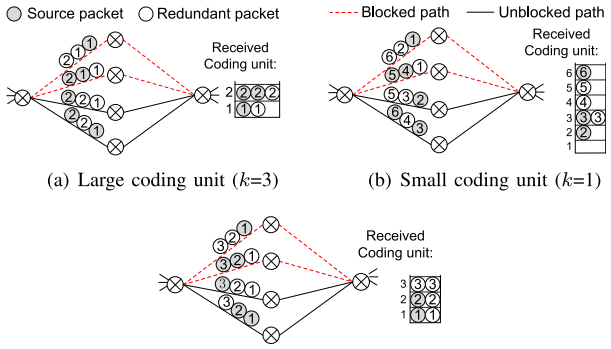
The numeric comparison of the successful decoding probability p_S is shown in Fig. 6 (b). The number of paths and the coding unit size k are set as 40 and 10, respectively. In the numeric analysis, we simply substitute the ratio of the number of long flows' paths to the total number of paths for the blocking probability. With the increasing number of redundant packets r, the successful decoding probability becomes larger due to more unblocked packets. Furthermore, it is much easier to decode successfully with a smaller number of long flows. For 2 long flows, the successfully decoding probability reaches almost 1 when r is 2. We also conduct the simulation experiments in NS2 with the same settings in Section VI-A. Fig.6 (c) shows the test result, which is consistent with the trend of numeric analysis.

C. Coding Unit Optimization

In our preliminary work [25], the size of coding unit k is simply set to the congestion window w. When the congestion window is small, the probability of hitting the blocked paths for small coding unit is very low. However, with the growth of the TCP flow size in many datacenter applications, the congestion window is likely to increase too large [26]. If k is still set to large w, the large coding unit easily experiences large tailed delay. In this section, we optimize the size of coding unit to achieve high coding efficiency.

We use an example to show the impact of coding unit size k. As illustrated in Fig. 7, there are 2 blocked paths in 4 equal cost paths. The sender generates 6 redundant packets for the 6 source packets.

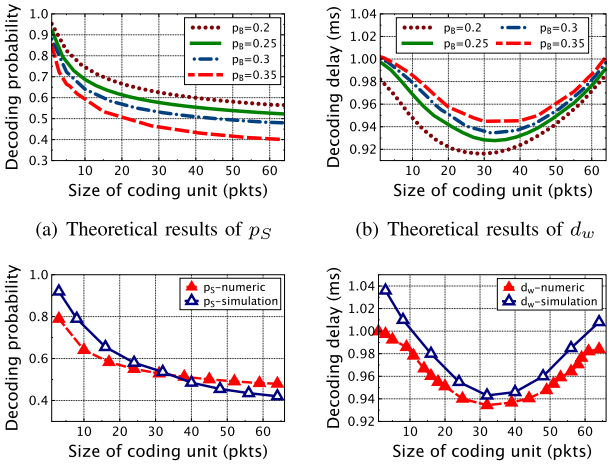
Fig. 7 (a) shows the case of 2 coding units with k of 3. Since 6 packets are blocked, the source packets can not be recovered at the receiver and have to wait for the blocked packets. This result shows that, when the packets of a coding unit are scattered on multiple paths, the packets of large coding unit are easily blocked, resulting in long-tailed delay. As shown in Fig. 7 (a), there are 3 redundant packets for a coding unit



(a) Large coding unit (k=3) (b) Small coding unit (k=1)

(c) Appropriate coding unit (k=2)

Fig. 7. Coding efficiency with varying k.



(a) Theoretical results of p_S (b) Theoretical results of d_w

(c) p_S with varying k (d) d_w with varying k

Fig. 8. Theoretical and simulation results.

of 3 source packets. The probability that more than 3 packets are blocked is $\sum_{j=4}^{6} C_6^j \times (\frac{1}{2})^j \times (1-\frac{1}{2})^{6-j} = 0.34$. Then, the successful decoding probability of the first 3 received packets is $1 - 0.34 = 0.66$.

Without loss of generality, we use Equation (2) in Section VI-B to calculate the successful decoding probability p_S for first k packets in a coding unit. Fig. 8 (a) presents the theoretical results of p_S with increasing of k and p_B. In Fig. 8 (a), when the coding unit becomes larger, the successful decoding probability p_S decreases, because more packets are easily blocked with increasing size of coding unit.

However, if the size of coding unit is too small, the decoding capability is impaired. From the macroscopic point of view, since the all packets are divided into different coding units, the decoding delay for all packets is determined by the stalled coding unit. Though small coding unit has large successful decoding probability, the less number of associated encoded packets in each coding unit leads to low recovery capability and large decoding delay of all packets. As shown in Fig. 7 (b), when k is reduced to 1, the number of coding units is 6. We find that, if 6 packets are blocked, the 6 received packets are still not able to decode because not all coding units receive enough packets. In this case, the decoding successful probability by the first 6 received encoded packets is only $\frac{(C_2^1)^6}{C_{12}^6} = 0.07$.

Fortunately, if we set the size of coding unit to a appropriate value, the high coding efficiency is achieved. We give the example of 3 coding units shown in Fig. 7 (c). When k is 2, all of the 6 source packets are recovered immediately by the first 6 received packets. This result shows that a proper value of k achieves a good tradeoff between the tailed delay and the decoding capability. On the one hand, a large coding unit easily experiences large delay due to the tailed packets. On the other hand, the decoding capability of whole data is impacted as k decreases. Since any encoded packets arriving at the receiver can only be used to recover the source packets in the same coding unit, if k is too small, it is hard for all coding units to receive enough packets to decode.

Next, we give the mathematical analysis on how to obtain the optimal size of coding unit. Firstly, we calculate the buffering delay d_B in decoding operation. Specifically, in the decoding operation, the receiver should buffer k encoded packets before reconstructing the original packets, introducing the extra buffering delay d_B. Since the k packets are sent back-to-back by the sender, the ith packet has to wait for the rest $k-i$ packets, with the buffering time as $\frac{(k-i)MSS}{C}$. Here, C and MSS denote the bottleneck link capacity and the size of a TCP segment, respectively. The average buffering delay for waiting any set of k encoded packets within a coding unit is calculated as

$$d_B = \sum_{i=1}^{k} \frac{(k-i)MSS}{C} \times \frac{1}{k} = \frac{(k-1)MSS}{2C}. \quad (3)$$

Then, we calculate the decoding delay d_w for a congestion window w as

$$d_w = p_S^{\frac{w}{k}} \times (RTT_S + d_B) + (1 - p_S^{\frac{w}{k}}) \times RTT_L, \quad (4)$$

where RTT_L and RTT_S are the maximum values of the RTT of paths with and without long flows, respectively. Here, since the buffering delay for decoding operation is much smaller than RTT_L, we only consider RTT_L for the blocked packets. Based on Equation (4), we obtain the optimal size of coding unit. Specifically, we compute the corresponding value of decoding delay d_w under different unit size k, and choose the optimal size of coding unit as the corresponding value of k to get the minimum d_w.

Fig. 8 (b) presents the theoretical results of the decoding delay for a congestion window d_w with varying k and p_B. In Fig. 8 (b), when k is relatively small, d_w decreases with increasing of k. This is because that, for smaller k, more coding units in a congestion window should obtain enough packets to decode although the decoding probability p_S of each unit is higher. For large k, d_w increases when k becomes larger, because more packets are easily blocked, resulting in lower decoding probability of each unit.

We evaluate the accuracy of the theoretical analysis by using NS2 simulations. We use leaf-spine topology with 10 equal cost paths. The senders generate 3 background TCP flows and 100 short TCP flows less than 100KB. The blocking probability of short flows is 0.3 in this test. The other settings are same as that in Section III. Firstly, we measured the proportion of the coding units that can be immediately decoded by unblocked packets. Fig. 8 (c) shows that p_S decreases monotonically with the increase of k. Then, we statistic the average decoding delay of congestion window. Fig. 8 (d) shows that too small or large size of coding unit leads to large decoding delay. Both the decoding probability and delay in numerical analysis closely follow the corresponding simulation results.

D. Delay Performance Analysis

In our designed CAPS, the long flows are limited to a few paths when the short flows are transmitted. For the short flows, once enough number of encoded packets arrive, the original packets can be recovered immediately by the receiver, thus greatly reducing the transfer delay. However, the redundancy still leads to the buffering delay for decoding operation at the receiver. Here, we analyze the delay performance of short flows without and with coding.

(*i*) Without Coding

Given the blocking probability p_B for each packet, the probability for a congestion window with w packets transmitted through the paths without long flows is $(1-p_B)^w$. Then the probability for at least a packet experiencing the paths with long flows is $1-(1-p_B)^w$. Finally, we obtain the average delay d_{wnc} of w packets without coding as

$$d_{wnc} = (1-p_B)^w \times RTT_S + (1-(1-p_B)^w) \times RTT_L. \tag{5}$$

(*ii*) With Coding

Though introducing the buffering delay in the decoding operation, the coding operation helps short flows to avoid the impact of blocked packets by long flows. Note that as long as the buffering delay is too small enough to be negligible compared to the delay caused by retransmission or even timeout, the coding method has significant benefit on delay. The delay improvement is analyzed as following.

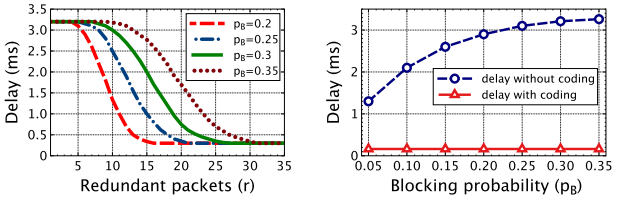
For a coding unit, when at least k encoded packets are transferred through the paths without long flows and successfully recovered at the receiver, the total delay of successful decoding includes both RTT_S and the buffering delay. The probability for this case is the successful decoding probability. Thus, the average coding delay d_w for a congestion window with $\frac{w}{k}$ coding units is calculated as Equation (4). Taking Equation (2) and (3) into (4), we obtain

$$d_w = RTT_L + (RTT_S - RTT_L + \frac{(k-1)MSS}{2C}) \times (\sum_{j=0}^{r} C_{k+r}^{j} \times {p_B}^j \times (1-p_B)^{k+r-j})^{\frac{w}{k}}. \tag{6}$$

Fig. 9 (a) shows the coding delay d_w decreases as the redundant packets increasing with a certain blocking probability p_B. The bottleneck link capacity C is 1Gbps and the size of a TCP segment MSS is 1500 Bytes. We set the size of congestion window w to 64, then the optimal k values are calculated to be 29, 31, 33 and 40 by the Equation (4) under the different p_B of 0.2, 0.25, 0.3 and 0.35, respectively. Based on the measurement result of the maximum RTT on the paths with and without long flows, RTT_S and RTT_L are set to 0.1ms and 3ms, respectively. When the blocking probability p_B is increased, the corresponding values of d_{wnc} and d_w are shown in Fig. 9 (b). It is clear that d_{wnc} is much greater than d_w, which means the total delay is significantly reduced by the coding operation.

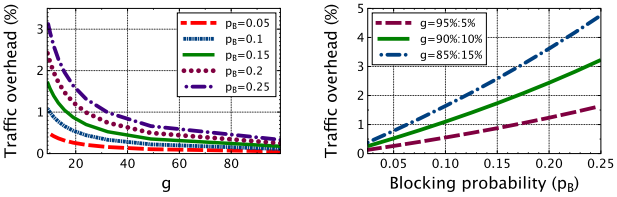
E. Traffic Overhead Analysis

The coding operation adds more redundant packets to the network system, bringing about the traffic overhead. However, our CAPS design only encodes the short flows, which account for about 10% of the total network traffic. That means the traffic overhead of CAPS is limited. Here, we analyze the traffic overhead of CAPS.



(a) Increasing r (b) Increasing p_B

Fig. 9. Coding delay with increasing r and p_B.



(a) Increasing g (b) Increasing p_B

Fig. 10. Traffic overhead with increasing g and p_B.

Supposing the ratio of total packets of long flows to short flows is g and the total packets of short flows is s, the load overhead η is calculated as

$$\eta = \frac{\frac{s}{k} \times r}{s + s \times g + \frac{s}{k} \times r} = \frac{p_B}{1 + g \times (1-p_B)}. \tag{7}$$

In Fig. 10 (a), the ratio of total packets of long flows to short flows g is set to from 90%:10% to 99%:1%, which is heavy-tailed distribution as illustrated in Section 1. The results show that the traffic overhead increases with larger g, but is always less than 3.5%. In Fig. 10 (b), it is clear that the traffic overhead under the increasing blocking probability is less than 5%, that is small enough to be negligible.

VII. Implementation

We implement the design of CAPS with two key points. The first one is to guarantee the throughput of long flows. When long flows are transmitted on a few paths, the short flows on most paths will not be blocked, while the throughputs of the long flows will be unavoidably decreased. Furthermore, unlike short flows, the throughput is much more important than FCT in view of the long flows. Thus, it is necessary to alleviate this performance impairment of long flows.

Fortunately, we can take advantage of the ON/OFF traffic pattern of short flows to adaptively adjust the number of paths for long flows to deal with rapid changes of network dynamics and make full use of available multiple paths. As shown in Fig. 4 in Section VI-A, the short flows are only transmitted during the ON periods, leaving the unused paths during the long OFF periods. Thus, to ensure the throughput of long flows without making damage to short flows meanwhile, we adopt different strategies to control the long flows according to the traffic mode of short flows.

The packet spraying algorithm of CAPS is specified in Algorithm 1. Specifically, CAPS samples the short flows periodically at the switch. When none packet of short flows is received during the sampling interval T, the mode of short flows is set as OFF and vice versa. Here, the sampling interval T is set as 500μs [7], which is the general inactivity gap between two bursts of packets in short flows. If the short

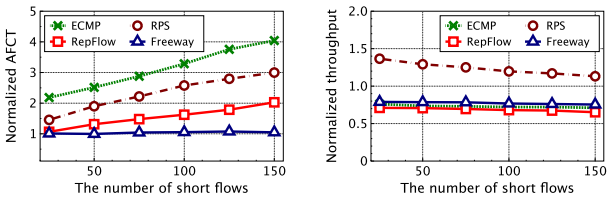
flows are in OFF mode, the long flows will be spread to all paths with RPS to achieve high throughput. Otherwise, the long flows are limited to a few paths with ECMP to avoid the impact on the short flow. CAPS is easy to implement in the switch. It only requires fewer than 50 lines of code change for routing.

Algorithm 1 Packet Spraying Algorithm of CAPS

1: **Initialization:**
2: $fs[\,] \leftarrow \phi$; $s_L \leftarrow 100KB$; $T \leftarrow 500\mu s$; $sfmode \leftarrow OFF$
3: When the timer expires after T:
4: **if** *received a packet with its flow size* $< s_L$ *in* T **then**
5: $sfmode \leftarrow ON$;
6: **else**
7: $sfmode \leftarrow OFF$;
8: reset the timer;
9: On receiving a packet P from TCP flow i: $fs[i]$ = $fs[i]$ + packet size of P ;
10: **if** *(*$fs[i] \geq s_L$*) and (*$sfmode$ == *ON)* **then**
11: transmit P using ECMP within n_L paths;
12: **else**
13: spray P using RPS within n paths;

The other key consideration of our CAPS is that, in most cases, it is hard to obtain accurate flow size information at the start of a flow. For example, the partial results for online query responses are typically transferred when they are generated, instead of waiting for the end of the query execution. Thus under these situations, CAPS needs to work in the dark even without prior knowledge. In the absence of prior knowledge, CAPS considers all flows as short flows in the beginning, and scatters all packets to all paths. When the amount of data belonging to one flow is larger than the threshold for large flows (i.e., 100KB) [3], [5], [11], [27], the flow is distinguished as a long flow and then transmitted in the different way. Unlike MMPTCP, CAPS transmits long flows by using ECMP when short flows are in the ON mode and spreads long flows to all paths when short flows are in OFF mode. For short flows, CAPS sprays the packets of short flows to all paths as MMPTCP, but also encodes the short flows using FEC technology. The experimental results in Section IX show that CAPS works well in the dark.

In this paper, the long flows select their paths by using ECMP when the short flows are in the ON mode. That is, CAPS does not limit the number of paths for long flows. n_L is a periodic statistical variable, which is the number of large RTTs continuously updated by the sender in a sampling period (i.e. 200μs). Therefore, n_L in CAPS does not make the performance of long flow worse. To protect short flows, there will be hash collisions when ECMP is used for long flows in the ON mode of short flows. However, when the short flows are in the OFF mode, which is longer than the ON mode, CAPS sprays the packets of long flows by using RPS. Thus, CAPS alleviates the hash collision problem and achieves good throughputs for long flows. In our implementation, we do not use the priority queues to avoid the head-of-line blocking caused by the long flows. The reason why we do not choose this priority scheduling is the flow starvation problem. When there are many concurrent short flows occupying the higher priority queue, the packets of long flows are easy to get starved in the lower priority queue. The flow starvation may frequently interfere with the TCP transmission scheme such as retransmission and timeout. In the high workload, this problem even terminates a TCP connection and degrades the application performance significantly. Therefore, we choose the coding scheme to solve the head-of-line blocking problem. The relevant experimental results are shown in Appendix B.



(a) AFCT of short flows (b) Throughput of long flows

Fig. 11. Varying the number of short flows.

VIII. Testbed Evaluation

In this section, we firstly use a Mininet implementation to test CAPS. Then we conduct the small-scale testbed experiment in Tencent cloud.

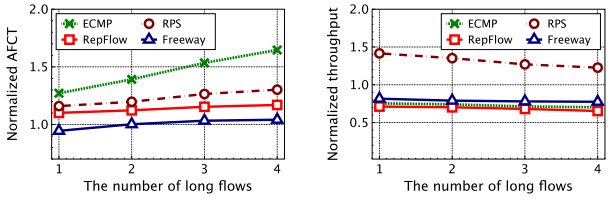
A. Mininet Implementation

In this section, we use a realistic Mininet implementation [5], [28], [29] in a small-scale testbed to test CAPS's broad applicability and effectiveness. We implement CAPS on Mininet, a network emulation system based on Linux kernel using virtualization. Mininet's virtual hosts, switches, links and controllers are real components running on the standard Linux kernel, and for the most part their behavior is similar to the discrete hardware elements.

In this test, CAPS is implemented in Mininet 2.3.0 on a Ubuntu kylin 16.04. The test topology is leaf-spine network with 20 equal cost paths shown in Fig. 1(a). We set the link bandwidth to 20Mb and delay to 1ms [5]. POX is installed as the controller on switches to support ECMP and RPS. The buffer size at switches is 256 packets. The default numbers of short flows and long flows are 100 and 4 [30], respectively. The sizes of short flows are randomly distributed within 100KB. The sizes of long flows are larger than 10MB [31]. The overall traffic obeys heavy-tailed distribution as illustrated in [3].

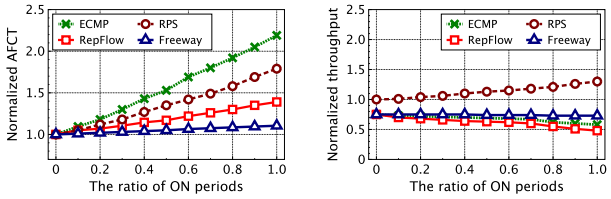
We normalize the results of ECMP, RPS, RepFlow and Freeway to that of CAPS. Then we compare the performances of them with varying number of short or long flows. RepFlow simply replicates each short flow to reap multipath diversity to minimize the flow completion time. Freeway adaptively partitions the paths into low latency paths and high throughput paths respectively for the short and long flows to alleviate the impact of long flows to the short ones.

In Fig. 11 (a), the normalized AFCT of ECMP, RPS, RepFlow and Freeway is larger than 1, meaning that CAPS achieves the smaller AFCT of short flows. Specifically, CAPS reduces the AFCT of short flows by ∼50%-70%, ∼40%-60%, ∼30%-40% and ∼1%-5% with increasing the number of short flows over ECMP, RPS, RepFlow and Freeway, respectively. This is because CAPS is not affected by the head-of-line blocking or packets reordering, only requires to receive sufficient packets regardless of which path they come from. Since ECMP and RPS are agnostic to the short and long flows, the short flows suffer from the long queueing delay and long-tailed FCT. The performance of RepFlow is worse



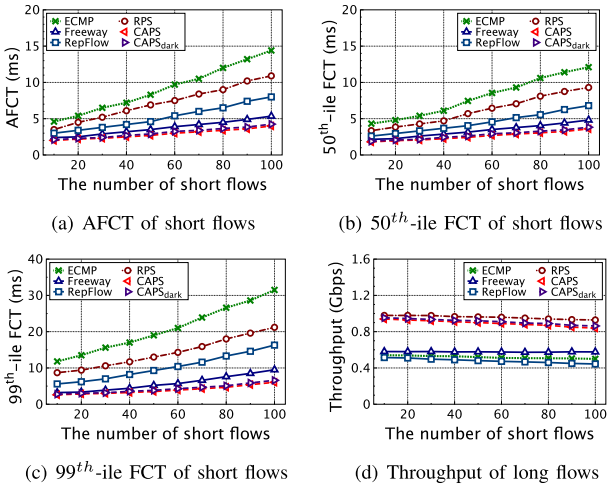
(a) AFCT of short flows (b) Throughput of long flows

Fig. 12. Varying the number of long flows.



(a) AFCT of short flows (b) Throughput of long flows

Fig. 13. Varying the ratio of ON periods to total ON and OFF periods of short flows.



(a) AFCT of short flows (b) 50th-ile FCT of short flows

(c) 99th-ile FCT of short flows (d) Throughput of long flows

Fig. 14. Varying the number of short flows.

than CAPS, because RepFlow only replicates, but does not isolate the short flows from the long flows on different paths. Moreover, the traffic overhead of RepFlow limits its performance improvement under the heavy load.

Fig. 11 (b) shows the normalized throughputs of long flows with increasing number of short flows. CAPS improves the throughputs of long flows by ∼30%-40%, ∼25%-45%, ∼25%-35% over ECMP, RepFlow and Freeway, respectively. The reason is that ECMP, RepFlow and Freeway employ only one path to transfer each long flow, resulting the low utilization on the multiple paths. CAPS and RPS work on the packet-level and flexibly scatter the packets to all paths, obtaining the higher throughputs of long flows. Since CAPS sprays the packets of long flows to all paths only during the OFF periods of short flows, the throughputs of long flows in CAPS are lower than that of RPS, showing the tradeoff between the delay gain of short flows and the throughput loss of long flows.

Fig. 12 (a) shows the normalized AFCT with varying number of long flows. Compared with the other schemes, CAPS achieves the better performance. However, with the increasing number of long flows, much more traffic is injected into the network system. Moreover, more long flows lead to larger blocking probability and thus more redundant packets after the coding operation, making the network congestion heavier. Thus, the performance improvement for short flows becomes less compared with the case of increasing only the number of short flows. Fig. 12 (b) shows that CAPS obtains the higher throughputs of long flows compared with ECMP, RepFlow and Freeway.

We test the CAPS performance under different ratios of ON periods of short flows. Specifically, we use the leaf-spine topology with 20 equal cost paths. The other parameters like the bandwidth, link latency, buffer size and flow size distribution are the same as the experiments in this section. There are 3 long flows each taking a different path without hash collision. The default number of short flows for each ON period is 50.

As shown in Fig.13 (a), the improvement of CAPS in the average FCTs of short flows increases with larger ratio of ON periods. CAPS outperforms the others because the long flows are switched to ECMP during ON periods of short flows, thus reducing the impacts on short ones. In Fig.13 (b), when the ratio of ON periods of short flows is 0, CAPS sprays long flows to all available paths without switching and therefore gets the same performance as RPS. With the increasing of ON periods of short flows, the long flows are transmitted in ECMP for longer period. In this case, although the long flows suffer from throughput loss because they do not utilize all paths, the short flows obtain great performance gain in AFCT.

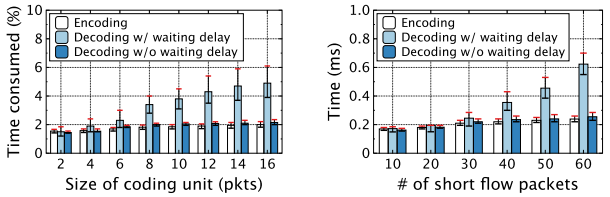
B. Testbed in Cloud

We conduct the testbed experiment in Tencent cloud with the real environment of data center network. Specifically, in this part, the performances of 6 different schemes including ECMP, RPS, RepFlow, Freeway, CAPS and CAPS$_{dark}$ are evaluated in the Tencent cloud, which is a leading public cloud service provider in China. We build a leaf-spine testbed to create 6 paths between any pairs of servers from 2 ToR switches as shown in Fig.1. We use Cloud Virtual Machine (CVM) as servers with dual core Intel Xeon Skylake 6133 CPU and CentOS 7.2 (kernel 3.10.0-327.e17.x86_64) installed. The link bandwidth is 1Gbps and the switch buffer size is 512 packets. We generate 2 long flows (>10MB) and short flows (<100KB) between random pairs of servers following a Poisson process. The number of short flows varies from 10 to 100.

Fig.14 shows the performance of each scheme as the number of short flows increases. In Fig.14 (a), CAPS outperforms the other schemes in the average completion time of short flows because CAPS uses the encoded packets from non-congested paths to recover the source packets and mask the negative effects of reordering and packet loss. Fig.14 (b) and Fig.14 (c) show the 50th and 99th percentile FCT of short flows, respectively. For CAPS, since the short flows avoid reordering and being blocked by long flows through coding, the tailed FCT is reduced by ∼60%-∼70%, ∼55%-∼65%, ∼45%-∼55% over ECMP, RPS and RepFlow, respectively. Fig.14 (d) shows the average throughput of long flows. Since each long flow in ECMP, RepFlow and Freeway can only be transmitted on a single path, CAPS and RPS outperform these three schemes by up to 45%.

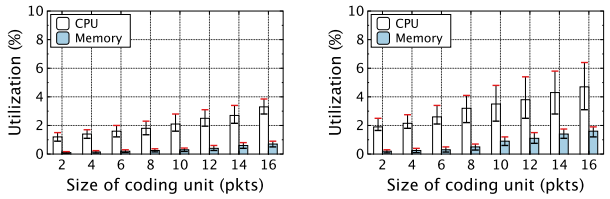
C. Coding Overhead

The coding and decoding operations bring about the additional processing delay in TCP stack. We measure the



(a) The ratio of encoding and decoding time to RTT. (b) Encoding and decoding time.

Fig. 15. Coding delay overhead.



(a) Encoding at sender (b) Decoding at receiver

Fig. 16. CPU and memory overhead at end-hosts.

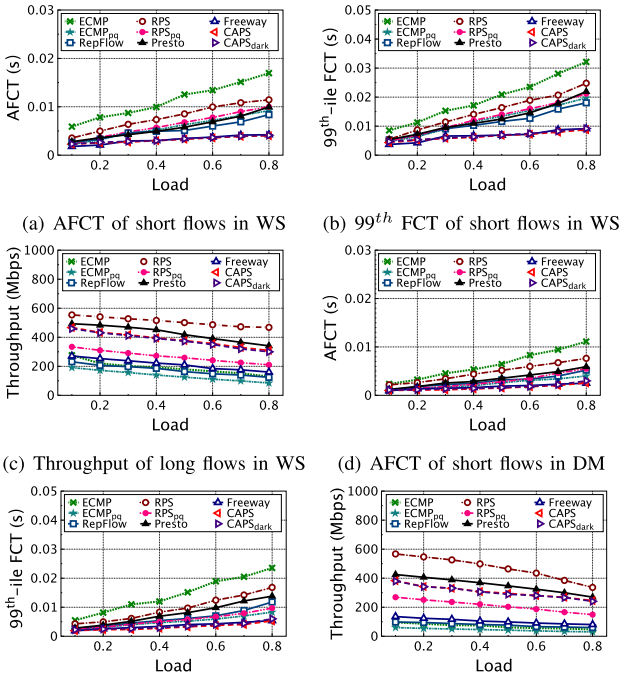
encoding and decoding delay to test the impact of the delay caused by coding and decoding operations. Moreover, we test the CPU and memory utilization ratio at the end-hosts.

Firstly, we measure the ratios of encoding delay and decoding delay to the round trip time on the default experimental settings in Section VIII-A. Since the receiver conducts the decoding after all the packets in coding unit arrive, to give the details of decoding delay, we measure the decoding delay including and excluding the waiting time for the late-arriving packets in the same coding unit. As shown in Fig.15 (a), the ratios of encoding and decoding time increase with larger coding unit. This is because more data packets are involved in encoding and decoding processes, resulting in larger processing delay. The average time consumed by both encoding and decoding without waiting delay is less than 5% even if the coding unit size increases to 16.

Secondly, we test the processing delay of LDPC itself. Specifically, we measure the encoding and decoding time with different sizes of short flows at end-hosts. In this test, the coding unit size equals to the congestion window size during transmission and the initial congestion window is set as 10. As shown in Fig.15 (b), with increasing number of packets in each short flow, the encoding and decoding delay without waiting for late-arriving packets in coding unit remains constant at about 0.2ms, while the decoding time with waiting delay increases from 0.2ms to 0.6ms. This result indicates that the impact of processing delay of coding is very small compared with the gain in flow completion time (i.e., about 20ms) of short flows.

Finally, in order to evaluate the system overhead of CAPS, we gradually increase the coding unit size from 2 to 16 and measure resource consumption at end-hosts with Intel Xeon E5-2687W CPU and 2GB memory. The results of average, maximum and minimum CPU and memory consumption for encoding and decoding are shown in Fig.16 (a) and Fig.16 (b), respectively. The average and maximum CPU consumption are less than 5% and 6.5%, respectively. For memory consumption, the cost is less than 2% of memory under different sizes of coding unit.

Overall, since CAPS only encodes short flows, which account for less than 10% of the total traffic in data centers [3], [4], [5], [17], the corresponding coding and decoding operations do not incur excessive procession delay and system overhead.



(a) AFCT of short flows in WS (b) 99^{th} FCT of short flows in WS

(c) Throughput of long flows in WS (d) AFCT of short flows in DM

(e) 99^{th} FCT of short flows in DM (f) Throughput of long flows in DM

Fig. 17. Web search (WS) and data mining (DM) applications.

IX. Simulation Evaluation

To evaluate the performance of CAPS in the large-scale scenarios, we conduct the NS2 simulation tests in the web search [17] and data mining [32] application scenarios. In the web search scenario, 30% of flows larger than 1MB provide more than 95% bytes. In the data mining scenario, $\sim$ 3.6% flows larger than 35MB provide 95% bytes, while around 80% of flows are less than 100KB.

A. Performances in Leaf-spine

We use the leaf-spine topology with 24 ToR switches, each of which connects to 36 hosts. The whole network has 864 hosts and 12 core switches. There are 12 equal cost paths between any pair of hosts. All flows are generated between random pairs of hosts following a Poisson process with load varying from 0.1 to 0.8 to thoroughly evaluate CAPS's performance. We also test the performance of CAPS without prior knowledge of flow size.

Here, we focus on the flow completion time of short flows and the throughputs of long flows with the web search and data mining workloads as shown in Fig. 17. Fig. 17 (a) and Fig. 17 (d) show the average flow completion time, while Fig. 17 (b) and Fig. 17 (e) give the 99^{th} percentile FCT, presenting the tail FCT. Fig. 17 (c) and Fig. 17 (f) show the throughputs of long flows.

We observe that CAPS improves the AFCT and tail FCT significantly compared to the other schemes except for Freeway, especially in high workloads. Presto [8] achieves better load balancing at flowcell granularity (i.e., 64KB) and mitigates the packet reordering, but it is agnostic to congestion

and does not distinguish between short and long flows. Thus, the FCT performance of Presto is not better than CAPS. Under the web search workload, CAPS reduces the AFCT of short flows by ∼60%-70%, ∼50%-60%, ∼40%-55% and ∼40%-50% for loads from 0.3 to 0.8 over ECMP, RPS, Presto and RepFlow, respectively.

The results demonstrate the advantage of CAPS in the typical multi-path topology. CAPS avoids the impact of out-of-order and head-of-line blocking problems due to the mixture of short and long flows. When the workload becomes high, more long flows occupy more paths, with the result that more short flows hit the long tail. CAPS is able to get enough gain by adaptively adjusting the redundancy of short flows and limiting the number of long flows' paths. For the other schemes, since most short flows experience large queueing delay, the delay performance is degraded. Only the result of Freeway is close to CAPS, because Freeway dynamically separates the short and long flows. We also test the performances of ECMP and RPS with the priority queue, which assigns high priority to short flows. With the help of priority queue, the performances of short flows in ECMPpq and RPSpq are improved compared with ECMP and RPS.

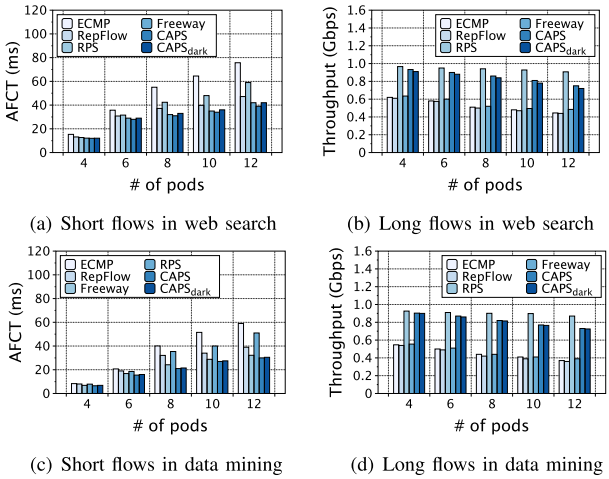
Moreover, though CAPS$_{dark}$ has not any prior knowledge of the flow size, the impact of long flows on the short flows is negligible when the long flows are regarded as short flows in the beginning, because the number of long flows for each ToR switch is very small (i.e., less than 4) and the threshold for large flows in CAPS$_{dark}$ is only 100KB. Thus, CAPS$_{dark}$ achieves almost the same performance as CAPS.

We also find that the short flows in the data mining workload has lower AFCT than those in the web search workload. The reason is that the data mining workload has more obvious boundary between the vast majority of short flows and a few long flows, while in the web search workload there are many medium flows between 100KB and 1MB. These medium flows are not encoded and lead to larger queueing delay.

We test the throughputs of long flows. As shown in Fig. 17 (c) and Fig. 17 (f), RPS obtains the highest throughput, because it spreads the packets of long flows to all paths. Presto also achieves high throughput by scattering the long flows among all parallel paths at flowcell granularity. Since ECMP, RepFlow and Freeway do not effectively utilize all available multiple paths, all of them have lower throughput. The long flows in ECMPpq and RPSpq with priority queue experience throughput degradation due to their large queueing delay. As a packet-level scheme, CAPS flexibly adjusts the number of paths for long flows according to the ON/OFF mode of short flows. When the short flows are in OFF mode, CAPS scatters the packets of long flows to all paths, significantly improving the throughput of long flows compared to the flow-level schemes. Moreover, since the threshold for large flows in CAPS$_{dark}$ is 100KB, which is much smaller than the long flow size, CAPS$_{dark}$ has the similar performance to CAPS.

B. Performances in FatTree

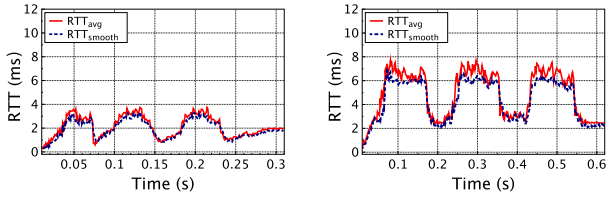
The performance of CAPS under the FatTree topology is evaluated. The number of pods gradually increases from 4 to 12 with fixed 30% traffic load. The number of servers connected to ToR switch under one pod increases from 4 to 36 accordingly. The buffer size is 256 packets. The bandwidth of all the links is set to 1Gbps and the propagation delay among pods is 100μs. In this test, the senders and receivers are randomly selected among pods.



(a) Short flows in web search (b) Long flows in web search

(c) Short flows in data mining (d) Long flows in data mining

Fig. 18. Web search and data mining applications under FatTree.



(a) Poisson parameter λ=5 (b) Poisson parameter λ=30

Fig. 19. Smoothed RTT and average RTT.

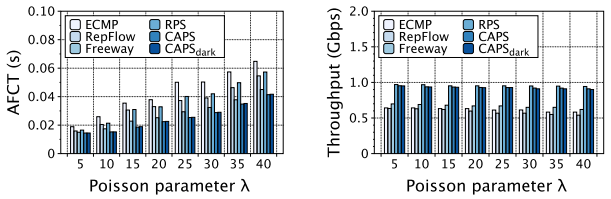
As shown in Fig.18 (a) and Fig.18 (c), the AFCT of short flows for CAPS in both the web search and data mining scenarios under FatTree topology is smaller than other schemes. This is because the encoded packets of CAPS can recover the blocked packets and avoid the negative impact of reordering in a timely manner. In Fig.18 (b) and Fig.18 (d), the throughputs of long flows in CAPS are close to RPS, because CAPS utilizes multiple paths by flexibly spraying packets like RPS when the short flows are in the OFF mode. The performances of long flows in ECMP, RepFlow and Freeway are poor due to the flow-based hashing policy, which may cause flow collisions on some paths while the other paths are not congested.

C. Impact of Network Dynamic

In our design, CAPS calculates the blocking probability according to RTT. The blocking probability varies with the rapid change of the network conditions such as bandwidth and delay when the burst requests from a lot of hosts are generated. We vary the distribution of burst requests to test the accuracy of RTT measurement and CAPS performance under the dynamic network status.

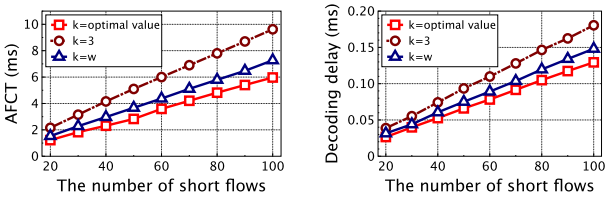
Specifically, in this part, we use the leaf-spine topology consisting of 20 ToR and 20 core switches. There are 28 servers connected to each ToR switch. The bottleneck link bandwidth and buffer size are 1Gbps and 256 packets, respectively. The propagation delay is set to 100μs. Additionally, all short flows with random size less than 100KB and 10 long flows larger than 20MB are generated in heavy-tailed distribution. Each burst of 500 short flows arrives in Poisson distribution. We gradually increase the parameter of Poisson distribution from 5 to 40 to vary the intensity of burst requests.

Fig.19 shows the changes of average RTT RTT_{avg} and smoothed RTT RTT_{smooth}. The value of RTT_{avg} is the average of all instantaneous RTTs in a sampling period. The value



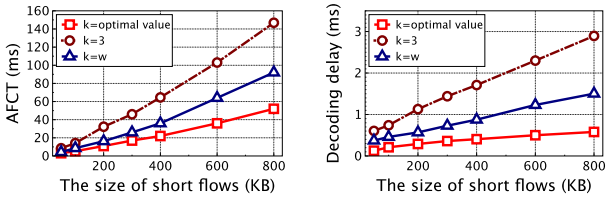
(a) AFCT of short flows (b) Throughput of long flows

Fig. 20. Varying the parameter λ of Poisson distribution.



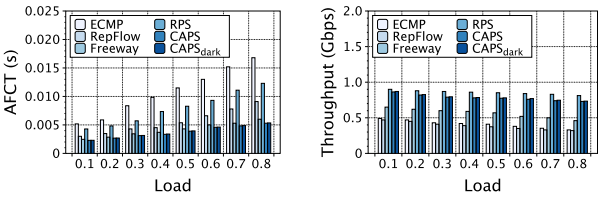
(a) AFCT of short flows (b) Average decoding delay

Fig. 21. Varying the number of short flows.

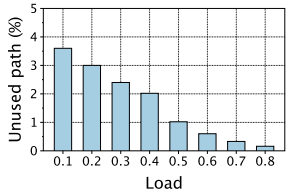


(a) AFCT of short flows (b) Average decoding delay

Fig. 22. Varying the size of short flows.



(a) AFCT of short flows (b) Throughput of long flows



(c) Unutilized paths

Fig. 23. Hybrid applications under fixed large-scale leaf-spine topology.

of RTT_{smooth} is calculated by $\frac{7}{8}RTT_{smooth}+\frac{1}{8}RTT_{avg}$. The sampling period of RTT is twice the propagation delay (i.e. 200μs). As shown in Fig.19 (a) and Fig.19 (b), the value of smoothed RTT is close to that of average RTT. This is because that the instantaneous RTTs on all paths in a sampling period are averaged, reducing the difference between RTT_{avg} and RTT_{smooth}. In CAPS, the blocking probability is the ratio of the number of ACK packets with RTT larger than twice smoothed RTT to the total number of received ACK packets at the sender. Therefore, the blocking probability calculated by smoothed RTT is close to that calculated by instantaneous average RTT. Thus, the blocking probability could capture the accurate congestion states under the rapidly changing network scenario.

We test CAPS performance under different intensities of burst requests. As shown in Fig.20 (a), when the parameter λ of Poisson distribution increases, all schemes experience larger AFCT due to higher burstiness. However, CAPS achieves larger improvement under larger λ compared with the others. This is because that CAPS timely recovers the original packets and mitigates the negative impact of packet loss and reordering caused by the rapid changing of network conditions. Fig.20 (b) shows that the throughputs of long flows decrease a little under larger intensity of burst requests. The reason is that, when the arrival intensity of the short flows increases, the completion times of short flows increase due to heavier congestion. In the longer ON period of short flows, the long flows use ECMP and reduce their throughputs. Nonetheless, since the traffic size of short flows is small compared with long flows, the impact of λ is limited. CAPS obtains the close throughput to RPS.

D. Impact of Coding Unit Size

To evaluate the performance of CAPS with optimal coding unit, we conduct simulations to compare the AFCT and decoding delay per packet with large, small and appropriate coding unit. The simulation settings are as same as that in Section VI-C.

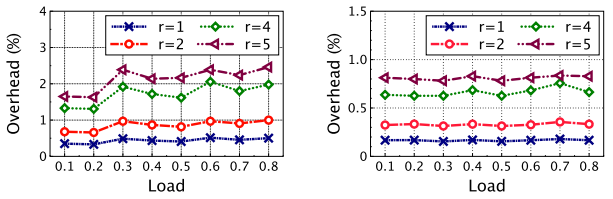
We test the impact of coding unit size on the delay performance of 100KB short flows. As shown in Fig. 21 (a) and Fig. 21 (b), both AFCT and average decoding delay per packet increase with larger number of short flows. Specifically, the AFCT with the optimal k is reduced by up to $\sim$15% and $\sim$40% compared with that of large and small k, respectively. The decoding delay of optimal coding unit is reduced by up to $\sim$10% and $\sim$30% compared to the results with large and small k, respectively. This result shows that CAPS with optimal coding unit improves the decoding probability and thus reduces the decoding delay.

Next, we evaluate the delay performance of 100 flows with different sizes of coding unit. From Fig. 22 (a) and Fig. 22 (b), we observe that the performance gain of the optimal coding unit becomes larger with the increasing flow size, which leads to larger congestion window. The reason is that, with the increase of congestion window, the coding efficiency decreases dramatically when k is set to the congestion window or a minimum value. This means that CAPS with optimal coding unit exhibits more robust scalability and better coding efficiency.

E. System Overhead

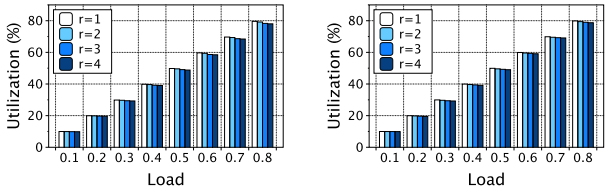
Firstly, we test the CAPS performance under different loads for hybrid applications in large-scale simulations. Meanwhile, we analyze the impact of the accuracy of distinguishing ON and OFF modes of short flows on the throughput of long flows. Specifically, we use the leaf-spine topology consisting of 12 ToR and 20 core switches. There are 28 servers connected to each ToR switch. The bottleneck bandwidth and switch buffer size are 1Gbps and 256 packets, respectively. The propagation delay is set to 100μs. Additionally, we generate mixed traffic for web search and data mining applications with the same features as that in Section IX-A. Each application accounts for 50% of the total traffic.

As shown in Fig.23 (a), CAPS outperforms the other schemes in average FCT since the blocked and lost packets are recovered by the encoded packets. Fig.23 (b) shows that CAPS achieves high throughput of long flows, because the long flows



(a) Web search workload (b) Data mining workload

Fig. 24. Traffic overhead.



(a) Web search workload (b) Data mining workload

Fig. 25. Effective link utilization.

are adaptively switched between ECMP and RPS according to the ON and OFF modes of short flows. In our design CAPS, if the packets of short flows are transferred in a given sampling period, the long flows will use ECMP in the next sampling period. However, due to RPS's random packet spraying, there may exist some paths that are not utilized by short flows during their ON periods under the light load of large-scale networks. Since the long flows use ECMP in the ON periods of short flows to mitigate the impact on short flows, these paths may also be not used by long flows, resulting in loss of network utilization. We measure the average ratios of unused paths by short flows to all paths. As shown in Fig.23 (c), the average ratio of unused paths is less than 5% and decreases with heavier traffic load. This result indicates that, when the traffic load increases, the short flows make better use of multiple paths, resulting in less loss of network utilization.

Secondly, we measure the percentages of extra bytes caused by coding in both workloads scenarios with varying redundancy. In our test, 12 spine switches are connected to each ToR, i.e., there are 12 equivalent paths on each ToR switch. The coding unit size (k) is 10. We set the number of long flows from 1 to 4 according to [30]. Then the corresponding numbers of the redundant packets are 1, 2, 4 and 5, respectively.

As shown in Fig. 24, even if the redundancy is 5 packets, the traffic overheads for both the web search and data mining application scenarios are very small, less than 3% and 1%, respectively. Compared with the overall traffic, such a small amount of overhead can be ignored.

Fig.25 shows the effective link utilization of spine switches. The effective link utilization is the ratio of network goodput to link bandwidth. As shown in Fig.25 (a) and Fig.25 (b), though the effective link utilization deceases a little with larger r, it is almost equal to the load in both the web search and data mining application scenarios. This is because CAPS only encodes short flows, which account for less than 10% of the total traffic [3], [4], [5], [17], and the corresponding redundant traffic overhead is only less than 5% of the overall traffic.

X. Conclusion

To mitigate the negative impact of short and long flows on each other in data center networks, we propose CAPS, a coding-based adaptive packet spraying design that reduces the flow completion time for short flows and guarantees the throughputs for long flows. CAPS utilizes the FEC code to encode only the short flows and spreads the packets of short flows to all equal cost multipath. CAPS limits the long flows when coexisting with the short flows to avoid the head-of-line problem and scatters the packets of long flows to the unused paths by the short flows to achieve high throughput. We evaluate CAPS with both NS2 simulations and small-scale Mininet testbed. The results indicate that CAPS significantly reduces the AFCT by ∼30%-70% for short flows and achieves high throughput for long flows with negligible traffic overhead.

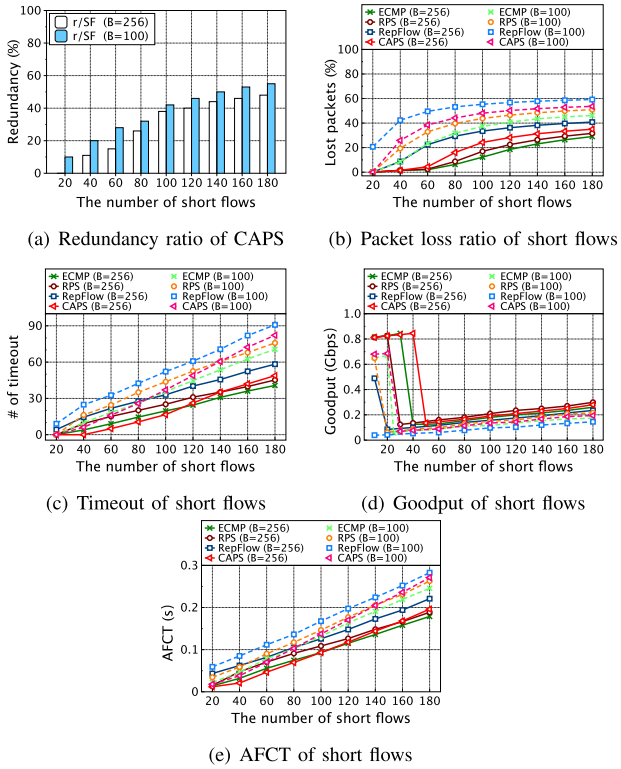
Appendix A
Performances in Incast Scenarios

With more flows or smaller buffer size, the redundant packets of CAPS are more difficult to recover the loss packets. However, since the redundant encoded packets can reduce the timeout probability of short flows especially due to the tail loss (i.e., at least one of the last three packets in the last round are lost), CAPS outperforms the other schemes in increasing the maximum number of supported concurrent short flows without TCP Incast.

We test the performances of five different schemes including ECMP, RPS, RepFlow, Freeway and CAPS under two Incast scenarios. In the first scenario, only the short flows are delivered to a single destination. In the second one, the short flows and 3 background long flows are tested. In these two cases, we use the leaf-spine topology with 40 parallel paths connected with 1Gbps links. The oversubscription ratio on the leaf switch at the destination is 40:1. The buffer size is 256 and 100 packets in the two scenarios, respectively. The other experimental settings are the same as those in Section III. The short flows are transmitted simultaneously from multiple hosts under randomly selected ToR switches to a same receiver. We gradually increase the number of short flows from 20 to 180. The sizes of short flows are uniformly distributed from 15KB to 100KB.

Fig.26 shows the test results of the first case with only short flows. In Fig.26 (a), the ratio of encoded packets to all packets of short flows increases with larger number of short flows, because the blocking probability increases with heavier congestion state. However, CAPS only encodes short flows, which account for less than 10% of the total datacenter traffic [3], [4], [5], [17]. Therefore, the coding redundancy is only less than 6% of the overall traffic. Because of the same reason, although the packet loss ratio of short flows is large as shown in Fig.26 (b), the packet loss ratio to the overall data center traffic is still less than 10%. Moreover, as long as the encoded packets are able to timely recover the blocked ones, the number of timeouts is reduced.

Consequently, as shown in Fig.26 (c), CAPS achieves the smallest number of timeouts until the number of short flows exceeds 100. As the number of flows continues to increase, though CAPS experiences more timeouts due to the negative impact of redundant packets, the performance degradation of CAPS is very small. RepFlow simply replicates all short flows to reap multipath diversity, but increases the traffic overhead and flow concurrency, resulting in the largest number of timeouts. Since RPS randomly sprays all packets to available paths, its flow concurrency is higher than the flow-based schemes, resulting in more timeouts than ECMP and Freeway.

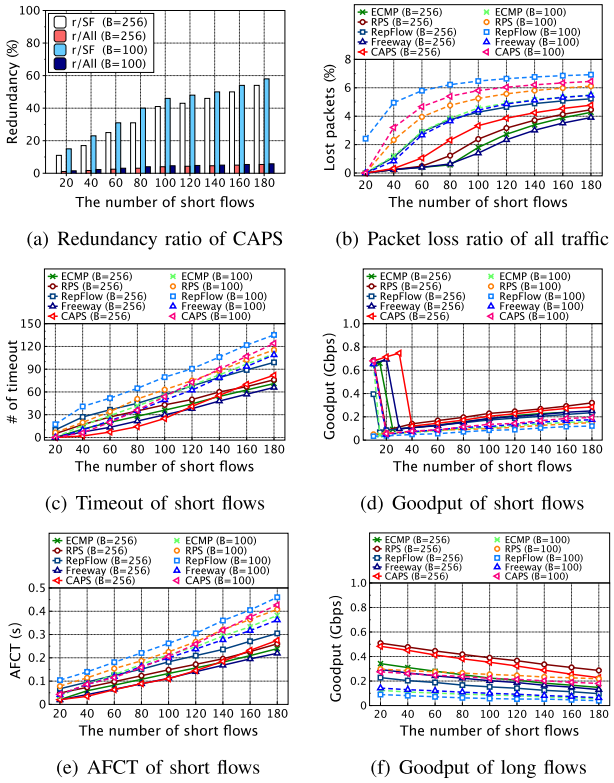


(a) Redundancy ratio of CAPS (b) Packet loss ratio of short flows

(c) Timeout of short flows (d) Goodput of short flows

(e) AFCT of short flows

Fig. 26. The first Incast scenario without long flows.



(a) Redundancy ratio of CAPS (b) Packet loss ratio of all traffic

(c) Timeout of short flows (d) Goodput of short flows

(e) AFCT of short flows (f) Goodput of long flows

Fig. 27. The second Incast scenario with long flows.

Since there is no long flow in this case, Freeway assigns all paths to short flows and thus gets the same performance as ECMP. As the buffer size decreases, the congestion becomes more serious. Thus, the packet loss ratio increases and the timeout occurs earlier than that of the cases with 256 packets buffer in all schemes.

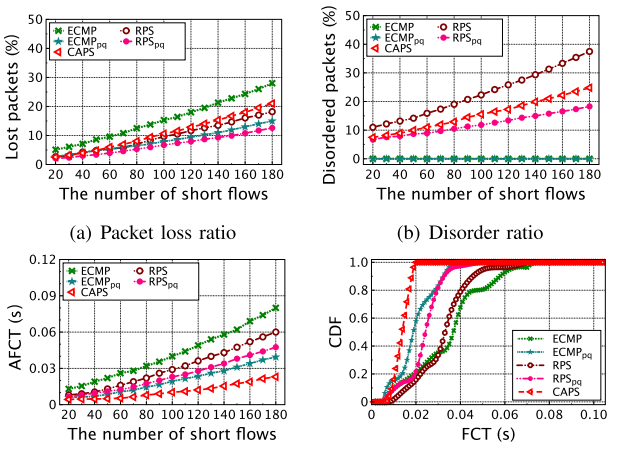
Fig.26 (d) shows the network goodput. CAPS obtains the highest goodput for supporting more flows without Incast. Finally, compared with the other schemes without coding, CAPS effectively reduces the average FCT until the number of short flows exceeds 100 as shown in Fig.26 (e). As the number of flows continues to increase after 100, compared with ECMP and RPS, CAPS suffers from small performance degradation due to traffic overhead.

Fig.27 shows the test results of the second case with both short and long flows. Since the long flows have much more data than short ones, the network congestion is aggravated. Thus, as shown in Fig.27 (a), CAPS has to use more encoded packets to recover the blocked ones and gets higher ratio of redundant packets compared with the results of the first scenario. However, the coding redundancy is only less than 8% of the overall traffic because short flows only provide less than 10% of the total traffic. Fig.27 (b) and Fig.27 (c) show that, though all schemes experience more packet loss and timeouts with the background long flows, CAPS still effectively alleviates the timeout by its coding scheme. Therefore, CAPS achieves the highest goodput as shown in Fig.27 (d). Unlike the first scenario, the flow concurrency of Freeway is lower than that of ECMP in the second scenario due to less paths taken by the short flows in Freeway. Therefore, Freeway performs better than ECMP and close to CAPS, which performs the best until the number of short flows exceeds 120 shown in Fig.27 (e). Fig.27 (f) shows that the throughputs of long flows decrease with larger number of short flows. CAPS obtains lower throughput of long flows than RPS, showing the tradeoff between the delay gain of short flows and the throughput loss of long ones. Compared with the hash-based transmission in ECMP, RepFlow and Freeway, CAPS achieves better performance of long flows because long flows are sprayed to all paths during the OFF periods of short flows.

APPENDIX B
COMPARISON WITH PRIORITY QUEUE SCHEMES

We conduct experiments to compare the performance of ECMP and RPS with and without priority queue, which are denoted by ECMPpq and RPSpq, respectively. In the priority queue, a short flow with high priority is served before a long flow with low priority. Specifically, we use the leaf-spine topology with 12 parallel paths connected with 1Gbps links. The other experimental settings are the same as those in Section III. The mixture of short flows ranging from 15KB to 100KB and 3 background long flows larger than 10MB are generated in heavy-tailed distribution. We gradually increase the number of short flows from 20 to 180 and show the performances of short and long flows in Fig.28 and Fig.29, respectively.

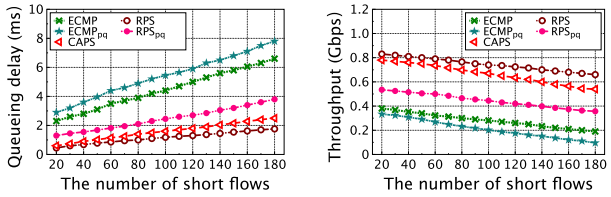
Fig.28 shows the performances of short flows. As shown in Fig.28 (a), the packet loss ratio increases in all schemes with more short flows. Since the priority queue policy ensures high priority for short flows, the packet loss ratios of RPSpq and ECMPpq are significantly lower than that of RPS and ECMP, respectively. In Fig. 28 (b), RPS has the largest ratio of disordered packets, because the long and short flows are sprayed to all paths and mixed together. ECMP and ECMPpq have no disordered packets due to the flow-based transmission. Although RPSpq and CAPS can reduce the impact of long flows on short ones, they still have disordered packets



(a) Packet loss ratio (b) Disorder ratio

(c) AFCT (d) CDF of FCTs with 120 short flows

Fig. 28. The performance of short flow.



(a) Queueing delay (b) Throughput

Fig. 29. The performance of long flow.

due to the packet-based transmission. Fig.28 (c) shows that, compared with the cases without priority queue, the AFCTs of short flows in ECMP and RPS with priority queue are lower. RPS performs poorly due to its high packet disorder ratio. Since using the encoded packets to recover the lost and disordered packets, CAPS achieves the lowest AFCT. We plot the CDFs of FCTs for 120 short flows in Fig.28 (d). The FCTs of CAPS is much smaller than the other schemes.

Fig.29 shows the performances of long flows. In Fig.29 (a), the long flows in ECMPpq and RPSpq suffer from higher queueing delay due to their low priorities. Therefore, as shown in Fig.29 (b), the throughputs of long flows decrease in ECMPpq and RPSpq compared with ECMP and RPS, respectively. In addition, coupled with the hash collisions between long flows, the performance of ECMPpq becomes the worst. For CAPS, the throughput of long flows is close to RPS, since the long flows adaptively spray packets to multiple paths during the OFF periods of short flows. Compared with priority queue policy, CAPS obtains the good tradeoff between the performances of short and long flows.

REFERENCES

[1] A. Dixit, P. Prakash, Y. C. Hu, and R. R. Kompella, "On the impact of packet spraying in data center networks," in *Proc. IEEE INFOCOM*, Apr. 2013, pp. 2130–2138.

[2] Cisco. *Per-Packet Load Balancing*. Accessed: Feb. 28, 2006. [Online]. Available: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/ipswitch_cef/configuration/15-mt/isw-cef-15-mt-book/isw-cef-load-balancing.html#GUID-C725A4B8-9263-4D2C-95FB-F31D14E477C4

[3] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. ACM IMC*, 2010, pp. 267–280.

[4] L. Chen, K. Chen, W. Bai, and M. Alizadeh, "Scheduling mix-flows in commodity datacenters with karuna," in *Proc. ACM SIGCOMM*, 2016, pp. 174–187.

[5] H. Xu and B. Li, "RepFlow: Minimizing flow completion times with replicated flows in data centers," in *Proc. IEEE INFOCOM*, Apr./May 2014, pp. 1581–1589.

[6] W. Wang *et al.*, "Freeway: Adaptively isolating the elephant and mice flows on different transmission paths," in *Proc. ICNP*, Oct. 2014, pp. 362–367.

[7] M. Alizadeh *et al.*, "CONGA: Distributed congestion-aware load balancing for datacenters," in *Proc. ACM SIGCOMM*, 2014, pp. 503–514.

[8] K. He *et al.*, "Presto: Edge-based load balancing for fast datacenter networks," in *Proc. ACM SIGCOMM*, 2015, pp. 465–478.

[9] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. USENIX NSDI*, 2010, pp. 1–15.

[10] C. Raiciu *et al.*, "Improving datacenter performance and robustness with multipath TCP," in *Proc. ACM SIGCOMM*, 2011, pp. 266–277.

[11] M. Kheirkhah, I. Wakeman, and G. Parisis, "MMPTCP: A multipath transport protocol for data centers," in *Proc. IEEE INFOCOM*, Apr. 2016, pp. 1–9.

[12] H. Zhang, J. Zhang, W. Bai, K. Chen, and M. Chowdhury, "Resilient datacenter load balancing in the wild," in *Proc. ACM SIGCOMM*, 2017, pp. 253–266.

[13] Y. Cui, L. Wang, X. Wang, H. Wang, and Y. Wang, "FMTCP: A fountain code-based multipath transmission control protocol," *IEEE/ACM Trans. Netw.*, vol. 23, no. 2, pp. 465–478, Apr. 2015.

[14] A. Munir *et al.*, "Minimizing flow completion times in data centers," in *Proc. IEEE INFOCOM*, Apr. 2013, pp. 2157–2165.

[15] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal, "Fastpass: A centralized 'zero-queue' datacenter network," in *Proc. ACM SIGCOMM*, 2014, pp. 307–318.

[16] G. Chen *et al.*, "Fast and cautious: Leveraging multi-path diversity for transport loss recovery in data centers," in *Proc. USENIX ATC*, 2016, pp. 29–42.

[17] M. Alizadeh *et al.*, "Data center TCP (DCTCP)," in *Proc. ACM SIGCOMM*, 2010, pp. 63–74.

[18] C. Hopps, *Analysis of an Equal-Cost Multi-Path Algorithm*, document RFC 2992, Internet Engineering Task Force, 2000.

[19] C. Jiang, D. Li, and M. Xu, "LTTP: An lt-code based transport protocol for many-to-one communication in data centers," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 1, pp. 52–64, Jan. 2014.

[20] J. K. Sundararajan, D. Shah, M. Medard, M. Mitzenmacher, and J. Barros, "Network coding meets TCP," in *Proc. IEEE INFOCOM*, Apr. 2009, pp. 280–288.

[21] D. Declercq and M. Fossorier, "Decoding algorithms for nonbinary LDPC codes over GF(q)," *IEEE Trans. Commun.*, vol. 55, no. 4, pp. 633–643, Apr. 2007.

[22] M. Luby, "LT codes," in *Proc. IEEE FOCS*, Nov. 2002, pp. 271–280.

[23] O. C. Kwon, Y. Go, Y. Park, and H. Song, "MPMTP: Multipath multimedia transport protocol using systematic raptor codes over wireless networks," *IEEE Trans. Mobile Comput.*, vol. 14, no. 9, pp. 1903–1916, Sep. 2015.

[24] S. Ahmad, R. Hamzaoui, and M. Al-Akaidi, "Adaptive unicast video streaming with rateless codes and feedback," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 20, no. 2, pp. 275–285, Feb. 2010.

[25] J. Hu *et al.*, "CAPS: Coding-based adaptive packet spraying to reduce flow completion time in data center," in *Proc. IEEE INFOCOM*, Apr. 2018, pp. 2294–2302.

[26] N. Dukkipati *et al.*, "An argument for increasing TCP's initial congestion window," *Comput. Commun. Rev.*, vol. 40, no. 3, pp. 26–33, Jul. 2010.

[27] W. Bai *et al.*, "Information-agnostic flow scheduling for commodity data centers," in *Proc. USENIX NSDI*, 2015, pp. 455–468.

[28] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Reproducible network experiments using container-based emulation," in *Proc. ACM CoNEXT*, 2012, pp. 253–264.

[29] A. Khurshid, X. Zou, W. Zhou, M. Caesar, and P. B. Godfrey, "VeriFlow: Verifying network-wide invariants in real time," in *Proc. USENIX NSDI*, 2013, pp. 15–27.

[30] J. Huang, T. He, Y. Huang, and J. Wang, "ARS: Cross-layer adaptive request scheduling to mitigate TCP incast in data center networks," in *Proc. IEEE INFOCOM*, Apr. 2016, pp. 1–9.

[31] A. R. Curtis, W. Kim, and P. Yalagandula, "Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection," in *Proc. IEEE INFOCOM*, Apr. 2011, pp. 1629–1637.

[32] A. Greenberg *et al.*, "VL2: A scalable and flexible data center network," in *Proc. ACM SIGCOMM*, 2009, pp. 51–62.

Jinbin Hu received the B.E. and M.E. degrees from Beijing Jiao Tong University, China, in 2008 and 2011, respectively. She is currently pursuing the Ph.D. degree with the School of Computer Science and Engineering, Central South University, China. Her 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 wireless networks, and data center networks.

Wenjun Lv is currently pursuing the master's degree with the School of Computer Science and Engineering, Central South University, China. His research interests are software defined networks and data center networks.

Yutao Zhou is currently pursuing the Ph.D. degree with the College of Information and Computer Sciences, University of Massachusetts Amherst, USA. Her research interests include congestion control, online strategies, 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.

Tian He (F'18) received the Ph.D. degree from the University of Virginia, Charlottesville, in 2004, under the Supervision of Prof. J. A. Stankovic. He is currently a Professor with the Department of Computer Science and Engineering, University of Minnesota, Twin Cities. He is the author and coauthor of over 200 articles in journals and conferences with more than 20 000 citations (H-Index 59). His publications have been selected as graduate-level course materials by over 50 universities in the United States and other countries. His research includes wireless sensor networks, cyber-physical systems, intelligent transportation systems, real-time embedded systems, and distributed systems, supported by the U.S. National Science Foundation, IBM, Microsoft and other agencies. He is an ACM Fellow. He has received a number of research awards in the area of networking, including five best paper awards. He was a recipient of the NSF CAREER Award 2009 and McKnight Land-Grant Professorship. He served a few program chair positions in international conferences and on many program committees, and also currently serves as an Editorial Board member for six international journals, including the IEEE TRANSACTIONS ON COMPUTERS.