Improving TCP Performance in Data Center Networks with Adaptive Complementary Coding

Jiyan Sun†, Yan Zhang†*, Ding Tang†, Shuli Zhang†, Zhen Xu† and Jingguo Ge†
†State Key Laboratory of Information Security, Institute of Information Engineering,
Chinese Academy of Sciences, Beijing, China
*Corresponding author, email: zhangyan80@iie.ac.cn

***Abstract*—TCP suffers from low throughput and high latency because of its expensive timeout based loss recovery mechanism in data center networks (DCNs). In this paper, we propose TCP with Adaptive Complementary Coding (TCP-ACC) to effectively address these problems. Without revising existing TCP congestion control, we first design a light-weight complementary coding scheme to avoid TCP timeout which will result in higher throughput and lower latency. In our scheme, the redundancy setting is adaptive to the real-time packet loss rate. Then we introduce Lyapunov optimization framework to find the optimal number of redundant coding packets for TCP-ACC, and we also prove that TCP-ACC can reduce the flow timeout probability close to that of the optimal complementary coding solution. Extensive NS2 simulations show that, compared with other three solutions for TCP's problems in DCNs, TCP-ACC can reduce the flow completion time by 45% and improve the flow throughput by 40% on average.**

I. Introduction

TCP is the dominant transport protocol in today's data center networks (DCNs) which have been the infrastructures for Internet and cloud computing. However, the widespread TCP has many inadequacies in meeting the throughput and latency demand of Internet applications and cloud services. First, TCP Incast [1], which refers to throughput collapse in many-to-one communication pattern, has risen to be a critical problem in DCNs. Since the many-to-one communication pattern widely exists in many Internet applications and cloud services (eg., web search and Map-Reduce), this problem could badly degrade their performance. Second, in DCNs, short TCP flows often suffer from high delays when long TCP flows occupy the link bandwidth for a long time. Since a large portion of traffic in DCNs is short flows which are latency sensitive and latency inversely correlates with business profit (Amazon estimates every 100ms of latency costs them one percent profit [2]), this problem is also critical for TCP in DCNs.

Actually, both of the two problems are highly related with TCP timeout. In the case where TCP Incast occurs, since no TCP sender can transfer a new data unit to the receiver until all senders finish transferring their current data units, even one flow's timeout can significantly reduce the whole throughput of all the flows. Moreover, timeout can easily prolong the flow completion time (FCT), which is the main metric of latency. Unfortunately, TCP timeout is common in today's DCNs. The authors of Corrective take a measurement of billions of TCP connections from clients to Google DCNs and found that about 10% of the TCP flows beget at least one packet loss. Furthermore, among all the losses in the measurement, 77% are recovered by timeout [3].

To improve TCP performance on throughput and latency in DCNs, many solutions that focus on avoiding timeout are provided [4]–[8]. Compared with transport layer solutions which need to make changes on switches or network interfaces (e.g., DCTCP [5], ICTCP [7] and ICaT [8]) and application layer solutions which spend expensive overhead costs in global real-time scheduling (e.g., [9], [10]), redundant transmission based solutions show great advantages in deployment and system overhead, and thus become research hotspots.

Existing redundant transmission based solutions for avoiding TCP timeout in DCNs can be grouped into two categories. The first is *pure* redundant transmission, which means that the redundancy is original flow [11], [12] or packet [1]. The second is *coding* redundant transmission, which means that the redundancy is coded packet (mixed by original packets [3]. Compared with *pure* redundant transmission, *coding* redundant transmission could be more efficient, since the coded redundant packet can recover any original packet mixed in it [13]. However, the existing *coding* redundant retransmission based solution (e.g., *Corrective* [3]) still has weaknesses and can be improved.

In this paper, we present a novel coding redundant transmission based solution named TCP with Adaptive Complementary Coding (TCP-ACC) to improve TCP performance on throughput and latency in DCNs. Without revising TCP's congestion control mechanism, we first design a light-weight coding scheme to prevent timeout and ensure consist transmission for flows. Compared with *Corrective*, TCP-ACC is more flexible in redundancy setting and can recover multiple losses in one window. Second, to determine the optimal number of redundant coding packets in our scheme, we model and analyze the tradeoff between avoiding TCP timeout and aggravating network congestion by a Lyapunov optimization method [14]. By fast solving this model, TCP-ACC can adapt its coding redundancy to the real-time packet losses in an near optimal way, and thus can effectively improve TCP performance on both throughput and latency. Analysis on this model also proves that TCP-ACC can reduce the flow timeout probability close to that of the optimal complementary solution. To the

978-1-4673-6770-7/15/$31.00 ©2015 IEEE

best of our knowledge, this is the first work that shows how to apply the Lyapunov optimization method to schedule the coding process. We carry out extensive simulations on NS2 to compare the performance of TCP-ACC with other three solutions for TCP's problems in DCNs. Simulation results show that TCP-ACC can nearly halve the FCT and double the throughput with different flow arrival rates and flow sizes.

The remainder of this paper is organized as follows. We first introduce the related works in Section II. In Section III, we present the details of our TCP-ACC scheme. We further provide a Lyapunov optimization model for our complementary coding scheme and analyze it in Section IV. We evaluate the performance of the proposed schemes in Section V and conclude our work in Section VI.

II. Related Works

As mentioned before, the transport layer solutions for avoiding TCP timeout in DCNs include DCTCP [5], ICTCP [7] and ICaT [8]. DCTCP employs the early congestion notification (ECN) scheme to keep a small switch queue length. ICTCP utilizes the dynamical Advertised Window (awnd) to help adjusting the sending rate. ICaT proposes admission control for TCP flows to ensure that the flow number would not exceed the capacity of the network. However, they need to make changes in network devices such as routers, switches and network interfaces. What's more, these new protocols are far less mature in robustness. For example, ICTCP only focuses on the incast scenarios where the last hop must be the bottleneck and DCTCP preforms poorly when the number of nodes becomes large [7]. ICaT does not provide differentiated services, some short flows may have no chance to get the transmitting admission with a great probability. Being different with the transport layer solutions, the application layer solutions including [9], [10] manage to schedule the transfer time of each TCP flow appropriately to avoid timeout, without any modifications on switches or network interfaces. However, the scheduling work is performed by a centralized server, and also needs real-time information about all TCP flows and the network status. This brings great system overhead on computation and measurements.

Compared with the above solutions, redundant transmission based solutions for avoiding TCP timeout in DCNs have great advantages in deployment and system overhead. As mentioned before, the *pure* redundant transmission based solutions include pure flow redundant transmission *Repflow* [11], *More-is-less* [12] and pure packet redundant transmission scheme GIP (Guaranteeing Important Packet) [1]. In data center networks, the need of low latency outweighs the demand to save bandwidth for short flows which are delay sensitive [5]. *Repflow* and *More-is-less* replicate short flows to reduce their delay. GIP redundantly transmits the last packet of a stripe unit to avert timeout for fast retransmission. However, they can only recover the specific losses which are redundantly transmitted. The coding redundant transmission based solutions include *Corrective* [3], which adds a random linear combination of the packets in the congestion window as the transmission redundancy. *Corrective* can recover single packet loss in one window without loss detect delay and retransmission delay. Compared with the *pure* redundant transmission based solutions, *Corrective* can provide better transmission reliability, loss-tolerance and faster loss recovery. However, *Corrective* still has two major shortcomings. First, it cannot adapt to the case in which more than one packet in one window losses. Second, its coding redundancy is fixed (e.g., one coded packet for each congestion window), which should be more flexible to network status. For example, when the TCP timeout probability is not high, the redundancy may be unnecessary, since it can not only influence TCP's congestion control but also make the congestion even worse.

The above weaknesses motivate us to propose a novel coding redundant transmission based solution, which will be introduced in the next sections.

III. Design of the Adaptive Complementary Coding Scheme

In this section, we first introduce the framework and notations of TCP-ACC, and then present the details about the sender and receiver operations in our scheme respectively.

A. Framework and notations

The compatibility is essential to the design of TCP extensions. Middle-boxes are widely used but hard to handle, which make the design of TCP with coding technology more complex [15]. TCP-ACC retains the framework of TCP/NC [13] which contributes to good compatibility with existing protocol stacks. Inspired by TCP/NC, the ACC layer in TCP-ACC is added between the TCP and IP layer to implement the light-weight coding. It can make full use of the TCP information on loss and congestion to estimate the online timeout probability, and then adaptively generates complementary coding packets for loss recovery.

At the sender side of TCP-ACC, we store TCP packets in a buffer queue and remove them when they are acknowledged. TCP-ACC adopts the simple random linear coding by applying the header structure of the coding layer proposed in TCP/NC [13]. In the coding header, the sequence number of the newest acknowledged packet is denoted as $base$. The packets whose sequence number is less than $base$ will never be coded and can be removed safely from the ACC layer. The number of packets involved in the linear combination of coding is denoted as n. Specially, we set n as zero when the packet is an original TCP packet.

The TCP/NC is primarily designed for lossy wireless networks, in which every TCP packet should be coded to overcome the random wireless losses. However, DCNs are known as high-throughput and low-latency networks [5], [7], where applying such a heavy coding technology would incur high computation and transmission overheads. Hence different from TCP/NC, the sender of TCP-ACC will transmit redundant coding packets only when it detects a high timeout probability, otherwise, it sends original packets. This is what *complementary coding* means: coding when needed to complement

the losses. In this work, we will show that a light-weight coding technology is possible and effective to improve TCP performance in DCNs.

At the receiver side, we additionally add the number of losses (denoted as d) in the newly generated ACKs, which in turn helps the sender to compute the real-time loss rate and adjust the coding redundancy. Once the sender gets the loss number d form the feedback, it can recover multiple losses in a same congestion window during single RTT. This is because the loss recovery of coding does not require the information about which packets are lost [13]. However, even *Corrective* and other standard TCP variants, like TCP NewReno which is popular in DCNs, can only recover one loss during one RTT.

For clarity, we list the notations and their meanings used in the later subsections as below.

- d: the loss number of packets at the receiver.
- d_{last}: the last value of d.
- d_{new}: the difference loss number between two contiguous ACKs.
- p: flow loss rate.
- w: the size of TCP congestion window.
- P^t: flow timeout probability.
- c_1: the number of complementary coding packets at the end of a TCP congestion window.
- c_2: the number of complementary coding packets when there are packet losses.
- c_{max}: the max number of complementary coding packets.
- l_c: the latest coded packet's sequence number.
- all_acked: a boolean variable to indicate whether the packets in the last congestion window are all acknowledged. Its initial value is true.

B. Sender operations in complementary coding design

At the sender side, TCP-ACC uses the light-wight coding redundancy to reduce the flow timeout probability while keeping necessary congestion control. The sender utilizes the new loss number d_{new} and the size of TCP congestion window w to calculate the real-time loss rate p and the timeout probability P^t. Then it adds proper complementary coding for flow transmission.

The sender will handle two different events at the ACC layer. First, ***Event I*** is triggered by packets arriving from the upper TCP layer. In this event, the sender will not only send the original TCP packet but also generate coding redundancy at the end of a congestion window if the current value of P^t is high. Second, ***Event II*** is triggered by ACKs from the receiver, in this event the ACC sender will update the estimation of P^t based on d.

According to TCP's congestion control, a flow will meet timeout at two situations: (1) There are too many packets are lost in a TCP congestion window (usually more than three); (2) When packet losses occur, unfortunately, the retransmission packets from the sender are also lost [8]. Correspondingly, we should make complementary coding to address the flow timeout problems in both of the two situations. The number of complementary coding packets in *Event I* and *Event II* is denoted as c_1 and c_2 respectively.

*1) Operations in **Event I**:* The core idea of the sender-side operations in ACC is to properly schedule the coding process based on real-time TCP timeout probability. In this way, a flow can keep stable transmission while avoiding simple retransmissions caused by expensive timeouts. However, we do not make complimentary for every TCP congestion window, because we should guarantee necessary congestion window reduction for complying with the mandatory TCP congestion control. Therefore, to help accurate TCP window adjustments, there is no extra complimentary when the timeout probability is low. In *Event I*, the sender should make c_1 complementary coding packets at the end of a congestion window to protect flows with small TCP congestion window. It prevents the delay-sensitive short flows from suffering large timeouts as the switch buffer is often kept being occupied by long flows [5].

Since the TCP congestion window can not be obtained directly at the ACC layer, we define **C**oding **C**ongestion **W**indow (CCW) instead, in which the sender will generate complementary coding to avoid the fisrt timeout situation. We also use a variable all_acked to help update the proper CCW. When the packets in the last CCW are all acknowledged, the packets which are sent but not acknowledged form the new CCW. Therefore, when all_acked is true, the current buffer becomes a new CCW. When we send the coding packets from the new CCW, the all_acked flag is set as false.

When a packet arrives from TCP, the sender of ACC will work as the following steps:

1) If the packet is a control packet used for connection management, deliver it to the IP layer.
2) If the packet is a data packet
   a) Add the packet to the buffer in ACC layer.
   b) Add the ACC header $base$ and set $n = 0$. Send it.
   c) If all_acked is true
      i) Generate c_1 coding packets mixing the packets in the current CCW and
      ii) Add the ACC header $base$ and set n as the number of original packets coded in the coding packets.
      iii) Send the coding packets and update l_c as the sequence number of the last packet in CCW, all_acked as false.

*2) Operations in **Event II**:* When an ACK arrives, the sender will check whether the last CCW is empty. Only when the packet with sequence number l_c is acknowledged, the packets in the last CCW are all acknowledged. The current buffer becomes the new CCW. In *Event II* the sender should make c_2 complementary coding packets to guarantee the reliability of the retransmission packets when the sender senses packet losses.

When an ACK arrives at the sender, it will work as the following steps:

1) If l_c is less than the sequence of the ACK, then update all_acked is true.
2) Remove the acknowledged packet from the CCW.

3) Extract the loss number d and hand over the packet to the TCP sender.
4) Update p and loss number d.
5) Set the new loss number d_{new} as $d_{last} - d$.
6) If $d_{new} > 0$
   a) Generate c_2 coding packets mixing the packets in the current CCW and send the packets.
   b) Add the ACC header $base$ and set n as the number of original packets coded in the coding packets.
   c) Send the coding packets and set $d_{new} = 0$ and $d_{last} = d$.

According to the above process, the sender will determine the value of c_2 based on the loss number provided by the receiver. Intuitively, c_2 can be set directly as the value of d_{new}. However, c_2 should be higher than the loss number in reality because the coded packets may also be lost.

One key point of ACC is to determine the proper values of c_1 and c_2. In ACC, the exact number of complementary coding packets is not fixed but adaptive to the network state. Specifically, the values of c_1 and c_2 can be adjusted according to the detected packet loss probability p. Intuitively, c_1 and c_2 should also increase as p increases, since more redundant packets will be more effective to recover losses and avoid timeout. However, this is not always the case because too many redundant packets will further deteriorate network congestion when the packet loss probability is already high. Therefore, for ACC, there is a tradeoff between the ability of avoiding **T**imeout and aggravating **C**ongestion (named as T-C tradeoff in the later), and the values of c_1 and c_2 should be determined with a careful tradeoff. We develop an optimization model for this tradeoff and solve it by the Lyapunov optimization method, which will be presented in the next section (see Section IV).

C. Receiver operations in complementary coding design

At the receiver side, there are two types of packets to handle, the original packets and coded packets. When a new TCP packet arrives, the receiver should first check whether it is coded. Only when the value of n in the coding header is larger than zero, it is a coded packet and requires decoding. The detailed operations at the receiver are described as follows.

When a packet arrives at the receiver of ACC,

1) If the packet is a control packet used for connection management, deliver it to TCP sink.
2) If the packet is a data packet
   a) Remove the ACC header of the packet and remove the packets whose sequence number is less than $base$ from ACC buffer.
   b) If n is 0, add the the packet to the nc buffer. Deliver it to the TCP sink.
   c) If n is larger than 0, decode it by Gaussian-Jordan elimination and deliver the decoded packets to TCP sink.
   d) Generate a new TCP ACK with the information of the current loss number d and send it.

IV. Determining the Optimal Complementary Number

In this section, we first model the T-C tradeoff through the Lyapunov optimization method. By solving this model, TCP-ACC can get the proper values of c_1 and c_2. Then we analyze this model and prove the finiteness of the timeout probability and the network congestion extent resulted from our scheme.

A. T-C tradeoff optimization

Consider the high performance requirements of distributed applications in DCN, we take into account both the flow transmission performance and coding overhead. To solve the T-C tradeoff, we schedule a fast and effective complementary coding process by introducing the Lyapunov optimization framework, which is generally used as an online scheduler design principle for optimizing two tradeoff factors [14].

The Lyapunov scheduler needs to decide the transmission rate of complementary coding packets based on TCP timeout probability which is decided by TCP loss rate p, the new loss number d_{new} and congestion window size w. For the first timeout situation, there are less than three duplicate ACKs to trigger fast retransmission recovery for a congestion window. Then the timeout probability P_1^t is computed as follow:

$$P_1^t(w, p, c_1) = \sum_{i=0}^{2} C_{c_1+w}^{c_1+w-i} p^{c_1+w-i}(1-p)^i \tag{1}$$

For the second timeout situation, when there are d_{new} new losses occur, the retransmission packets are also lost. The timeout probability P_2^t is as follow:

$$P_2^t(d_{new}, p, c_2) = 1 - \sum_{j=0}^{c_2} C_{c_2+w}^{j} p^{j}(1-p)^{c_2+w-j} \tag{2}$$

Hence we have

$$P^t(d_{new}, w, p, \mathcal{C}) = \begin{cases} P_1^t(w, p, c_1) & \text{, if the 1st timeout case} \\ P_2^t(d_{new}, p, c_2) & \text{, if the 2nd timeout case} \end{cases} \tag{3}$$

To apply the Lyapunov optimization in our system, we assume that the sender generates $\lambda(t)$ coding packets every RTT and we have a virtual buffer with capacity size of S coding packets. The compensation packets we send to the receiver every RTT is $\mathcal{C}(t)$. Consider a queueing network $Q(t)$ that denotes the *remaining* space of virtual buffer at time t. We have to note that the definition of queue here is not a conventional queue that used for buffering packets. It is defined as the remaining space of virtual buffer deployed at the source server. Note that $Q(t)$ is a queueing network that evolves in discrete time with normalized time. We can visually see that as the queue gets longer(which means more compensation packets are sent), the congestion will get heavier but the timeout probability will get lower. Therefore, $Q(t)$ directly describes the congestion extent caused by sending coding packets into the network.

Following the Lyapunov framework [14], to optimally control our dynamical system, the Lyapunov function can be

defined as

$$L(t) = \frac{1}{2}Q^2(t) \tag{4}$$

Then the queue length change over time as below:

$$Q(t+1) = \max[Q(t) + \mathcal{C}(t) - \lambda(t), 0] \tag{5}$$

where $\lambda(t)$ and $\mathcal{C}(t)$ are arrivals and sending rates of complementary coding packets at time slot t respectively.

Denote the queue length as the performance cost of the coding overhead at time t. Then the one-step Lyapunov drift $\Delta(t)$ is defined as:

$$\Delta(t) = L(t+1) - L(t) \tag{6}$$

We add the timeout probability minimization objective into the Lyapunov drift by the *drift-plus-penalty* form $\Delta(t) + VP^t(t)$ and generate the following lemma:

Lemma 1: Assume that the data arrival process $\lambda(t)$, and the transmission process $\mathcal{C}(t)$ have finite expectation, i.e., $\exists$ constants $\widetilde{\lambda}$ and $\widetilde{\mathcal{C}}$ such that $\mathbb{E}\{\lambda(t)\} \leq \widetilde{\lambda}$ and $\mathbb{E}\{\mathcal{C}(t)\} \leq \widetilde{\mathcal{C}}$. We have

$$\begin{aligned} &\Delta(t) + V\mathbb{E}\{P^t(t)\} \\ &\leq B + \mathbb{E}\{\mathcal{C}(t)Q(t) + VP^t(t)|Q(t)\} + \widetilde{\lambda}Q(t)] \end{aligned} \tag{7}$$

where $B \triangleq \frac{1}{2}\{\widetilde{\lambda}^2 + \widetilde{\mathcal{C}}^2\}$.

Based on Equ. (3), (4), (5) and (6), we can easily obtain the above Lemma. Following the Lyapunov optimization theory, we need to minimize the RHS of Equ. (7) at each time slot t. It is equivalent to choose the optimal transmission rate $\mathcal{C}(t)$ of the complementary coding packets for the following optimization problem

$$Minimize \quad \mathcal{C}(t)Q(t) + VP^t(t) \tag{8}$$

where V is a constant control parameter and $Q(t)$ is the queue length that can be easily obtained at time t. $P^t(t)$ is the exponential function of $\mathcal{C}(t)$ shown in equation (3).

Since $P^t(t)$ in the objective equation (8) has an exponential form of the decision variable $\mathcal{C}(t)$ (shown in the equations (1) and (2)), solving the optimal solution of $\mathcal{C}(t)$ to the minimization problem will introduce a high computation overhead. Considering the tradeoff between coding benefit and computation overhead, we limit $\mathcal{C}(t)$ to be within a constant value C_{max} and thus enable a fast search of the suboptimal solution of $\mathcal{C}(t)$ for the minimization problem. In this way, we can set the value of c_1 and c_2 as $\mathcal{C}(t)$ at time slot t corresponding to *Event I* and *Event II* respectively. On the other hand, when there are d_{new} new losses occur, we should take full advantage of coding technology to accelerate the loss recovery. That is, if $d_{new} < C_{max}$, the fast search of the suboptimal solution of $\mathcal{C}(t)$ should be performed in the range of $[d_{new}, C_{max})$. In this way, we can transmit at least d_{new} coding packets for fast retransmission.

B. T-C tradeoff analysis

In this section, we analyze the bound properties for both the timeout probability and the queue length when using the above Lyapunov-based method.

Theorem 1: Assume that the data arrival rate is strictly within the network capacity region, and the above online scheduling decision is applied at each time slot. For any control parameter $V > 0$, it generates the time-average timeout probability $\overline{P^t}$ and time-average queue length $\overline{Q}$ satisfying:

$$\overline{P^t} = \lim_{\Gamma\to\infty} \sup \frac{1}{\Gamma}\sum_{\tau=0}^{\Gamma-1} \mathbb{E}\{P^t(\tau)\} \leq P^* + \frac{B}{V} \tag{9}$$

$$\overline{Q} = \lim_{\Gamma\to\infty} \sup \frac{1}{\Gamma}\sum_{\tau=0}^{\Gamma-1} \mathbb{E}\{Q(\tau)\} \leq \frac{B + VP^*}{\varepsilon} \tag{10}$$

where B and ε are positive constants. P^* is the theoretical optimal time-average timeout probability.

Proof of Theorem 1: Since the arrival process is strictly within the network capacity region, there exists one stationary randomized control policy that can stabilize the queue [14], which satisfies the following properties:

$$\mathbb{E}\{P^t(t)\} = P^* \tag{11}$$

$$\mathbb{E}\{\mathcal{C}(t)\} \geq \ \lambda, \text{ i.e., } \ \mathbb{E}\{\mathcal{C}(t)\} = \lambda + \varepsilon, (\varepsilon > 0). \tag{12}$$

where P^* is the minimum achievable timeout probability using any control policy that achieves the queue stability.

By applying (11) and (12) to (7), we have:

$$\Delta(t) + V\mathbb{E}\{P^t(t)\} \leq B - \varepsilon Q(t) + VP^* \tag{13}$$

By taking the expection of (13) and applying iterative expectation law, we have:

$$\mathbb{E}\{L(t+1) - L(t)\} + V\mathbb{E}\{P^t(t)\} \leq B - \varepsilon\mathbb{E}\{Q(t)\} + VP^* \tag{14}$$

By summing (14) over all time slots $t \in \{0, 1, ..., \Gamma - 1\}$ and dividing by the time period Γ, we have:

$$\frac{\mathbb{E}\{L(\Gamma) - L(0)\}}{\Gamma} + \frac{V}{\Gamma}\sum_{\tau=0}^{\Gamma-1}\mathbb{E}\{P^t(\tau)\} \leq B - \frac{\varepsilon}{\Gamma}\sum_{\tau=0}^{\Gamma-1}\mathbb{E}\{Q(\tau)\} + VP^* \tag{15}$$

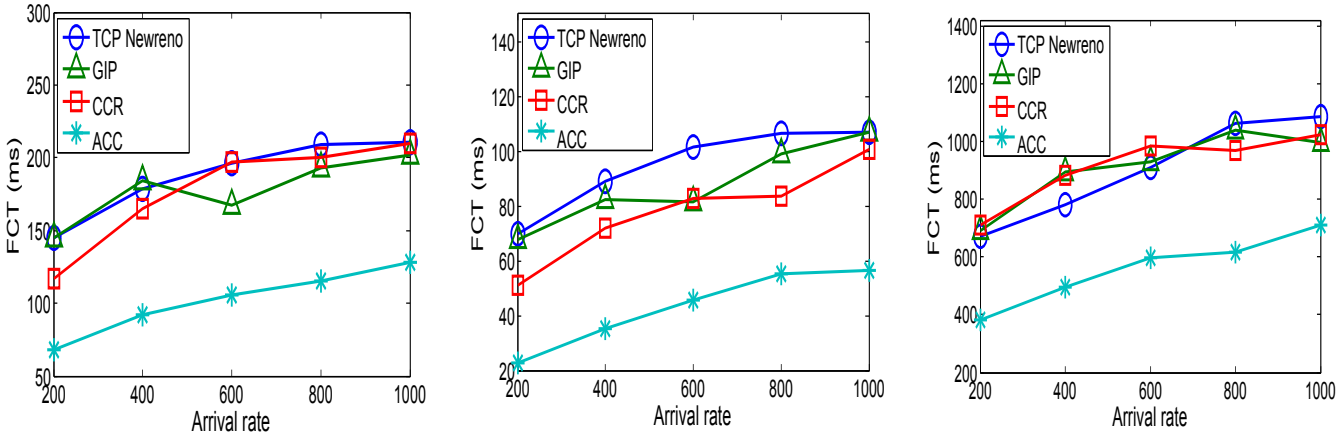
Since both the Lyapunov function and $P^t(t)$ are non-negative, based on (15), we have:

$$\frac{1}{\Gamma}\sum_{\tau=0}^{\Gamma-1}\mathbb{E}\{Q(\tau)\} \leq \frac{B + VP^* + \mathbb{E}\{L(0)\}/\Gamma}{\varepsilon} \tag{16}$$

$$\frac{1}{\Gamma}\sum_{\tau=0}^{\Gamma-1}\mathbb{E}\{P^t(\tau)\} \leq P^* + \frac{B}{V} + \frac{\mathbb{E}\{L(0)\}}{V\Gamma} \tag{17}$$

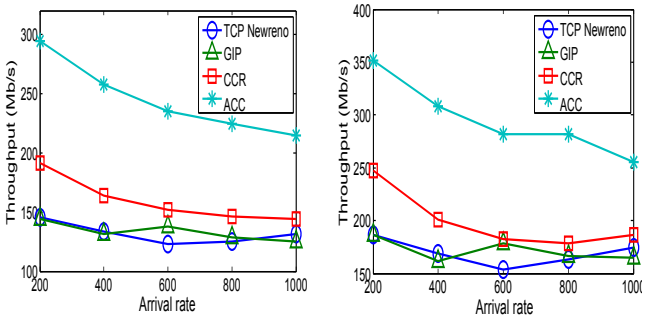
Taking the limit as $\Gamma \to \infty$ for (16) and (17), we can obtain the average timeout probability bound (9) and queue length bound (10) respectively. This completes the proof. ■

According to Theorem 1, we can see that the time-average timeout probability and the time-average queue length both have a definite upper bound. This means that, based on the Lyapunov optimization method, our complementary coding scheme will not cause the worst case in which the timeout probability or the network congestion extent is out of control. More importantly, we can also see that when the control



(a) Overall: Avg FCT (b) Short flow: Avg FCT (c) 99-th percentile: FCT

Fig. 1: Average FCT with multiple arrival rates



(a) Overall: Avg Throughput (b) Long flow: Avg Throughput

Fig. 2: Average throughput with multiple arrival rates

parameter V is increasing, the uppper-bound of timeout probability P^t is converging to the optimal value P^*. This theoretically proves that TCP-ACC can reduce the timeout probability close to that of the optimal complementary coding solution. However, at the same time, the upper-bound of Q is also increasing, which means more complementary packets are sent and thus aggravating the network congestion. Therefore, the control parameter V should be adjusted to effectively minimize the timeout probability while avoiding intolerable network congestion. Actually, there is no optimal setting of V if we do not define what the optimal term here means in a mathematic way. Since V is a trade-off parameter between avoiding the timeout and the congestion, it represents a preference weight on which factor the system desire a better performance. If we desire avoiding the timeout more than relieve the congestion, we should put a larger V value; Otherwise, we should put a smaller V value. In reality, to consider a comfortable trade-off between the two metrics, we test them under different input environment factors, such as different flow arrival patterns, flow sizes. We do not search the optimal V value as an exact number under different input parameters. In another way, through extensive simulations, we find that the two metrics are both good under the range [2000, 5000], which reveals the range of the pareto optimal points of these two metrics.

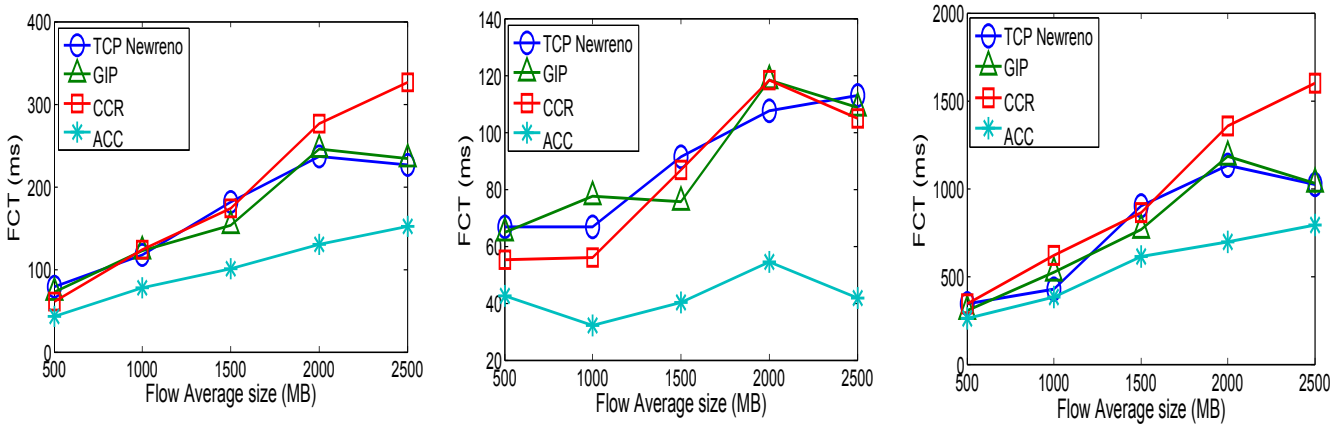
V. Performance Evaluations

In this section, we evaluate the performance of TCP-ACC against three state-of-art transmission strategies through extensive simulations on NS-2.

A. Simulation setup

We simulate an DCN as a commonly used 3-layer Fat-Tree topology with 8 pods [16], where the link bandwidth is set as 1Gbps. The topology consists of 32 ToR (Top-of-Rack) switches, and each ToR switch connects to 4 hosts, i.e., the network accommodates totally 128 hosts. The buffer size of each switch in this topology is 32 kB.

For comparative analysis, we evaluate four transmission schemes: (1) our adaptive complementary coding scheme TCP-ACC, (2) the *pure* redundant transmission scheme GIP [1], (3) the *coding* redundant transmission scheme CCR (i.e., the coding solution of *Corrective*) proposed in [3] , (4) the original DCNs' transmission scheme TCP NewReno [17]. We study the impact of different workloads on all the transmission schemes by scaling the average flow arrival rate and flow size. For the input traffic, we apply the flow model in [18] where flows arrive independently in a uniformly random way. Specially, previous measurements [5] reveal traffic in data center networks usually consists of delay-sensitive short messages (100KB to 1MB), and throughput-sensitive long flows (1MB to 100MB). Hence we additionally analyze the FCT for short flows whose size is smaller than 1MB and the throughput for long flows whose size is larger than 1MB respectively. To eliminate the long tail effect in DCN traffic, the performance of 99% flows is also evaluated. The detailed parameter setup of flow arrival rate and flow size is shown in Table I. During the simulations, when one factor is changed, the other factor is set to their default values. The values of the parameters of



(a) Overall: Avg FCT (b) Short flow: Avg FCT (c) 99-th percentile: FCT

Fig. 3: Average FCT with multiple flow sizes

the Lyapunov optimization based T-C tradeoff model are set as follows. V is 3000, C_{max} is 5, λ is 4, and the capacity size S of the virtual buffer is 1000 packets.

B. Simulation results with multiple arrival rates

Fig. 1 shows the FCT performance for all the transmission schemes under various flow arrival rates. Overall, we can see that TCP-ACC obtains much lower FCT than all peer schemes. Compared to the other schemes, the average FCT of TCP-ACC is reduced by about 50%, while the FCT of short flows is reduced by about 60%. It shows that TCP-ACC is more effective to protect the transmission of short flows, which are generally very sensitive to delays in DCNs. We can also see that, compared with TCP-Newreno, GIP and CCR can reduce the FCT, especially for short flows. This shows that adding redundancy actually gains higher benefits on FCT for short flows than long flows. However, their FCT performance for the 99^{th} percentile is still similar with that of TCP-Newreno, which indicates a poor performance on eliminating the long tail effect.

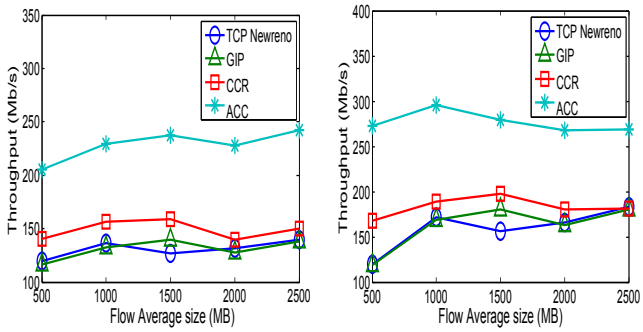
Fig. 2 shows the throughput performance of all the schemes under different flow arrival rates. We can see that TCP-ACC achieves much higher throughput than all peer schemes. Specifically, the average throughput of all flows and that of long flows are both increased by about 45%. Results in Fig. 1 and Fig. 2 show that, as the network get congested with increasing flow arrival rate, TCP-ACC can always achieve the best performance on both latency and throughput.

C. Simulation results with multiple flow sizes

Fig. 3 shows the FCT performance for all the transmission schemes under various flow sizes. Overall, we can see that TCP-ACC wins much lower FCT than all peer schemes. Compared to the other schemes, the average FCT of TCP-ACC is reduced by about 41%, while the FCT of short flows is reduced by about 52%. It shows again that TCP-ACC is much more effective to protect the transmission of short flows, although the increasing flow size would directly prolong the FCT for all flows. We can also see that, compared to TCP-Newreno, GIP and CCR have no obvious advantages on the FCT performance, even for short flows. This shows that their redundant transmission schemes cannot work effectively if the average flow size increases. Fig.3c shows that TCP-ACC is the most effective one to avoid the long tail effect as the average flow size increases.

TABLE I: Evaluation Parameters Setup

Parameters	Values	Default
Average Arrival rate	200,400,600,800,1000	600
Average Flow size(MB)	500,1000,1500,2000,25000	1500



(a) Overall: Avg Throughput (b) Long flow: Avg Throughput

Fig. 4: Average throughput with multiple flow sizes

Fig. 4 shows the throughput of all the schemes under various flow sizes. We can see that TCP-ACC achieves much higher throughput than all peer schemes. Specifically, the average throughput of all flows and that of long flows are both increased by about 43%. We can also see that, for all schemes, the throughput performance does not change much as the flow size increases. This is because the network will not get more congested if the number of flows does not increase. Results in Fig.3 and Fig.4 show that, as the flow size increases,

TCP-ACC can always achieve the best performance on both latency and throughput.

The above presented great performance advantages of TCP-ACC over other schemes can be interpreted as follows. First, its complementary coding scheme can ensure continuous flow transmission without losing necessary congestion control. This is important when the network get congested. Second, based on the Lyapunov optimization model, its adaptive coding scheme can ensure the optimal number of redundant packets, which achieves a good equilibrium of the tradeoff between reducing timeout probability and increasing network congestion extent.

VI. Conclusion

In this paper, we proposed an coding redundant transmission based solution named TCP-ACC (TCP with Adaptive Complementary Coding) to improve TCP performance on throughout and latency in DCNs. TCP-ACC is implemented in a new light-weight coding layer between the TCP and IP layer, which keeps its good compatibility to existing the protocol stacks. With flexible redundancy setting, TCP-ACC can avoid timeout effectively even for multiple losses in one window and thus ensure continuous flow transmission. To address the tradeoff between avoiding TCP timeout and aggravating network congestion for TCP-ACC, we introduced Lyapunov framework to model and optimize this tradeoff. By the Lyapunov optimizaiton method, TCP-ACC can determine the optimal number of coding redundant packets according to the real-time packet losses. We also theoretically proved that TCP-ACC can reduce the timeout probability close to that of the optimal complementary coding solution. We carried out extensive simulations by implementing the complete TCP-ACC coding layer in the original NS2-platform. Our evaluations show that, compared with other three solutions for TCP's problems in DCNs, TCP-ACC can always achieve the best performance both on throughput and latency in a wide range of flow arrival rates and sizes. Specially, it can reduce the flow completion time by 45% and improve the flow throughput by 40% on average.

Since our solution and all the other compared solutions do not make direct changes to TCP congestion window, to make a fair comparison, we only implement part of the whole Corrective solution. Specifically, we implement CCR as the key coding solution of Corrective, but does not implement the congestion control mechanism of Corrective which is similar to explicit congestion notification (ECN). In this way, we are easy to illustrate the effects of different coding solutions with comparable analysis results, rather than just show the performance of the whole simulated solution without useful analysis. We will compare the complete implementation of whole Corrective solution in the future work. We are also targeting for a real-world implementation of TCP-ACC in DCNs.

VII. Acknowledgment

This work is supported in part by National Science Foundation of China (Grant No. 61303250), the Strategic Pilot Project of Chinese Academy of Sciences (Grant No. XDA06010306). The corresponding author of this paper is Yan Zhang.

References

[1] J. Zhang, F. Ren, L. Tang, and C. Lin, "Taming tcp incast throughput collapse in data center networks," in *Network Protocols (ICNP), 2013 21st IEEE International Conference on*. IEEE, 2013, pp. 1–10.

[2] G. Linden, "Make data useful," 2006.

[3] T. Flach, N. Dukkipati, A. Terzis, B. Raghavan, N. Cardwell, Y. Cheng, A. Jain, S. Hao, E. Katz-Bassett, and R. Govindan, "Reducing web latency: the virtue of gentle aggression," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 159–170.

[4] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, G. A. Gibson, and B. Mueller, "Safe and effective fine-grained tcp retransmissions for datacenter communication," in *ACM SIGCOMM computer communication review*, vol. 39, no. 4. ACM, 2009, pp. 303–314.

[5] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," *ACM SIGCOMM computer communication review*, vol. 41, no. 4, pp. 63–74, 2011.

[6] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 50–61.

[7] H. Wu, Z. Feng, C. Guo, and Y. Zhang, "Ictcp: incast congestion control for tcp in data-center networks," *IEEE/ACM Transactions on Networking (TON)*, vol. 21, no. 2, pp. 345–358, 2013.

[8] A. S.-W. Tam, K. Xi, Y. Xu, and H. J. Chao, "Preventing tcp incast throughput collapse at the initiation, continuation, and termination," in *Proceedings of the 2012 IEEE 20th International Workshop on Quality of Service*. IEEE Press, 2012, p. 29.

[9] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient coflow scheduling with varys," in *Proceedings of the 2014 ACM conference on SIGCOMM*. ACM, 2014, pp. 443–454.

[10] C. W. Maxim Podlesny, "Solving the tcp-incast problem with application-level scheduling," 2012, pp. 99–106.

[11] H. Xu and B. Li, "Repflow: Minimizing flow completion times with replicated flows in data centers," in *INFOCOM, 2014 Proceedings IEEE*. IEEE, 2014, pp. 1581–1589.

[12] A. Vulimiri, O. Michel, P. Godfrey, and S. Shenker, "More is less: reducing latency via redundancy," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*. ACM, 2012, pp. 13–18.

[13] J. K. Sundararajan, D. Shah, M. Médard, S. Jakubczak, M. Mitzenmacher, and J. Barros, "Network coding meets tcp: Theory and implementation," *Proceedings of the IEEE*, vol. 99, no. 3, pp. 490–512, 2011.

[14] M. J. Neely, "Stochastic network optimization with application to communication and queueing systems," *Synthesis Lectures on Communication Networks*, vol. 3, no. 1, pp. 1–211, 2010.

[15] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda, "Is it still possible to extend tcp?" in *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*. ACM, 2011, pp. 181–194.

[16] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, pp. 63–74, 2008.

[17] N. Parvez, A. Mahanti, and C. Williamson, "An analytic throughput model for tcp newreno," *IEEE/ACM Transactions on Networking (TON)*, vol. 18, no. 2, pp. 448–461, 2010.

[18] N. Hamedazimi, Z. Qazi, H. Gupta, V. Sekar, S. R. Das, J. P. Longtin, H. Shah, and A. Tanwer, "Firefly: a reconfigurable wireless data center fabric using free-space optics," in *Proc. SIGCOMM*, 2014.