Huang *et al. Journal of Cloud Computing: Advances, Systems and Applications* (2021) 10:31
https://doi.org/10.1186/s13677-021-00248-4

Journal of Cloud Computing: Advances, Systems and Applications

RESEARCH Open Access

HTPC: heterogeneous traffic-aware partition coding for random packet spraying in data center networks

Jiawei Huang[1], Shiqi Wang[1], Shuping Li[1], Shaojun Zou[1,2,3*], Jinbin Hu[4] and Jianxin Wang[1]

Abstract

Modern data center networks typically adopt multi-rooted tree topologies such leaf-spine and fat-tree to provide high bisection bandwidth. Load balancing is critical to achieve low latency and high throughput. Although the per-packet schemes such as Random Packet Spraying (RPS) can achieve high network utilization and near-optimal tail latency in symmetric topologies, they are prone to cause significant packet reordering and degrade the network performance. Moreover, some coding-based schemes are proposed to alleviate the problem of packet reordering and loss. Unfortunately, these schemes ignore the traffic characteristics of data center network and cannot achieve good network performance. In this paper, we propose a Heterogeneous Traffic-aware Partition Coding named HTPC to eliminate the impact of packet reordering and improve the performance of short and long flows. HTPC smoothly adjusts the number of redundant packets based on the multi-path congestion information and the traffic characteristics so that the tailing probability of short flows and the timeout probability of long flows can be reduced. Through a series of large-scale NS2 simulations, we demonstrate that HTPC reduces average flow completion time by up to 60% compared with the state-of-the-art mechanisms.

Keywords: Data center, Packet spraying, Heterogeneous traffic, Network coding

Introduction

With the rapid development of cloud computing and big data, data centers have been used as a critical infrastructure for online services such as recommendation systems, advertisement [1–4], and back-end computations such as MapReduce and GFS [5, 6]. Moreover, the heterogeneous traffic in data center network comes from various applications such as data mining and web search [7, 8]. When these applications are running, a mix of long and short flows will be generated in the network, constituting heterogeneous traffic [9, 10]. Prior work shows that most bytes are contained in a very small number of long flows that demands high throughput while most flows are short and require low latency [11]. Moreover, a small delay seriously degrades the user experience and then affects the financial revenue [11]. For example, Google observes that an extra 500ms of latency causes a 20% traffic reduction, and Amazon reports that every additional 100ms of latency comes at the cost of 1% of business revenue [1, 12].

To improve application performance, various fine-grained load balancing solutions have been proposed to make full use of bandwidth resources. For example, per-packet load balancing schemes such as Random packet Spraying (RPS) [13] split flows across multiple paths at packet granularity and achieve good performance in symmetric topologies. However, they inevitably cause serious packet reordering in asymmetric topologies, degrading performance of network. Moreover, although flowlet-based load schemes can effectively mitigate packet reordering, they suffer from inflexible path switching due to improper threshold of flowlet timeout [14].

*Correspondence: zoushj@csu.edu.cn
[1]School of Computer Science and Engineering, Central South University, 410083 Changsha, China
[2]School of Computer Engineering and Applied Mathematics, Changsha University, 410022 Changsha, China
Full list of author information is available at the end of the article

Fortunately, network coding can effectively mitigate the packet reordering by recovering some unlucky packets that are blocked or dropped on congested paths. With the help of network coding, per-packet load balancing schemes not only can take full advantage of multiple paths between each host pair, but also prevent packet reordering and reduce flow completion time. For instance, both Corrective [15] and CAPS [16] are typically coding-based schemes that the source packets are encoded at the sender and these encoded packets are scattered to multiple paths at the switch. Although some flows experience packet loss or out-of-order, the receiver can immediately recover source packets when it receives the sufficient encoded packets.

However, the existing schemes either only consider the coexistence of long and short flows without network coding [17, 18] or leverage the same strategy to encode packets of long and short flows [15, 16]. That is, these coded-based schemes do not differentiate between short and long flows when they calculate the number of encoded packets. As a result, these solutions obtain suboptimal performance.

In this paper, we propose a Heterogeneous Traffic Partition Coding mechanism HTPC for data center networks to improve overall benefits of short and long flows. To effectively reduce the tailing probability of short flows and the timeout probability of long flows, HTPC dynamically adjusts the number of redundant packets based on the multi-path congestion information and the traffic characteristics. Moreover, the coding layer is deployed between the TCP and IP layers at the end hosts and does not modify the existing TCP/IP protocols. By using NS2 simulations, we demonstrate that HTPC performs remarkably better than the state-of-the-art schemes. Specially, HTPC greatly reduces the 99th percentile flow completion time (FCT) of short flows by ∼65%-83% over RPS under high workload. Meanwhile, HTPC yields up to ∼25% and ∼62% throughput improvement for long flows over Corrective and CAPS, respectively.

In summary, the key contributions of this work are:

- We conduct a series of experiments using NS2 simulator to explore the issues of multi-path transmission solutions: the existing coding-based schemes ignore the characteristic of heterogeneous traffic (i.e., a mix of long and short flows) [10], inevitably degrading the network performance.
- We propose a packet-level transmission mechanism HTPC, which considers the characteristics of heterogeneous traffic in data centers. HTPC smoothly adjusts the number of redundant packets based on the congestion information and the traffic characteristics of heterogeneous traffic, aiming to reduce the tailing probability of short flows and the timeout probability of long flows.

The remainder of this paper is organized as follows. In "Design motivation" section, we discuss our design motivation. In "Desgin overview" section, the overview of HTPC is presented. In "Redundancy adjustment" and "Coding and decoding" sections, we introduce redundancy adjustment and the coding algorithm of HTPC, respectively. In "Model analysis" section, we present a modeling analysis of HTPC. In "Simulation evaluation" section, we evaluate the performance of HTPC through NS2 simulations. In "Related works" section, we summarize the related works. Finally, we conclude the paper in section "Conclusion" section.

Design motivation

In this section, we first introduce the background of multi-path transmission in data center networks. Then we analyze the limitations of existing schemes through extensive tests. Finally, we summarize our observations to motivate our design.

Background

Recently, a substantial amount of research efforts on multi-path transmission mechanisms have been proposed to improve the overall throughput in data center networks. They can be roughly divided into the following three categories: multi-path transmission mechanisms without network coding, multi-path transmission mechanisms with undistinguished coding and multi-path transmission mechanism with distinguishing coding. As shown in Fig. 1, there are 3 flows coexisting on 3 equal-cost paths between switches. Both flow 1 and flow 2 are short flows while flow 3 is long one. The switch uses the RPS load balancing strategy to forward packets.

Figure 1a shows the multi-path transmission mechanism without network coding, whose representative work is RPS. When the packet arrives, the switch randomly picks one available path for each packet to forward. Since RPS forwards packets at packet granularity, it can make full use of the multiple paths. Unfortunately, some packets from flow 1 and flow 3 are scattered to the blocked path. These packets can only be recovered by retransmission. That is, flow 1 and flow 3 at least take one additional round trip time (RTT) to finish data transmission, resulting large flow completion time.

In Fig. 1b, the transmission mechanism with undistinguished coding additionally transmits redundant encoded packet. Although some packets are dropped or blocked on the congested path, the receiver can recover the blocked/lost data when it receives the sufficient encoded packets. This transmission mechanism avoids the retransmission and decreases flow completion time. Corrective

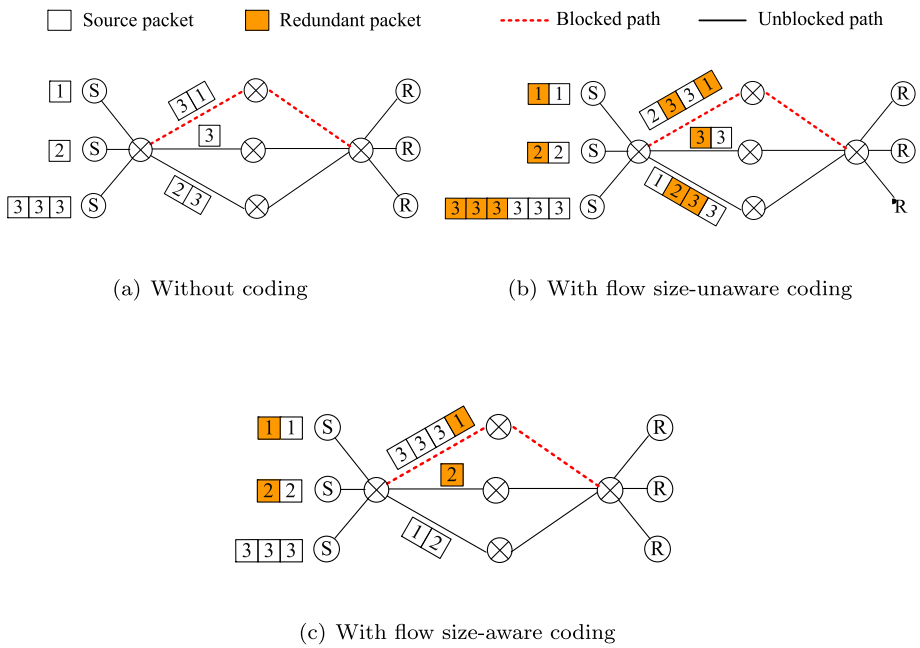


Fig. 1 Three transmission mechanisms. The numbers in the boxes indicate flows' ID

is a representative protocol in this type of transmission mechanism. It implements encoding by performing an XOR operation on all the packet payloads of a congestion window, and adds an encoded data packet for each congestion window. Corrective significantly reduces network latency with low computational and bandwidth overhead.

However, the encoding method does not distinguish between long and short flows, which may inject a large amount of redundant packets into the network. As a consequence, packets of short flows suffer from large queuing delay and increase the flow completion time. Besides, corrective can only recover a single packet loss within a window. If more than one packet is dropped, the sender has to transmit the lost packets via the fast or timeout retransmission. That is, Corrective hardly adapts to the dynamic traffic with the fixed number of encoded packets [19].

The multi-path transmission mechanism with distinguishing coding considers the characteristics of heterogeneous traffic in data center network. CAPS is a typical representative of this type of transmission mechanism. The sender only encodes short flows and the switch adopts an adaptive load balancing scheme to forward packets. Specially, the switch uses the ECMP [20] method to forward packets of the long flows when the long flows coexist with the short flows while leveraging RPS mechanism to forward data of the long flows when there are no short flows.

However, Fig. 1c shows that the switch leverages the ECMP mechanism to forward the packets of long-live flow 3. When long flow 3 is hashed to the blocked path and cannot be switched in time, it inevitably results in low throughput.

The limitations of existing solutions

To intuitively illustrate the above problems, we take RPS, Corrective and CAPS as the representatives of the above three transmission mechanisms, and then conduct a series of experiments to test their performances. In this test, we adopt the Leaf-Spine network topology with 4 spine switches and 4 leaf switches, as shown in Fig. 2. Each

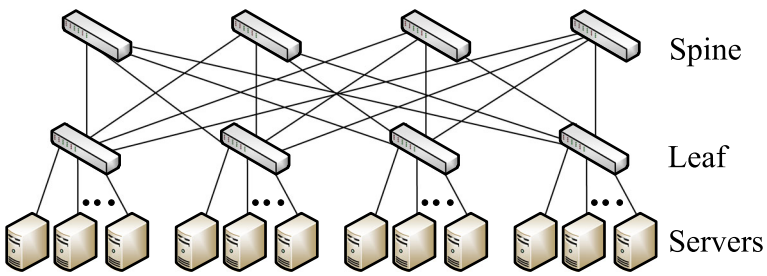


Fig. 2 Leaf-Spine topology

leaf switch is connected to 16 hosts. All links have 1Gbps bandwidth and 100μs latency. The buffer size of per port is 256 packets. Here, we use two realistic workloads (i.e., the Web Search and Data Mining [21]) measured from deployed data centers to evaluate their performance. The experimental results of the three transmission mechanisms are shown in Fig. 3.

Figure 3a shows that in the Web Search and the Data Mining scenario, RPS without coding strategy has the longest short flow completion time, followed by Corrective, and CAPS achieves the best performance of short flows. For RPS, the short flows obtain low performance due to packet loss. Besides, since corrective can recover a single data packet loss through a single redundant coded packet in the window, the performance of short flow is improved slightly. CAPS adopts coding and adaptively load balancing scheme, greatly reducing the flow completion time of short flows.

Figure 3b shows that the throughput of long flows in CAPS is the lowest. This is because that only short flows are encoded, and its load balancing scheme prioritizes short flows, which impairs the performance of long flows. Since Corrective uses the same coding strategy for long and short flows, Corrective's long flows perform better. RPS suffers from packet loss and causes congestion window reduction so that long flows obtain lower throughput than other protocols.

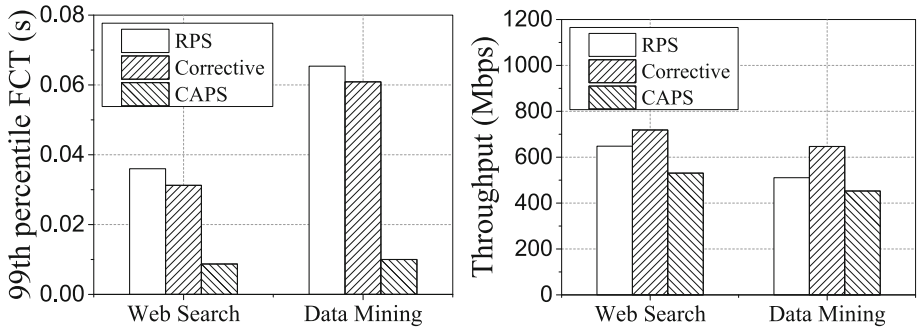
Summary

In summary, the current multi-path transmission mechanisms have the following limitations: (i) the transmission mechanism without network coding cannot recover the packet loss, so both short and long flows are impacted by the path asymmetry; (ii) For the coding strategy without distinguishing heterogeneous traffic, long and short flows use the same coding mechanism and long flows may generate lots of redundant packets, which inevitably increase the queuing delay and short flows' flow completion time; (iii) the current encoding strategy that distinguishes heterogeneous traffic uses encoding technology only for short flows to compensate for packet loss, and ignores long flows, thereby reducing the throughput of long flows. These conclusions motivate us to tackle above problems by designing and implementing a Heterogeneous Traffic-aware Partition Coding mechanism.

Desgin overview

In this section, we present an overview of HTPC. The key point of HTPC design is that the sender adopts a heterogeneous traffic-aware partition coding to improve the overall network performance. On the one hand, redundant encoded packets can recover the blocked and lost packet. On the other hand, based on the characteristics of the heterogeneous traffic, the sender smoothly adjusts the number of redundant packets so that both the tailing probability of short flows and the timeout probability of long flows can be reduced. The architecture of HTPC consists of three modules, as shown in Fig. 4.

(1) **Encoding Module at the sender:** In our design, we implement an encoding layer between the IP and transport layer. The main reason is that it does not modify the existing TCP/IP protocols, which can keep compatibility with existing protocols and is ease of practical deployment. In encoding layer, we implement network status estimation, heterogeneous flow identification as well as redundancy adjustment. Specifically, the packets delivered from the transport layer to IP layer are stored in the encoding buffer for coding while the control packets (i.e., SYN, FIN and ACKs) are directly sent to the IP layer without coding. During data transmission process, the sender estimates the network status through RTTs and distinguishes heterogeneous flows based on the number of data bytes sent. According to the discrimination result and the network congestion information, the sender smoothly adjusts the coding parameters of long and short flows, reducing the tailing probability of short flows and the timeout probability of long flows.



(a) 99th percentile FCT of Short flows (b) Throughput of Long flows

Fig. 3 Network performance

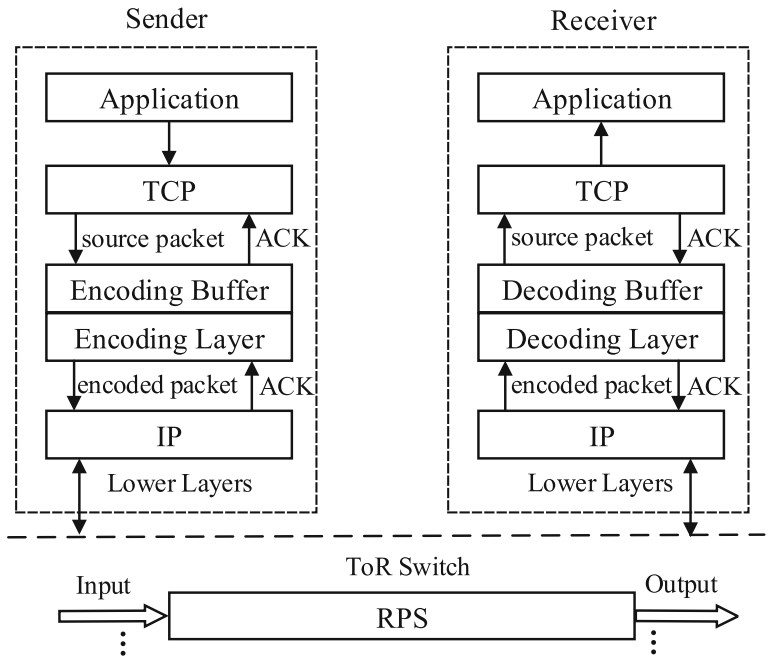


Fig. 4 HTPC architecture

Note that the key drawbacks of encoding at other layers is that it inevitably needs to modify the control logic at corresponding layer and increases its complexity. What is worse, it may interfere with the original control logic.

(2) **Packet Spraying Module at switch:** To make full use of the bandwidth resources, the switch adopts the RPS load balancing scheme to forward packets, which have already been implemented in many commodity switches [16]. Specially, when a packet arrives at the switch, the forwarding engine randomly selects one available path for the packet.

(3) **Decoding Module at the receiver:** Upon receiving one packet, the receiver stores the packet in the decoding buffer and determines whether the packet is a redundant coded packet by a coding flag bit in packet header. If the packet is source packet, it is submitted directly to the upper layer without decoding. If the packet is the encoded packet and there are enough source and encoded packets, the receiver uses Gaussian elimination to decode so that the blocked or lost packets can be fast recovered and delivered to the upper layer.

Redundancy adjustment

To better adjust the redundancy of coding mechanism, HTPC leverages a measurement module to periodically calculate the real-time packet loss rate at sender so that it can effectively cope with the rapid change of dynamic network. The main variables used in the paper are shown in Table 1.

Table 1 Variables in HTPC

Variable	Description	Variable	Description
T	sampling period	PL	Packet loss rate
n_t	Number of packets transmitted during T	w	Weight of each flow
n_d	Number of packets loss during T	D	Number of bytes sent by the flow
T_s	Threshold of short flows	k	Size of the encoding unit
T_w	Threshold of weight adjustment	w_c	Congestion window size
x	Number of packets expected to be received by receiver	n_b	number of ACK whose RTT is greater than 2x the average RTT
r	Number of redundant encoded packets	n	Total number of ACKs
G	a random linear independent coefficient matrix	s	Identity matrix of the original data packet in a coding unit
c	Corresponding generating matrix	T_{fr}	Threshold of fast retransmission
PB_{RPS}	Packet blocking rate of RPS	PTO_{RPS}	timeout probability of RPS
PB_{Cor}	Packet blocking rate of Corrective	PTO_{Cor}	timeout probability of Corrective
PB_{CA}	Packet blocking rate of CAPS	PTO_{CA}	timeout probability of CAPS
PB_{HT}	Packet blocking rate of HTPC	PTO_{HT}	timeout probability of HTPC

Let n_t and n_d denote the total number of transmitted packets and the number of packet loss during sampling period T, respectively. Then we can calculate the real-time packet loss rate PL as

$$PL = \frac{n_d}{n_t}. \tag{1}$$

In our design, the sampling period T is set as 500μs. This is because that the time interval of packet burst is generally larger than 500μs, and micro-burst is the main cause of packet loss [22–24].

It is a common traffic pattern that long and short flows coexist in data center network. Short flows require low latency while long flows demands high throughput. HTPC allocates the weight for each flow according to the number of transmitted bytes and adjusts its redundancy accordingly. The weight of each flow gradually decreases as it transmits more packets.

Inspired by L^{2}DCT [11], we use D, T_s and T_w to denote the number of bytes sent by each flow, the thresholds of short flows and weight adjustment respectively, and then calculate the weight w of each flow as

$$w = \begin{cases} 1, & \text{if } D < T_s \\ 1 - \frac{D-T_s}{T_w-T_s}, & \text{if } T_s \le D \le T_w \\ 0, & \text{if } D > T_w, \end{cases} \tag{2}$$

When the sender detects packet loss, it has to retransmit the lost packet, which at least increases one RTT to finish data transmission. Since short flows are delay-sensitive and requires low delay, one additional RTT inevitably leads to a sharp decline in network performance. For long flows, when TCP timeout happens, the sender has to wait the minimum retransmission timeout (RTO_{min}, whose default value is 200ms in most operating systems), greatly degrading the throughput of long flows. Fortunately, when the number of packets received by each long flow's receiver is not less than the threshold of fast retransmission (its default value is 3), the sender triggers the fast retransmission mechanism that retransmits lost packets without expensive timeout. Therefore, long flows can avoid long waiting and achieve high throughput.

Note that the RTO_{min} can be adjusted to 10ms in data center. However, this method may cause safety problems such as spurious retransmission [25]. Once the sender triggers spurious retransmission due to timeout, it will enter the slow-start phase and set its congestion window to the minimum value, inevitably resulting in the waste of available bandwidth. A detailed discussion of verifying the proposal under different RTO_{min} is left for future work.

To reduce the packet retransmission of short flows and the timeout of long flows, the lower and upper bounds of the number of packets expected to be successfully received by the receiver are set to the threshold of fast retransmission T_{fr} (its default value is 3) and the congestion window size w_c, respectively. Besides, let k denotes the size of the encoding unit, whose value is set to the congestion window size (i.e. w_c). Then the number of packets x expected to be received successfully can be calculated as

$$x = w \times k + (1 - w) \times T_{fr}. \tag{3}$$

We assume that a coding unit has r redundant encoded packets. It means that the sender will transmit k source packets and r redundant encoded packets. To achieve high decoding probability, the number of redundant encoded packets should satisfy $(k + r) \times \left(1 - \frac{n_b}{n}\right) \ge k$, where n is the total number of ACKs and n_b is the number of ACKs whose RTTs are greater than 2x the average RTT. Moreover, to avoid unnecessary bandwidth waste, we can obtain the optimal number of redundant encoded packets r as

$$r = \frac{x}{1 - \frac{n_b}{n}} - k = \frac{w \times k + (1 - w) \times T_{fr}}{1 - \frac{n_b}{n}} - k, \tag{4}$$

With Eqs. (2) and (4), the number of redundant encoded packets r can be calculated as

$$r = \begin{cases} \left(\frac{1}{1-\frac{n_b}{n}} - 1\right) \times k, & \text{if } D < T_s \\ \frac{(T_{fr}-k)\times\frac{D-T_s}{T_w-T_s}+k}{1-\frac{n_b}{n}} - k, & \text{if } T_s \le D \le T_w \\ \frac{T_{fr}}{1-\frac{n_b}{n}} - k, & \text{if } D > T_w. \end{cases} \tag{5}$$

Coding and decoding

In this section, we will introduce how HTPC implements its coding and decoding mechanisms at the end hosts.

In our design, we uses the system coding due to its simpleness and effectivity [26]. Specially, the sender initially transmits source packets within congestion window and then generates proper encoded packets according to the network status. When enough source and encoded packets arrive at the receiver, the receiver recovers the lost or blocked packets on congested path by decoding operation.

In coding, the data of each flow is divided into several data blocks. The data block size (i.e., coding unit size) is denoted by k. In each data block, several source data packets are encoded, and the redundant encoded packets are a random linear combination of these original data packets. Specifically, assuming that each data packet has the same size *pktsize* and can be represented as a symbol of size. By using a random matrix to multiply data packets of the same coding unit, an encoded data packet can be generated. Furthermore, we assuming that G is a random linear independent coefficient (RLIC) matrix and s is the identity matrix of the original data packet in a coding unit. Then the corresponding generating matrix c can be calculated as

$$c = s \times G. \tag{6}$$

Then we have

$$\begin{bmatrix} c_1 & c_2 & \cdots & c_m \end{bmatrix} = \begin{bmatrix} s_1 & s_2 & \cdots & s_k \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & g_{1,k+1} & g_{1,k+2} & \cdots & g_{1,m} \\ 0 & 1 & 0 & 0 & \cdots & 0 & g_{2,k+1} & g_{2,k+2} & \cdots & g_{2,m} \\ 0 & 0 & 1 & 0 & \cdots & 0 & g_{3,k+1} & g_{3,k+2} & \cdots & g_{3,m} \\ 0 & 0 & 0 & 1 & \cdots & 0 & g_{4,k+1} & g_{4,k+2} & \cdots & g_{4,m} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \\ 0 & 0 & 0 & 0 & \cdots & 1 & g_{k,k+1} & g_{k,k+2} & \cdots & g_{k,m} \end{bmatrix}, \quad (7)$$

where $\begin{bmatrix} c_1 & c_2 & \cdots & c_m \end{bmatrix}$ is the matrix composed of m encoded packets, $\begin{bmatrix} s_1 & s_2 & \cdots & s_k \end{bmatrix}$ is the matrix composed of k source packets, $G = (g_{i,j})_{k \times m}$ is the $k \times m$ matrix.

Using system coding greatly reduces the receiver's decoding delay, the CPU usage as well as memory occupancy rate. Specifically, literature [27] proves that in the worst case, system coding can reduces the decoding delay of non-system coding by at least 50% in theory. Moreover, compared with non-system coding, the arithmetic complexity of generating encoded packets in system coding can be reduced by 1 order of magnitude while the arithmetic complexity of decoding a generation is reduced by 2 or even 3 orders of magnitude [27], resulting in low the CPU usage and memory occupancy rate. At the receiver, the received packets are restored in the decoding buffer to recover lost and blocked packets. To speed up data transmission, the source packets are also directly delivered to application layer. Besides, the receiver uses Gaussian elimination to decode. Once enough source and encoded packets are received, the lost or blocked packets are recovered and delivered to the upper layer. The decoding equation in term of matrix can be expressed as

$$\begin{bmatrix} s_1 & s_2 & \cdots & s_k \end{bmatrix} = \begin{bmatrix} c_{a1} & c_{a2} & \cdots & c_{ak} \end{bmatrix} \times \begin{bmatrix} g_{1,a1} & g_{1,a2} & g_{1,a3} & \cdots & g_{1,ak} \\ g_{2,a1} & g_{2,a2} & g_{2,a3} & \cdots & g_{2,ak} \\ g_{3,a1} & g_{3,a2} & g_{3,a3} & \cdots & g_{3,ak} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ g_{k,a1} & g_{k,a2} & g_{k,a3} & \cdots & g_{k,ak} \end{bmatrix}, \quad (8)$$

where $\begin{bmatrix} c_{a1} & c_{a2} & \cdots & c_{ak} \end{bmatrix}$ is the matrix composed of any k encoded packets received by the receiver from m coded packets, $g_{i,aj}$ is the ith element of column aj in the matrix G.

Model analysis

In this section, we theoretically analyze the performance of two representative transmission mechanisms (i.e., RPS, Corrective and CAPS) and our design HTPC. Then we test their performance through NS2 simulation experiments.

(i) **RPS**

First, we analyze the packet loss rate of RPS, which doesn't apply network coding. As in Eq. (4), n_b is the number of ACKs whose RTT is greater than 2x the average RTT and n is the total number of ACKs. The packet blocking rate PB_{RPS} of RPS can be calculated as

$$PB_{RPS} = \frac{n_b}{n}. \quad (9)$$

When packet blocking happens and the sender cannot trigger the fast retransmission mechanism due to lacking of enough duplicate ACKs, the sender can not transmit new data until the timeout retransmission timer expires. Therefore, the timeout probability can be obtained by calculating the probability that the number of successfully received packets is less than the fast retransmission threshold T_{fr}. Then the timeout probability PTO_{RPS} of RPS can be calculated as

$$PTO_{RPS} = \sum_{i=0}^{i<T_{fr}} C_{w_c}^{i} \times PB_{RPS}^{w_c - i} \times (1 - PB_{RPS})^{i}, \quad (10)$$

where w_c is the size of congestion window.

(ii) **Corrective**

Next, we analyze the packet loss rate of Corrective. No matter whether it's a long flow or a short flow, Corrective additionally transmits one redundant encoded packet for each congestion window. When the number of source packets is k, the total number of packets injected into the network by Corrective is $k + 1$. The packet blocking rate PB_{Cor} can be computed as

$$PB_{Cor} = \frac{\sum_{i=1}^{k+1} C_{k+1}^{i} \times \left(\frac{n_b}{n}\right)^{i} \times \left(1 - \frac{n_b}{n}\right)^{k+1-i} \times (i-1)}{k}. \quad (11)$$

Then we can obtain the timeout probability PTO_{Cor} as

$$PTO_{Cor} = \sum_{i=0}^{i<T_{fr}} C_{k+1}^{i} \times PB_{RPS}^{k+1-i} \times (1 - PB_{RPS})^{i}. \quad (12)$$

(iii) **CAPS**

CAPS distinguishes between long and short flows. In CAPS, short flows generate several redundant encoded packets according to the blocked probability and the switch adopts RPS mechanism to forward the packets of short flows. For long flows, the switch uses an adaptive load balancing scheme to forward packets. According to the encoding strategy of CAPS, the number of redundant encoding packets is

$$r_{CA} = \begin{cases} \frac{k}{1 - \frac{n_b}{n}} - k, & \text{if } D < T_s \\ 0, & \text{if } D \geq T_s. \end{cases} \quad (13)$$

Obviously, the number of packets injected into the network by CAPS is $k + r_{CA}$. Then we can calculate the CAPS's packet blocking rate PB_{CA} as

$$PB_{CA} = \frac{\sum_{i=1}^{k+r_{CA}} C_{k+r_{CA}}^{i} \times \left(\frac{n_b}{n}\right)^{i} \times \left(1-\frac{n_b}{n}\right)^{k+r_{CA}-i} \times (i-1)}{k}. \tag{14}$$

According to the Eq. (14), the timeout probability PTO_{CA} of CAPS can be calculated as

$$PTO_{CA} = \sum_{i=0}^{i<T_{fr}} C_{k+r_{CA}}^{i} \times PB_{CA}^{k+r_{CA}-i} \times (1-PB_{CA})^{i}. \tag{15}$$

(iv) **HTPC**

Finally, we analyze the packet blocking rate and timeout probability of HTPC. We assume that the number of redundant encoded packets of HTPC is r_{HT}, whose value can be calculated from Eq. 5. We can obtain the packet blocking rate PB_{HT} of HTPC as

$$PB_{HT} = \frac{\sum_{i=1}^{k+r_{HT}} C_{k+r_{HT}}^{i} \times \left(\frac{n_b}{n}\right)^{i} \times \left(1-\frac{n_b}{n}\right)^{k+r_{HT}-i} \times (i-1)}{k}. \tag{16}$$

Then we can calculate the timeout probability PTO_{HT} of HTPC as

$$PTO_{HT} = \sum_{i=0}^{i<T_{fr}} C_{k+r_{HT}}^{i} \times PB_{HT}^{k+r_{CA}-i} \times (1-PB_{HT})^{i}. \tag{17}$$

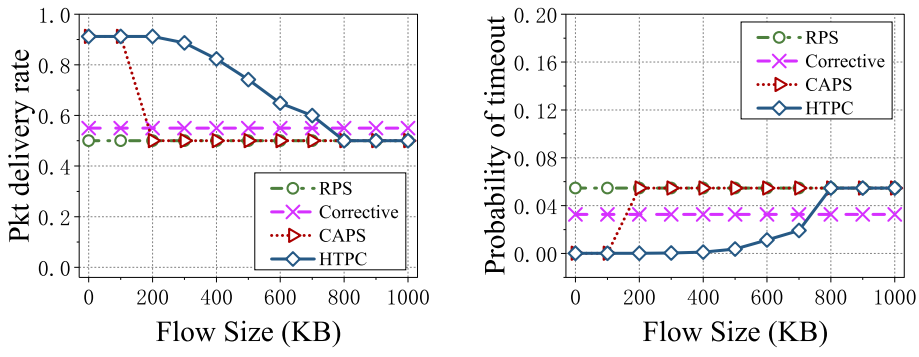
Finally, we compare the successful packet delivery rate and timeout probability of RPS, Corrective, CAPS and HTPC, as shown in Fig. 5. The successful delivery rate of data packets refers to the ratio of the number of data packets successfully received by the receiver to the number of source data packets. Here, we suppose the packet blocking rate of link is 0.5.

Figure 5a shows that compared with RPS and Corrective, the packet delivery rate of short flows in HTPC has been greatly improved. The major reason is that HTPC recovers packet loss by encoded packets. From Fig. 5b, we observe that HTPC greatly reduces the timeout probability of long flows. Since the long flows of RPS and CAPS don't adopt network coding, the timeout probability of long flows can not be improved. Besides, Corrective slightly improves the performance of long flows as the sender only generate one redundant encoded packet in each congestion window. For HTPC, the long flows dynamically generate proper encoded packets according to its timeout probability, which helps reduce the timeout probability.

Simulation evaluation

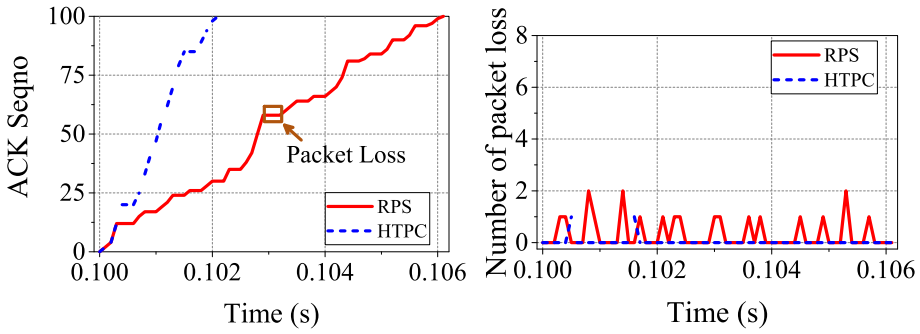
In this section, we conduct large-scale NS2 simulations to evaluate HTPC's performance. In the test, we adopt a 256-host leaf-spine topology with 8 leaf switches and 8 spine switches. That is, there are 8 equal cost paths between any pair of hosts. Each leaf switch connects to 32 servers. Besides, link capacity, round trip propagation delay, and buffer size of switches are 1Gbps, 100μs and 100 packets, respectively.

We compare HTPC with ECMP, RPS, Corrective and CAPS. Specifically, ECMP is the standard load balancing mechanism and it allocates a static path for each flow based on the five-tuple information in packet header. RPS randomly selects one available path to forward each date packet. Corrective transmits one additional redundant encoded packet for each congestion window. CAPS leverages the forward error correction (FEC) coding for short flows to handle the out-of-order problem. Note that the network coding used in HTPC is different from the one in CAPS. The main reason is that HTPC adopts system coding, which generates the expected number of source packets and serval encoded packets via RLIC. However, LDPC used in CAPS does not guarantee that it can generate the expected number of source packets. Therefore,



(a) Packet delivery rate (b) probability of timeout

Fig. 5 Model performance



(a) ACK Seqno (b) Number of packet loss

Fig. 6 The basic performance of short flows

we choose RLIC in HTPC. Moreover, the short flow is the one with flow size less than 100KB and the size of each long flow is more than or equal to 100KB [16].

Basic performance

To validate whether HTPC can improve the performance of long and short flows, we conduct the micro experiments with RPS and HTPC. In this test, there are 5 long flows and 40 short flows. Since the total number of both long and short flows is small (only 45), these flows are randomly distributed under 4 Leaf switches and coexist on 4 multiple paths, which helps to increase the probability that long and short flows compete for bandwidth resources on the paths. Here, we measure the real-time ACK sequence number of short flows and timeout probability of long flows. Thereinto, we obtain ACKSeqno and timeout event of each flow from each end host and calculate their average values.

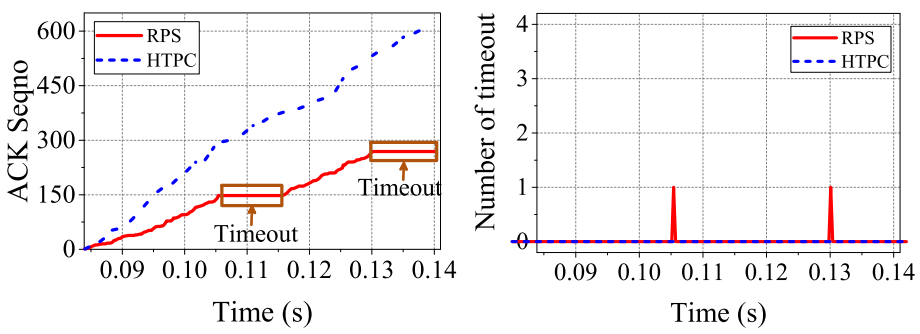
Figure 6 shows the ACK seqno and packet loss amount of short flows. In Fig. 6a, when the sequence number of ACK remains unchanged, it means that some packets are dropped due to buffer overflow. From Fig. 6b, we observer that short flows of RPS has more packet loss than that of HTPC. This is because that HTPC uses redundant coding packets to recover lost packets so that the sender does not sense the packet loss.

Figure 7 shows that long flows of RPS has experienced two timeout events, which inevitably degrades network performance. Fortunately, HTPC dynamically adjusts the number of redundant encoded packets according to network status. Since these redundant encoded packets can recover the lost packets, the long flow of HTPC does not suffer from timeout.

Performance under different flow proportions

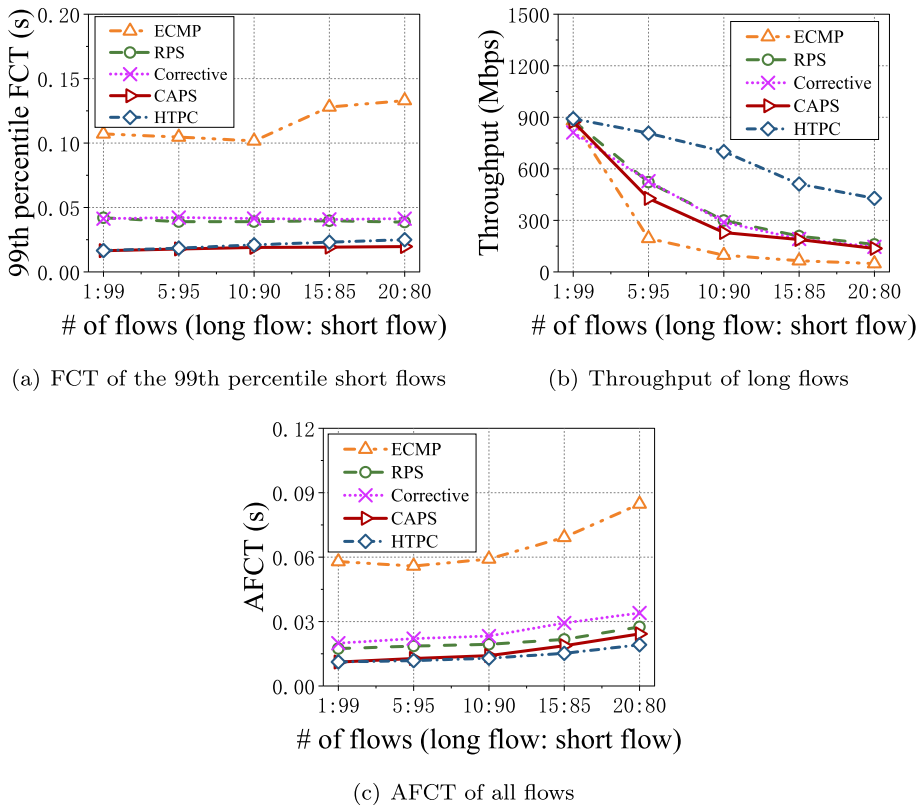
In this section, we evaluate the performance of HTPC under different flow proportions. In this scenario, the total number of long and short flows is 100. We vary the number of long flows from 1 to 20. That is, the ratios of long flow amount to short ones are set to 1:99, 5:95, 10:90, 15:85 and 20:80, respectively.

As shown in Fig. 8a, HTPC obtains lower 99th flow completion time of short flows than that of ECMP, RPS and Corrective. This is because HTPC uses coding to recover the lost and blocked packets of short flows, reducing the packet blocking rate of short flows. ECMP and



(a) ACK seqno (b) Number of timeout events

Fig. 7 The basic performance of long flows



(a) FCT of the 99th percentile short flows

(b) Throughput of long flows

(c) AFCT of all flows

Fig. 8 Performance under different flow proportions

RPS do not distinguish between long and short flows, resulting in large queueing delay for short flows and tailing flow completion time. Corrective does not consider the real-time network status and transmits fixed number of redundant encoded packets without distinguishing long and short flows so that the improved performance is limited.

Figure 8b shows that HTPC obtains the highest throughput for long flows. The main reason is that ECMP, RPS and CAPS do not generate redundant encoded packets for long flows while the long flows in HTPC generate several encoded packets with low redundancy, which can greatly reduce the number of timeouts and achieve high throughput for long flows. Overall, HTPC obtains lower average flow completion time (AFCT) of all flows than other protocols, as shown in Fig. 8c.

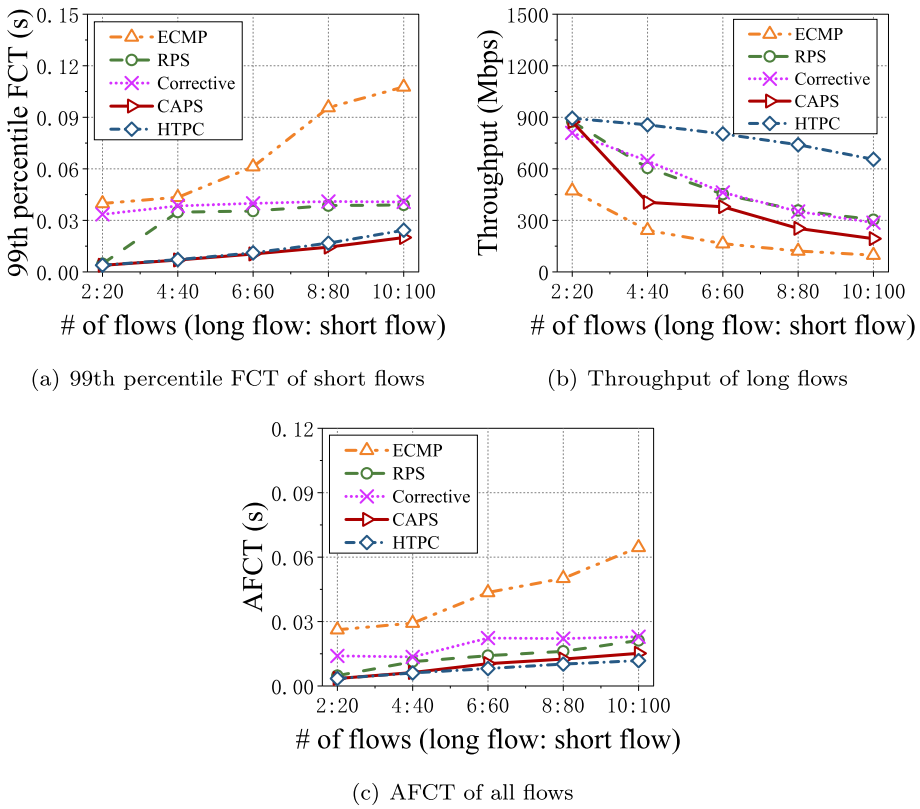
Performance under different numbers of flows

In this section, we evaluate the performance of HTPC under different numbers of flows. In this experiment, the ratio of the number of long flows to short ones is fixed (i.e., 0.1) and we vary the number of short flows from 20 to 100. The experimental results are shown in Fig. 9.

Figure 9a and c show that, with the increasing of flow amount, all protocols take more time to finish data transmission and the throughput of long flows gradually decrease, as shown in Fig. 9b. This is because that the larger the number of flows, the more flows compete. Therefore, each flow gets less bandwidth resources and takes more time to transmit data. Fortunately, HTPC obtains lower the FCT of short flows and higher throughput of long flows than other schemes. The main reason is that HTPC adopts the coding technology to generate several encoded packets for long and short flows according to the network status, which can fast recover the lost packets of short flows and greatly reduce the number of timeouts. This helps reduce the average FCT of short flows and achieve high throughput for long flows.

Performance under realistic workloads

To evaluate HTPC's broad applicability and effectiveness in realistic scenarios, we conduct our tests using realistic workloads measured from deployed data centers [28, 29]. In this test, we adopt two typical realistic workloads: web search [3, 30, 31] and data mining [8, 32, 33]. Both of their traffic distributions have the following characteristic: a small fraction of long flows contribute most of the traffic(i.e., heavy-tailed distribution). Specially, in web search workload, 30% of flows larger than 1MB provide more than 95% bytes. In data mining workload, $\sim$ 3.6% flows



(a) 99th percentile FCT of short flows

(b) Throughput of long flows

(c) AFCT of all flows

Fig. 9 Performance under different numbers of flows

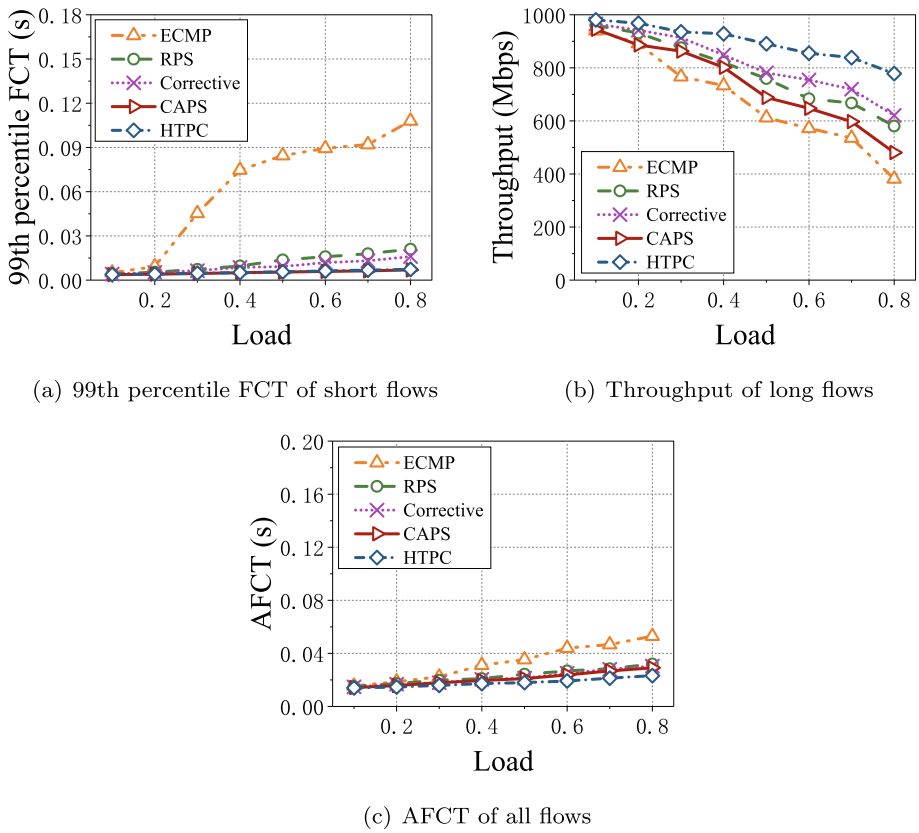
larger than 35MB provide 95% bytes [34], while around 80% of flows are less than 100KB. Table 2 shows the flow size distribution under data mining and web search workloads. Besides, the source and destination of each flow are chosen uniformly random from all hosts while the arrival times of all flows follow a Poisson process. Here, we measure the average FCT of all flows, 99th percentile FCT of short flows (<100KB) as well as the average FCT of long flows (≥100KB) under different network loads. The load can be calculated by $\frac{\lambda \times E}{C}$, where λ, E and C denote the flow arrival rate, the average flow size and link capacity, respectively. In our experiments, we vary the load by changing the flow arrival rate and draw the test results of web search and data mining in Figs. 10 and 11, respectively.

Table 2 Flow size distribution under data mining and web search workloads

Flow size	Data Mining	Web Search
0-10KB	78%	59%
10KB-100KB	5%	3%
100KB-1MB	8%	18%
>1MB	9%	20%

Figures 10a and 11a show that compared with ECMP, RPS and Corrective, HTPC significantly reduces the 99th percentile flow completion time of short flows, especially in high load. Figures 10b and 11b show the throughput of long flows. HTPC obtains the highest throughput of long flows. This is because that HTPC adopts the multi-path transmission method with network coding so that the long flows can avoid timeout and improve long flows' throughput. RPS and Corrective achieve higher throughput than ECMP, which gets the lowest throughput of long flows due to hash collision. Besides, since CAPS does not encode packets for long flows, some long flows may suffer timeout and obtain lower throughput than HTPC.

Figures 10c and 11c show that since ECMP and RPS cannot recover packet loss due to lacking of redundant encoded packet, they can only recover lost packets through the fast or timeout retransmission, resulting in larger flow completion time. Corrective's coding strategy is fixed and cannot adapt to dynamically changing networks. CAPS only encodes for short flows and ignores the performance of long flows. HTPC leverages an adaptive encoding strategy to recover the lost packets, which helps reduce the number of short flow retransmissions and timeouts of long flow. As a result, HTPC obtains lower FCT of all flows than the other schemes.



(a) 99th percentile FCT of short flows

(b) Throughput of long flows

(c) AFCT of all flows

Fig. 10 Web search

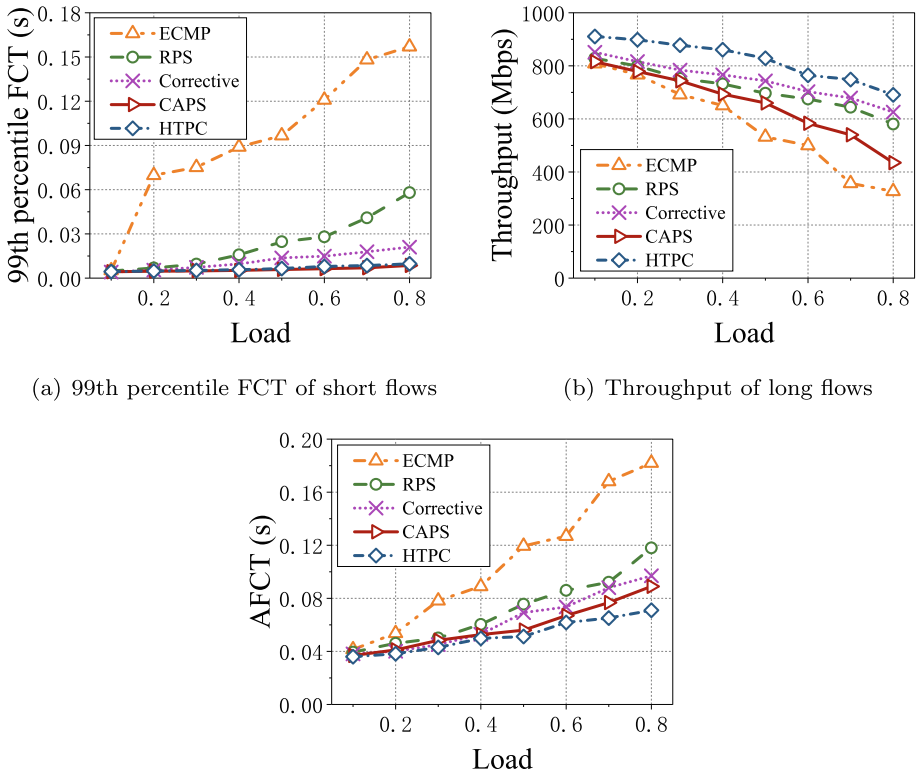
Performance under asymmetric topology and different network scales

In this section, we first adopt a leaf-spine topology with 8 leaf switches and 8 spine switches to validate the HTPC's performance under symmetric topology. In this test, we increase one path's RTT to produce asymmetric topology. The link capacity and RTT are set to 10Gbps and 100μs, respectively. Besides, there are 400 short flows and 100 long flows coexisting on 8 multiple paths.

As shown in Fig. 12, when the RTT ratio of bad path to good one changes from 1.2 to 2, HTPC obtains the lower 99th percentile flow completion time and higher throughput than other protocols. On the one hand, since RPS causes serious packet reordering in asymmetric topology and ECMP is unable to make full of multiple paths due to hash collision, they get worse performance than other schemes. Besides, Corrective only generates one encoded packets in every congestion window, it fails to deal with the case that multiple packets within congestion window are lost or blocked on the bad paths. CAPS does not encode packets for long flows, potentially making some long flows experience timeout. On the other hand, HTPC dynamically adjusts the number of encoded packets according to the flow types and network status, effectively reducing the tailing probability of short flows and the timeout probability of long flows. As a result, HTPC outperforms other schemes under asymmetric scenario.

Furthermore, we evaluate the performance of HTPC under different network scales. In this test, we gradually expand the network scale by increasing the number of leaf switches from 8 to 16, the other experimental parameters are the same as previous scenario. In this test, we evaluate the performance of HTPC under the web search workload. Here, we measure the 99th percentile flow completion time of short flows and the throughput of long flows. The experimental results are shown in Fig. 13.

Figure 13 shows that the 99th percentile flow completion time of short flows becomes large and the throughput of long flows gradually reduces with the increasing of leaf switch amount. Fortunately, HTPC obtains lower flow completion time for short flows and higher throughput for long flows than other schemes under all scenarios. The main reason is that HTPC smoothly adjusts the number of encoded packets based on the characteristics of heterogeneous traffic and network status, reducing the tailing probability of short flows and the timeout probability of long flows.



(a) 99th percentile FCT of short flows

(b) Throughput of long flows

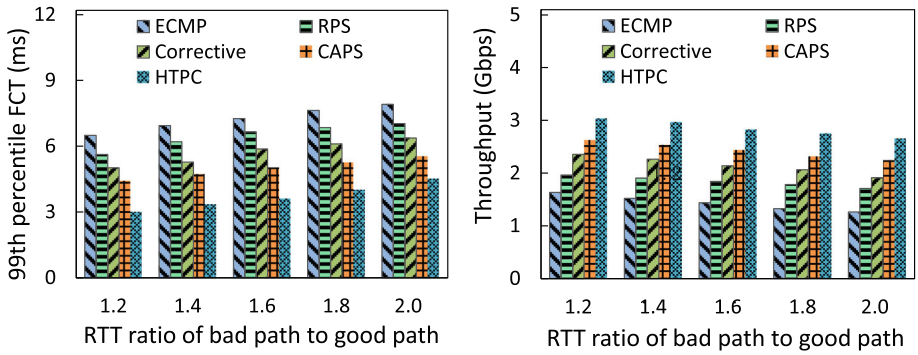
(c) AFCT of all flows

Fig. 11 Data mining

Related works

Modern data center networks widely adopt multi-rooted topologies such as leaf-spine and fat-tree [35] to transmit data in parallel. In recent years, although lots of transport control protocols [33, 36–38] have been proposed to reduce flow completion time, they fail to effectively improve the application-level performance. To this end, various load balancing mechanisms are proposed to make full use of bandwidth resources. Although these solutions can improve network performance, they still have shortcomings. The most relevant works are introduced as follows.

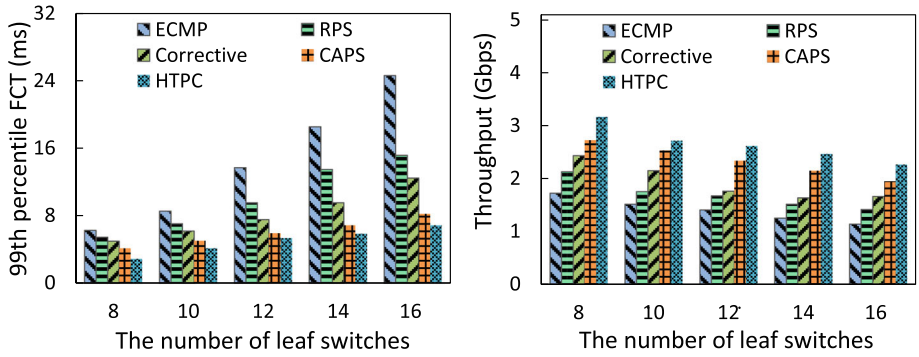
ECMP [20] is a flow-based scheme, which is the standard load-balancing mechanism used in today's data centers. In ECMP, the switch calculates the hash value based on five-tuple in packet header, and then maps each flow to



(a) 99th percentile FCT of short flows

(b) Throughput of long flows

Fig. 12 Performance under asymmetric topology



(a) 99th percentile FCT of short flows (b) Throughput of long flows

Fig. 13 Performance under different network scales

a static outgoing port according to the hash value. The key problem of ECMP is that some long flows may be transmitted on the same path due to hash collisions. These long flows can not be rerouted to other paths with low link utilization, resulting in low network utilization. To this end, some researchers have proposed more fine-grained mechanisms, which can be divided into flowlet-based, flowcell-based and packet-based approaches.

The flowlet-based schemes consists of CONGA [39], LetFlow [40], CLOVE [41] and CAF [42], which are designed to prevent packet reordering and achieve good network performance. The basic idea of them is that when the packet gap of the same flow exceeds a given threshold, the switch either shifts subsequent flowlets to the least congested paths or randomly spreads subsequent flowlets to one available path. Besides, Presto [43] is a flowcell-based transmission scheme and splits a flow into several units with fixed size (i.e., 64KB). However, Presto is insensitive to path conditions and then spreads these units to all available paths. Some flowcells may be transmitted on congested path, which inevitably degrading the network performance.

Although the flowlet-based schemes such as CONGA can improve network utilization in some scenarios, they do not reroute new flowlets until the time interval between two packets of the same flow exceeds a pre-determined threshold. Unfortunately, setting a right threshold value is very difficult under highly dynamic traffic in datacenters[1]. One too small threshold fails to avoid packet reordering while a too large threshold inevitably leads to inflexible path switching, making these flowlet-based approaches be unable to effectively use multiple paths. Fortunately, combined with network coding, per-packet load balancing solutions can not only efficiently make use of multiple paths, but also recover the blocked or lost data packets, reducing flow completion time. In short, network coding is a more promising technique than the routing (CONGA).

MPLOT [44] adjusts the coding redundancy using measured packet blocking rate. Then MPLOT maps the more useful packets to paths with shorter RTTs. However, MPLOT does not consider the differences of path quality. Moreover, its scheduling scheme is too simple to adapt to the highly dynamic data center networks.

FMTCP [45] adopts digital fountain code to flexibly solve the bottleneck problem of MPTCP. FMTCP designs a data allocation algorithm based on packet's expected arrival time and decoding requirements to coordinate the transmission of different subflows. When the path quality declines, the sender only needs to transmit the newly encoded packets according to the decoding requirements at receiver.

MPTCP-dFEC [46] combines dynamic FEC with MPTCP protocol and each subflow adjusts the coding algorithm according to its own path quality. MPTCP-dFEC can recover lost data through the FEC-encoded packets. However, when the redundant packets may be mapped to bad path, which makes the bad path become worse.

Conclusion

This paper presents a Heterogeneous Traffic-aware Partition Coding named HTPC to eliminate the impact of packet loss and reordering. HTPC dynamically adjusts the number of redundant packets according to the packet blocking rate and the traffic characteristics, which effectively reduces the tailing probability of short flows and the timeout probability of long flows. Our experimental results show that HTPC shortens the flow completion time by up to 60% compared with the state-of-the-art mechanisms.

Acknowledgements
Not applicable.

Authors' contributions
All authors have participated in conception and design, or analysis and interpretation of this paper. All authors read and approved the final manuscript.

Funding
This work is supported by the National Natural Science Foundation of China (61872387, 61872403), CERNET Innovation Project (Grant No. NGII20170107), Project of Foreign Cultural and Educational Expert (G20190018003).

Availability of data and materials
Not applicable.

Declarations

Competing interests
The authors declare that they have no competing interests.

Author details
[1]School of Computer Science and Engineering, Central South University, 410083 Changsha, China. [2]School of Computer Engineering and Applied Mathematics, Changsha University, 410022 Changsha, China. [3]Hunan Province Key Laboratory of Industrial Internet Technology and Security, Changsha University, 410022 Changsha, China. [4]School of Computer and Communication Engineering, Changsha University of Science and Technology, 410114 Changsha, China.

Received: 25 August 2020 Accepted: 21 May 2021
Published online: 05 June 2021

References

1. Alizadeh M, Greenberg A, Maltz D, Padhye J, Patel P, Prabhakar B, Sengupta S, Sridharan M (2010) Data center tcp (DCTCP). In: Proceedings of the ACM SIGCOMM: 30 August-September 3 2010; New Delhi. pp 63–74
2. Zhang T, Wang J, Huang J, Chen J, Pan Y, Min G (2017) Tuning the aggressive TCP behavior for highly concurrent HTTP connections in intra-datacenter. IEEE/ACM Trans Networking 25:3808–3822
3. Huang J, Huang Y, Wang J, He T (2020) Adjusting packet size to mitigate TCP incast in data center networks with COTS switches. IEEE Trans Cloud Comput 8(3):749–763
4. Zeng G, Bai W, Chen G, Chen K, Han D, Zhu Y (2017) Combining ECN and RTT for datacenter transport. In: Proceedings of the Asia-Pacific Workshop on Networking: 3-4 August 2017; Hong Kong. pp 1–7
5. Ghemawat S, Gobioff H, Leung S-T (2003) The google file system. In: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles. pp 29–43
6. Dean J, Ghemawat S (2008) MapReduce: simplified data processing on large clusters. Commun ACM 51(1):107–113
7. Zhou Z, Shojafar M, Alazab M, Abawajy J, Li F (2021) AFED-EF: An Energy-efficient VM Allocation Algorithm for IoT Applications in a Cloud Data Center. IEEE Trans Cogn Commun Netw:1–12
8. Cho I, Jang K, Han D (2017) Credit-scheduled delay-bounded congestion control for datacenters. In: Proceedings of the ACM SIGCOMM. pp 239–252
9. Liu F, Guo J, Huang X, Lui JCS (2017) eBA: Efficient bandwidth guarantee under traffic variability in datacenters. IEEE/ACM Trans Networking 25(1):506–519
10. Liu J, Huang J, Lv W, Wang J (2020) APS: Adaptive packet spraying to isolate mix-flows in data center network. IEEE Trans Cloud Comput:1–14. https://doi.org/10.1109/TCC.2020.2985037
11. Munir A, Qazi IA, Uzmi ZA, Mushtaq A, Ismail SN, Iqbal MS, Khan B (2013) Minimizing flow completion times in data centers. In: Proceedings of IEEE INFOCOM. pp 2157–2165
12. Hoff T (2009) Latency is everywhere and it costs you sales how to crush it. http://highscalability.com/latency-everywhere-and-it-costs-you-sales-how-crush-it
13. Dixit A, Prakash P, Hu YC, Kompella RR (2013) On the impact of packet spraying in data center networks. In: Proceedings of the IEEE INFOCOM. pp 2130–2138
14. Huang J, Lyu W, Li W, Wang J, He T (2021) Mitigating packet reordering for random packet spraying in data center networks. IEEE/ACM Trans Networking:1–14. https://doi.org/10.1109/TNET.2021.3056601
15. Flach T, Dukkipati N, Terzis A, Raghavan B, Govindan R (2013) Reducing web latency: the virtue of gentle aggression. In: Proceedings of the ACM SIGCOMM: 12-16 August 2013; Hong Kong. pp 159–170
16. Hu J, Huang J, Lv W, Zhou Y, Wang J (2019) CAPS:coding-based adaptive packet spraying to reduce flow completion time in data center. IEEE/ACM Trans Networking 27(6):2338–2353
17. Bai W, Chen L, Chen K, Han D, Tian C, Wang H (2017) PIAS: Practical information-agnostic flow scheduling for commodity data centers. IEEE/ACM Trans Networking 25(4):1954–1967
18. Carpio F, Engelmann A, Jukan A (2016) DiffFlow: Differentiating short and long flows for load balancing in data center networks. In: Proceedings of IEEE GLOBECOM. pp 1–6
19. Huang J, Li W, Li Q, Zhang T, Dong P, Wang J (2020) Tuning high flow concurrency for MPTCP in data center networks. J Cloud Comput 9(1):1–15
20. Hopps C (2000) Analyssis of an equal-cost multi-path algorithm. In: RFC 2992. pp 1–8
21. Bai W, Chen K, Chen L, Kim C, Wu H (2016) Enabling ECN over generic packet scheduling. In: Proceedings of the ACM CoNEXT: 12-15 December 2016; Irvine. pp 191–204
22. Shan D, Jiang W, Ren F (2017) Analyzing and enhancing dynamic threshold policy of data center switches. IEEE Trans Parallel Distrib Syst 28:2454–2470
23. Shan D, Ren F, Cheng P, Shu R, Guo C (2018) Micro-burst in data centers: Observations, analysis, and mitigations. In: Proceedings of the IEEE ICNP: 24-27 September 2018; Cambridge. pp 88–98
24. Shan D, Ren F (2017) Micro-burst in data centers: Observations, analysis, and mitigations. In: Proceedings of the IEEE INFOCOM: 1-4 May 2017; Atlanta. pp 1–9
25. Zhang J, Ren F, Tang L, Lin C (2013) Taming TCP incast throughput collapse in data center networks. In: Proceedings of the IEEE ICNP. pp 1–10
26. Cloud J, Leith D, Medard M (2015) A coded generalization of selective repeat ARQ. In: Proceedings of the IEEE INFOCOM: 26 April-1 May; Hong Kong. pp 2157–2165
27. Li M, Lukyanenko A, Tarkoma S, Cui Y, Ylae-Jaeaeski A (2014) Tolerating path heterogeneity in multipath TCP with bounded receive buffers. Comput Netw 64(8):1–14
28. Zhou Z, Abawajy J, Chowdhury M, Hu Z, Li K, Cheng H, Alelaiwi AA, Li F (2018) Minimizing SLA violation and power consumption in Cloud data centers using adaptive energy-aware algorithms. Futur Gener Comput Syst 86(2018):836–850
29. Zhou Z, Li F, Zhu H, Xie H, Abawajy JH, Chowdhury MU (2020) An improved genetic algorithm using greedy strategy toward task scheduling optimization in cloud environments. Neural Comput & Applic 32(6):1531–1541
30. Huang J, Li S, Han R, Wang J (2019) Receiver-driven fair congestion control for TCP outcast in data center networks. J Netw Comput Appl 131:75–88
31. Ren Y, Zhao Y, Liu P, Dou K, Li J (2014) A survey on TCP incast in data center networks. Int J Commun Syst 27:1160–1172
32. Hu S, Zhu Y, Cheng P, Guo C, Tan K, Padhye J, Chen K (2017) Tagger: Practical PFC deadlock prevention in data center networks. In: Proceedings of the ACM CoNEXT: 12-15 December 2016; Seoul/Incheon. pp 451–463
33. Hu S, Bai W, Zeng G, Wang Z, Qiao B, Chen K, Tan K, Wang Y (2020) Aeolus: A building block for proactive transport in datacenters. In: Proceedings of the ACM SIGCOMM. pp 422–434
34. Susanto H, Jin H, Chen K (2016) Stream: Decentralized opportunistic inter-coflow scheduling for datacenter networks. In: Proceedings of the IEEE ICNP: 8-11 Nov. 2016; Singapore. pp 1–10
35. Liu S, Huang J, Zhou Y, Wang J, He T (2019) Task-aware TCP in data center networks. IEEE/ACM Trans Networking 27:389–404
36. Zou S, Huang J, Wang J, He T (2021) Flow-aware adaptive pacing to mitigate TCP incast in data center networks. IEEE/ACM Trans Networking 29(1):134–147
37. Zhang J, Bai W, Chen K (2019) Enabling ECN for datacenter networks with RTT variations. In: Proceedings of the ACM CoNEXT. pp 233–245
38. Zhang T, Huang J, Chen K, Wang J, Chen J, Pan Y, Min G (2020) Rethinking fast and friendly transport in data center networks. IEEE/ACM Trans Networking 28(5):2364–2377
39. Alizadeh M, Edsall T, Dharmapurikar S, Vaidyanathan R, Varghese G (2014) CONGA: Distributed congestion-aware load balancing for datacenters. In: Proceedings of the ACM SIGCOMM: 17-22 August 2014; Chicago. pp 503–514

40. Alizadeh M, Edsall T, Dharmapurikar S, Vaidyanathan R, Varghese G (2017) Let it flow: Resilient asymmetric load balancing with flowlet switching. In: Proceedings of the USENIX NSDI: 9-11 April 2017; Renton. USENIX, Boston. pp 407–420
41. Katta1 N, M. H, Ghag A, Kim C, Keslassy I, Rexford J (2016) CLOVE: How i learned to stop worrying about the core and love the edge. In: Proceedings of the ACM HotNets: 9-10 November 2016; Atlanta. pp 155–161
42. Zou S, Huang J, Jiang W, Wang J (2020) Achieving high utilization of flowlet-based load balancing in data center networks. Futur Gener Comput Syst 108:546–559
43. He K, Rozner E, Agarwal K, Felter W, Carter J, Akella A (2015) Presto: Edge-based load balancing for fast datacenter networks. In: Proceedings of the ACM SIGCOMM: 17-21 August 2015; London. pp 465–478
44. Sharma V, Kalyanaraman S, Kar K, Ramakrishnan KK, Subramanian V (2008) MPLOT: A transport protocol exploiting multipath diversity using erasure codes. In: Proceedings of the IEEE INFOCOM: 13-18 April 2008; Phoenix. pp 121–125
45. Cui Y, Wang L, Wang X, Wang H, Wang Y (2015) FMTCP: A fountain code-based multipath transmission control protocol. IEEE/ACM Trans Networking 23:465–478
46. Ferlin S, Kucera S, Claussen H, Alay Ö (2018) Mptcp meets fec: Supporting latency-sensitive applications over heterogeneous networks. IEEE/ACM Trans Networking 26:2005–2018

Publisher's Note