DC^2-MTCP: Light-weight Coding for Efficient Multi-path Transmission in Data Center Network

Jiyan Sun[1], Yan Zhang[1,*], Xin Wang[2], Shihan Xiao[2], Zhen Xu[1], Hongjing Wu[1], Xin Chen[1], Yanni Han[1]
[1] Institute of Information Engineering, Chinese Academy of Sciences, China
[2] Stony Brook University, USA
[*] Corresponding author, email: zhangyan80@iie.ac.cn

***Abstract*—Multi-path TCP has recently shown great potential to take advantage of the rich path diversity in data center networks (DCN) to increase transmission throughput. However, the small flows, which take a large fraction of data center traffic, will easily get a timeout when split onto multiple paths. Moreover, the dynamic congestions and node failures in DCN will exacerbate the reorder problem of parallel multi-path transmissions for large flows. In this paper, we propose DC^2-MTCP (Data Center Coded Multi-path TCP), which employs a fast and light-weight coding method to address the above challenges while maintaining the benefit of parallel multi-path transmissions. To meet the high flow performance in DCN, we insert a very low ratio of coded packets with a careful selection of the packets to be coded. We further present a progressive decoding algorithm to decode the packets online with a low time complexity. Extensive ns2-based simulations show that with two orders of magnitude lower coding delay, DC^2-MTCP can reduce on average 40% flow completion time for small flows and increase 30% flow throughput for large flows compared to the peer schemes in varying network conditions.**

***Keywords*-Data center networks; MPTCP; Network coding**

I. Introduction

Today's data center networks (DCN) provide large-scale symmetric paths to guarantee high aggregate network bandwidth and transmission reliability [1]. To fully utilize the bandwidth in DCN, Multi-path TCP (MPTCP) splits one application flow into multiple *subflows* and transmits them in parallel on different paths [2]–[4]. Since the link congestions happen randomly over space and time in DCN [5], [6], the subflows in one MPTCP connection will experience *asynchronous packet losses* (termed APL) [2], [7], which results in important performance issues when applying MPTCP in DCNs.

First, the APL in DCN will seriously reduce the delay performance of small flows, which take a large portion of DCN traffic [2], [5], [6], [8], [9]. The small flows often suffer from high loss probability when large flows occupy the link buffer [9]. Specifically, when a small flow is split into multiple subflows, the average congestion window of each subflow becomes even smaller, and thus causing a timeout more easily if any loss occurs [8]. When a timeout happens on a congested subflow, even though other subflows complete their transmissions first, the receiver has to wait for 200ms until the lost packet is retransmitted successfully by its original subflow [10]. Since small flows are often delay-sensitive and their latency are highly related to business profit (e.g., Amazon estimates every 100ms of latency costs them one percent profit [11]), such a long delay is generally unacceptable in DCN [5], [6], [9], [12].

Second, APL may block the orderly transmissions of large flows. As the receiver must deliver data to the application level in sequence, the asynchronous losses will result in a serious reorder problem. Specially, when a receiver's buffer is filled up with the disordered packets, all the newly received subflow packets have to be dropped, which therefore blocks the transmissions of large flows [10], [12].

Network coding has been exploited to alleviate the APL issues by improving the transmission reliability through encoding and recovering data packets across subflows [10], [13]–[15]. Despite the potential, there are several important challenges to apply the coding-based MPTCP designed for general networks [10], [14], [15] to the special environment of DCN. First, the introduced coding overhead (i.e., the coding delay and the number of coded packets to insert) should be low enough to meet the high transmission performance requirements in DCN. Second, with such a low coding overhead, it poses a great challenge to make the coded packets effective in improving the flow throughput and reducing the transmission time in the presence of dynamic DCN traffic.

To address the above challenges, in this paper, we propose DC^2-MTCP, a multi-path transmission protocol with a light-weight coding technique for subflows to coordinately recover the loss, which significantly improves the performance for both delay-sensitive small flows and throughput-sensitive large flows in DCN. To the best of our knowledge, this is the first work that studies coding-based multi-path TCP in DCN. The main contributions of our work are summarized as follows.

First, we propose the architecture design of DC^2-MTCP to ensure the ratio (denoted by ϵ, $\epsilon \ll 1$) of coded packets to be low in DCN. Benefiting from its flexibility without any revisions on the MPTCP congestion control and switching hardwares, it can be easily incorporated into existing multi-path transmission protocols with higher performance but little overhead. Second, at the sender side, we propose three strategies: 1) fast coding exploiting the packet sending

1530-2075/17 $31.00 © 2017 IEEE
DOI 10.1109/IPDPS.2017.40

intervals to reduce the coding delay; 2) a simple but effective scheme to carefully select the packets that should be coded for a higher performance benefitting from each coded packet; 3) a flow differentiated scheme to adapt the coding redundancy for small flows to achieve lower transmission delay and large flows to achieve higher throughput. Finally, at the receiver side, we develop a progressive decoding algorithm (PDA) by fully exploiting the feature of the low-enough coding ratio ϵ in DC^2-MTCP to decode the packets online. We show that the time complexity of PDA is at most ϵ times that of conventional decoding methods.

We implement DC^2-MTCP in ns2 and our extensive simulations driven by real DCN traffic traces show that DC^2-MTCP can reduce the flow completion time of small flows by 40% and improve the throughput of large flows by 30% under varying network conditions.

II. Background and Related Work

General issues in MPTCP. The APL is a common issue in multi-path transmission protocols. In the conventional loss recovery strategy, each subflow recovers its own losses independently, thus the asynchronous packet loss of subflows will prolong the packets' delivery time and consequently result in a high buffer requirement [7], [10], [12]. The buffer size suggested by IETF can only satisfy the retransmission triggered by triple duplicate acknowledgements [16]. In case that a large number of subflows experience the timeout, the receiver may not have enough buffer space to hold all the out-of-order packets [10]. If the buffer overflows, the transmission will be blocked, leading to a dramatic throughput reduction [10], [17].

MPTCP in DCN (DC-MTCP). When the multi-path transmission is applied in DCN, the fundamental problem still exits [2], [8], [12], [18]. The recent work [2] proposes to apply the IETF-MPTCP in DCN. The IETF-MPTCP has published its experimental standard in RFC 6824 in 2013 and implemented its most advanced linux version *v0.91* in August 2016 [19]. In this standard, IETF-MPTCP chooses a best subflow to retransmit a timeout packet, which helps relieve the transmission bottleneck. However, the cross-subflow loss recovery can only be triggered by a timeout [19]. A timeout timer (RTO) is generally as long as 200ms[1], which is a long time especially for the delay-sensitive small flows in DCN [12], [21], [22]. Even if we use a very small timeout duration, the problems associated with asynchronous packet loss exist. Cao et al. explore an explicit multi-path congestion control with the Explicit Congestion Notification (ECN) to satisfy the low latency requirement of small flows [3]. However, it requires special switching hardwares to support the ECN function rather than using current commodity switches in DCN.

[1]Reported by Google [20], reducing the length of RTO will increases the probability of spurious timeouts. Further, a spurious timeout will force a slow start with congestion window reset and thus slow the transfer rate.

MPTCP with network coding. Extensive recent efforts have been made in solving the problem of asynchronous loss in general networks with the coding technology [10], [13]–[15]. Network coding can provide higher throughput and reliability by mixing packets to overcome losses. Li et al. propose to introduce the network coding to parts of the subflows in [14] and apply a systematic coding strategy SC-MPTCP to optimize the buffer delay in [15]. Recent work further introduces the fountain code to encode the packet blocks for relieving the reorder problem [10].

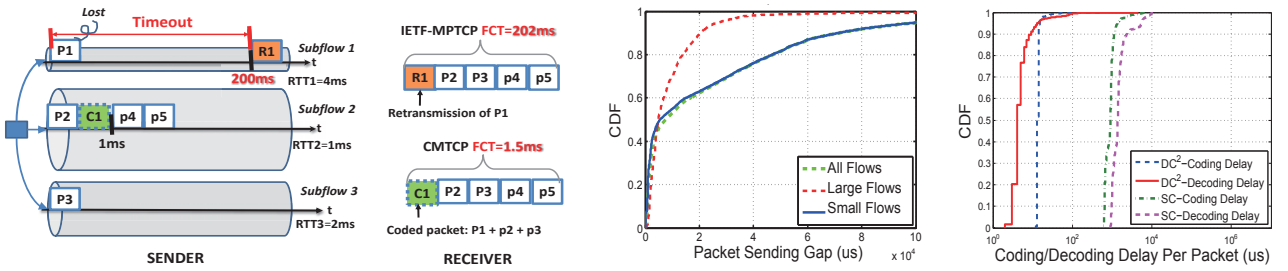
Although the above schemes work well in general networks, their heavy coding overheads cannot meet the application requirements in DCN [12]. On the one hand, a heavy coding overhead will lead to long coding/decoding delay and thus pose great challenges to the transmissions of delay-sensitive small flows. On the other hand, a heavy coding overhead will increase congestions to dynamic traffic in DCN, which decreases the transmission performance of throughput-sensitive large flows. This motivates us to design a practical light-weight coding solution to address the APL problem of multi-path transmissions in DCN with higher performance but little overhead.

III. Architecture Design

A. Motivation

In this section, we will present a simple example to demonstrate how a light-weight coding can effectively address the APL problem, compared to the conventional IETF-MPTCP strategy. Figure 1a shows an example of a short query flow (which is popular in DCN [9]) with 5 packets. It has three subflows with different path quality: round-trip time RTT1 = 4ms, RTT2 = 1ms, and RTT3=2ms. Suppose packet $P1$ in the most congested subflow 1 is lost, then the subflow 1 suffers from a timeout without three duplicate ACKs to trigger the fast retransmission. Since IETF-MPTCP adopts the independent loss recovery, the receiver has to wait for a timeout duration of 200ms for the subflow 1 to send $P1$'s retransmitted packet $R1$. After half of $RTT1$, $R1$ arrives at the receiver side and the flow completes its transmission. In this case, the flow completion time (FCT) is 202ms. Suppose that after sending $P2$, we insert a single coded packet $C1 = P1 + P2 + P3$ into the fastest and least congested subflow 2. After one and a half $RTT2$, the packets $P2, C1, P3, P4, P5$ arrive at the receiver in order. $P1$ will be recovered by the coded packets $C1$ simply after receiving $P2$ and $P3$. Hence the FCT is reduced to 1.5ms.

To explore a practical light-weight coding strategy in DCN, we first analyze the real traffic traces derived from one university data center [23]. The trace files contain about $2GB$ of compressed data collected from two data centers. Figure 1b shows the statistical results. We found that the sending gap Γ_1 in 98% transmissions is larger than 20us. Then we test the running time τ_1 (i.e., the DC^2-Coding Delay in Figure 1c) of encoding one original packet with the use of random linear



(a) Motivation Example (b) Sending Gap in DCN (c) Single Packet's Coding Time in Linux

Figure 1: Motivation

network coding. The coding operations are done over a finite field $GF(2^8)$ of size 256 in our implementation [24]. The length of every symbol is one byte in GF(256). It optimizes the addition as a bitwise xor of two-byte long symbols and uses logarithms to multiply. Through running the test of thousands of times on a server with 4-core 2.5GHz CPU, we obtain the CDF of τ_1 for the packets with the payload length of 1000 bytes in Figure 1c. It shows that, with a probability of 98%, the coding time τ_1 is less than 20us. The above results motivate us to take advantage of the packet sending intervals to complete the coding operations. Specifically, in each packet sending interval, we can have a single original packet encoded. When there is a need to send a coded packet, most original packets are already encoded, and thus it takes at most τ_1 to encode the last packet to send out. In this way, we can achieve a nearly-zero coding delay.

Similarly, at the receiver side, we can also utilize the time gap between receiving packets to perform *progressive* decoding operations after receiving each packet (i.e., the PDA algorithm in Section V), rather than starting the decoding until receiving enough packets. We test the interval Γ_2 between receiving two packets and the decoding time τ_2 in PDA on handling each newly received packet. We find that Γ_2 has a distribution similar to Γ_1 and is larger than 20us in 98% transmissions. Figure 1c shows that with a probability of 97%, the τ_2 (i.e., the DC^2-Decoding Delay) is less than 20us. Therefore, most of the decoding operations can be completed using the coded packets already received, and a receiver needs only τ_2 to perform the remaining decoding operations. PDA thus has a lower decoding delay and can timely deliver the recovered packets to the application.

Recent work in [15] proposes a systematic coding named *SC-MPTCP* to reduce the coding and decoding delay. To compare with our method, we further evaluate its coding and decoding delay per packet under the same setting in Figure 1c. Since SC-MPTCP can not utilize the sending/receiving gaps for efficient encoding/decoding, we can see that the coding and decoding delay of SC-MPTCP is about two orders of magnitude larger than that of our DC^2-MTCP strategy.

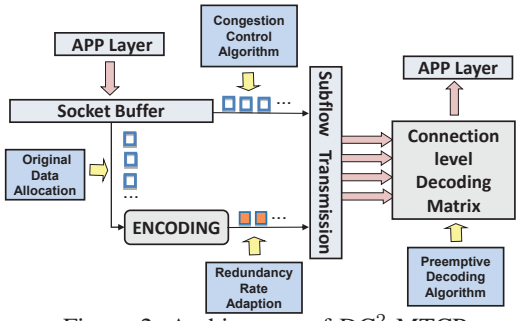


Figure 2: Architecture of DC^2-MTCP

B. Architecture of DC^2-MTCP

Our architecture of DC^2-MTCP is built upon the protocol stack of IETF-MPTCP. Actually, it also has good compatibility with many other MPTCP protocols (e.g., delay-based MPTCP [4]). We add the coding/decoding modules without revising existing stack while adopting the same functions of the MPTCP protocols like path management, subflow interface and congestion control. The sender design of DC^2-MTCP is illustrated in Figure 2 and includes two key modules. The first is the *Original Data Allocation* (ODA) module. By modeling the losses at the connection level, it is responsible for selecting suitable original packets to be coded so that the loss recovery is efficient. We note that only *independent* packets can help recover the loss. If all the original packets used to form the coded one are successfully received, the coded packet is dependent and useless for the loss recovery. By cautiously selecting the packets to encode together, ODA module can essentially increase the effectiveness of coding. The second is the *Redundancy Rate Adaption* (RRA) module, which adjusts the sending rate of coded packets (termed the *redundancy rate*) by modelling the connection-level path quality. A reasonable redundancy rate can effectively recover the losses with a minimal overhead. Moreover, it needs to take the flow characteristics in DCN into account and ensure the priority of delay-sensitive small flows.

To support our coordinated loss recovery mechanism, we add a new DC^2 option in the optional TCP header fields. The protocol statck of IETF-MPTCP uses a 64-bit data sequence number (DSN) at the connection level to

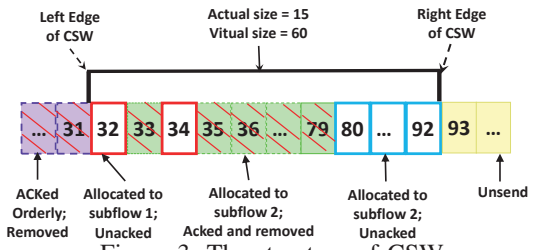


Figure 3: The structure of CSW

number all the data sent over different subflows [25]. We take advantage of this number to ensure that the original data from different subflows can be encoded together and the losses can be recovered across subflows. To ensure a fast coding process, we use the random linear network coding as illustrated in Section III-A. Correspondingly, we use a preemptive decoding algorithm at receiver side to perform the progressive decoding and deliver the recovered packets to the application layer online. To control the header overhead, every coded packet contains C_{max} original packets at most. For a coded packet, its DSN equals the DSN of the first original packet mixed in it. We use 1 byte to denote the number n of original packets mixed in the coded packet and 2 bytes to denote the offset of their DSNs. Specially, when $n = 0$, it denotes this is an original packet. Hence the total size of the DC^2 option is $3n - 1$ bytes.

IV. Coding Design of Sender

In this section, we will provide the detailed sender design in DC^2-MTCP to address the APL problem for multi-path transmissions in DCN. In the following, we will address the three challenges in order: 1) how the sender can sense the APL to help select appropriate original packets to be coded. 2) how to decide the proper redundancy rate; 3) how to perform a time-scattering coding process to reduce the coding delay.

A. Original Data Allocation

There are two kinds of coding process in network coding. The first is *pre-coding* process which performs the Forward Error Correction (FEC) at the packet level. The second is *post-coding* process which performs the Backward Error Correction (BEC) according to the losses already happened. The losses recovered by pre-coding process do not incur any retransmission delay. Since we can not predict the specific losses ideally, and there is a tradeoff between coding redundancy and transmission reliability, there may be some losses after the pre-coding process. When the sender senses any actual losses, it is emergent to perform the post-coding to recover the lost packets in time. Post-coding helps avoid unnecessary overhead introduced into the pre-coding.

In the following, we will illustrate the feasibility for the sender to sense the asynchronous losses based on the disordered ACKs. At the DC^2-MTCP sender side, all the subflows share one data buffer in the kernel at the connection level. When one subflow has room to send a data packet, it picks the data from the buffer sequentially. As a result, the packets with continuous DSNs are orderly allocated to different subflows. Due to the asynchronous losses among subflows, the packets may arrive at the receiver out of order. As a result, the sender will get the disordered ACKs and thus detect the APL. Figure 3 shows an example of connection-level send window (CSW) in a condition of disordered ACKs. There are two subflows in the connection. The CSW stores all the original packets which are sent but not acknowledged. Packets 33 and 35-79 of the subflow 2 are all acknowledged. However, the packets 32 and 34 of the subflow 1 are still not acknowledged. Then the sender can infer that there exist asynchronous losses and the path of subflow 1 currently has low transmission quality. In this case, the receiver must store all the packets with the sequence number larger than 32.

Next, we present a novel metric ψ for the sender to quantify how much APL it senses. Let W_v denote the virtual size of CSW, i.e., the difference between CSW's left bound and right bound. It demonstrates the size of data that the receiver buffer needs to store currently. Let W_s denote the actual size of CSW which contains only the unacknowledged packets in W_v (the acknowledged packets in W_v are already removed from memory). For example, in Figure 3, W_v is 60 and W_s is 15. Then we denote $\psi = W_v - W_s$. Intuitively, the ψ increases when more and more packets of the high-quality paths are blocked in the receiver buffer due to the delayed arrival of only a few packets from the low-quality paths. In the example of Figure 3, the reorder problem becomes worse with larger ψ when more packets sent on subflow 2 before the packets 32 and 34 have arrived or been recovered at the receiver side. On the contrary, if a multi-path connection has similar path qualities for all the subflows, the W_v and W_s will have similar values to keep ψ close to zero.

Since a light APL may be caused by lightly varied RTTs and can be recovered by the flow itself quickly, we set a threshold δ for different data allocation schemes. When $\psi < \delta$, the sender infers that the APL is light enough and we choose the last α unacknowledged packets in the CSW to perform the *pre-coding* process. When $\psi > \delta$, the sender infers that the APL problem is becoming worse and it would be hard for a flow to recover the loss alone. We choose the first β unacknowledged packets in the CSW to perform the *post-coding* process. With the coded packets allocated to different subflows, even if the packets are lost on some low-quality paths, the receiver can still recover the original data and thus the subflows on low-quality paths no longer block the ones on high-quality paths. By testing the value δ in range $[0, 20]$ with simulations, we find that $\delta = 3$ works efficiently to alleviate the APL problem.

Next, we illustrate how to choose a proper value for α and β. Intuitively, since the number of coded packets to insert is kept small in our architecture, we desire α and β to be large enough so that each coded packet encodes more original

packets. However, a too large value for α and β would lead to a long coding delay. To achieve a reasonable low coding overhead, we set the upper bound of α as a controllable constant C_{max}, and in this paper we set C_{max} to 15, the same value as that in [20]. Moreover, since β is used in the case where losses already happen, it desires higher efficiency to recover the lost packets rather than simply encoding more original packets as the pre-coding α intends to. When testing the β value in range of $(0, C_{max}]$, we find that $\beta = \frac{2}{3}C_{max}$ is large enough to efficiently recover the lost in DCNs.

B. Redundancy Rate Adaption

A proper redundancy rate adaption (RRA) is essential to increase the DC^2-MTCP performance in DCN. In this section, we will present the detailed design of RRA and how it performs the flow differentiated control.

1) Redundancy rate adaption for large flows: For large flows in DCN, the asynchronous packet losses will enlarge the required buffer size at the receiver side. For a multi-path transmission with n subflows, when waiting for a fast retransmission on the slowest subflow, the minimal buffer size that can store all the reorder packets is $B_{fr} = \sum_{i=1}^{n} BW_i * \max_{1\leq i\leq n} RTT_i$, where BW_i is the bandwidth of subflow i. The B_{fr} is the product of maximum RTT and the total bandwidth available across all subflows, which is also the suggested receiver buffer size by IETF standard [25]. The required buffer size B_{fr} will grow fast when the bandwidth and subflow number increase. For a multi-path connection with 8 subflows, considering a DCN with bandwidth 10Gbps and maximum RTT 1.5ms, the B_{fr} would be 120Mb which is costly to deploy for each data flow in DCN. If the buffer overflows, the transmission will be blocked [10], [12], resulting in a large throughput reduction. In this case, it is essential to perform the coordinated loss recovery to address the reorder problem.

An effective coordinated loss recovery scheme should fully utilize the light-loaded paths to help compensate for the losses on the heavy-loaded ones. To achieve this goal, DC^2-MTCP adjusts the redundancy sending rate based on the connection-level transmission quality and the subflows' original packet transmission schedule.

For a connection with the loss probability p, the theoretically optimal redundancy rate γ is $(p/(1-p))\lambda$ [10], where λ is the sending rate of original packets. This is because that the successful receiving rate of the original packets is only $(1-p)\lambda$ on average at the receiver side. To achieve the coordinated loss recovery, we schedule the redundancy rate γ_i on the subflow i based on the connection-level loss probability p, rather than the loss probability p_i of the subflow itself. We define the redundancy rate of subflow i as

$$\gamma_1^i = (p/(1-p))\lambda_i \tag{1}$$

where λ_i is the sending rate of the original packets in the subflow i. According to (1), the number of coded packets (i.e., the *redundancy*) allocated on subflow i is proportional to the number of original packets sent through the subflow i. The congestion control algorithm LIA provided in RFC6356 [26] ensures that the subflows on less congested paths send the original packets faster than those from the congested paths. Thus, our DC^2-MTCP design balances the redundancy load in DCN by transmitting more redundant packets through a high-quality path.

Although the equation (1) offers good theoretical properties, it requires an exact knowledge about the packet loss probability at the connection level in real time, which is hard to acquire in current DCNs. Even if great efforts can be made to actively detect the packet loss probability, its value can only be updated after the actual loss has been observed, which can not satisfy the need of our pre-coding. In the following, we present another practical calculation to estimate it based on only the metrics that easy to acquire in real-time (e.g., the window size and RTT of subflows).

Since our DC^2-MTCP is built over IETF-MPTCP stack, it adopts the standard congestion control algorithm LIA. Suppose each subflow i maintains its own congestion window W_i, and let W_{total} denote the sum of the congestion windows of all subflows. The detailed LIA algorithm is as bellow: (1) for each ACK received from the subflow i, increase W_i by $\min(\alpha/W_{total}, 1/W_i)$; (2) for each loss of the subflow i, decrease W_i by $W_i/2$. The α determines the aggressiveness of all the subflows and is a function of the RTT and window of each flow:

$$\alpha = W_{total}\frac{\max_{1\leq i\leq n} W_i/RTT_i^2}{(\sum_{i=1}^{n} W_i/RTT_i)^2}. \tag{2}$$

In order to find the redundant sending rate, we have the following set of derivations. First, based on the network utility model [27], at the network equilibrium point of subflow i with a loss rate p_i, the increase and decrease of the sending rate are equal, and we have

$$\frac{\alpha}{W_{total}}(1-p_i) = \frac{W_i}{2}p_i \tag{3}$$

Further, based on (3), the connection-level loss probability can be calculated as:

$$p = \frac{\sum_{i=1}^{n} W_i p_i}{W_{total}} = \frac{\sum_{i=1}^{n} W_i(\frac{2\alpha}{2\alpha+W_i}W_{total})}{W_{total}} \tag{4}$$

By applying the equation (4) into the original equation (1), we have the redundancy rate for a large flow as

$$\begin{aligned}\gamma_1 &= \sum_{i=1}^{n}\gamma_1^i = \sum_{i=1}^{n}(p/(1-p))\lambda_i \\ &= \{1/(1-\frac{\sum_{i=1}^{n} W_i(\frac{2\alpha}{2\alpha+W_i}W_{total})}{W_{total}}) - 1\}\lambda \end{aligned} \tag{5}$$

The above redundancy rate calculation takes only the real-time information of the window size and RTT of subflows, which are already embedded in the existing protocol stack

of MPTCP and can be directly read with little effort. There are two key features in this design. First, by adjusting the redundancy rate based on the connection-level transmission quality, the losses on different subflows can be recovered jointly with a small ratio of coding packets. Second, for a subflow, we make the redundancy rate set according to the sending rate of original packets. Subflows that have the ability to send more original packets will be allocated more redundant transmissions. It actively schedules the transmissions through less congested subflows to recover the losses from more congested ones.

2) Redundancy rate adaption for small flows: Although conventional IETF-MPTCP has the potential to use multiple paths to improve the throughput for large flows in DCN, it performs worse than single-path TCP for small flows with fewer than 10 packets [2], [8]. Small flows like the query traffic (2KB to 20KB in size) typically complete in just a few RTTs in DCN and always have a small congestion window [2], [8]. Multi-path transmission increases the probability of timeout for such a small flow.

For a subflow i with the congestion window size W_i and path loss probability p_i, the timeout probability P_{to} is equal to the probability without three multiple ACKs [10]:

$$P_{to} = \begin{cases} 1-(1-p)^{W_i} & \text{, if } W_i \leq 3 \\ \sum_{j=0}^{2} C_{W_i}^{W_i-j} p^{W_i-j}(1-p_i)^j & \text{, if } W_i > 3 \end{cases} \quad (6)$$

Based on the LIA algorithm, a small flow does not have enough time to increase its congestion window into a large size and easily gets timeout when losses occur [9]. According to (6), the case becomes worse when the transmissions from a flow are distributed into n sublfows because the congestion window W_i becomes even smaller on average than that of a single-path small flow.

Since small flows often finish in very few RTTs, the CSWs are small and there is little time to sense the real path quality. The conventional path quality metrics like the loss probability and even RTT are difficult to be estimated accurately. Hence we turn to the statistical properties obtained by previous data center traffic measurements. In the recent traffic measurement of a large Google DCN, it is reported that about 10% of the TCP connections incur losses [20]. Furthermore, among the flows with losses, the measurements indicated that most of them experience only a single loss on a burst bottleneck (known as the *single packet tail drop*). Since the above property is related to the single-path TCP flow, we first analyze the relationship between a single-path TCP flow and a multi-path TCP flow. Based on equation (3), we have

$$p_i = \frac{2\alpha}{2\alpha + W_i} W_{total} \quad (7)$$

For a single-path TCP on path i, based on the network equilibrium theory similar to the equation (3), we have

$$\frac{1}{W_i^s}(1-p_i) = \frac{W_i^s}{2} p_i, \quad (8)$$

where W_i^s denotes the congestion window size of the single-path TCP on the path of subflow i at the network equilibrium point.

Based on (2),(7) and (8), we have

$$\sum_{i=1}^{n} \frac{W_i}{RTT_i} \leq \max \frac{W_i^s}{RTT_i} \quad (9)$$

where the ratio of window size and RTT is a conventional path quality metric in IETF-MPTCP.

The equation (9) shows that a multi-path flow does not take up more capacity than that of single-path TCP flows when they share the same bottleneck link. Based on the further fact that losses generally occur at the bottleneck, we conservatively estimate the multi-path TCP flow would have the single-loss property at the connection level, i.e., we have the redundancy rate as $1/W_{total}$ for small flows.

Another practical issue is that it is generally difficult to identify a small flow until it is finished in DCN [28]. As the CSW of small flows are always small, we set the redundancy rate of a small flow as

$$\gamma_2 = \frac{1}{W_{total}}, \quad \text{when} \quad W_{total} \leq 5n \quad (10)$$

where n is the number of subflows. That is, we consider the flow with $W_{total} \leq 5n$ as a small flow because each of its subflows has an average window size less than 5.

So far we have given the settings of redundancy rate for the large flows and small flows respectively. Now we will present the specific procedure to apply them in the coding process. First, we use a variable R to denote the accumulated redundancy based on the redundancy rate. It is initialized as zero. Then the redundancy variable R will increase by γ when any original packet is sent, where $\gamma = \gamma_1$ (i.e., Equ (5)) for large flows and $\gamma = \gamma_2$ (i.e., Equ (10)) for small flows. When reaching a time point that some subflow i sends an original packet to make $R > 1$, a request to send a coded packet on subflow i is generated, and R is updated as $R - 1$.

C. Time-scattering Coding Process

The traditional random linear coding in general networks starts the coding computation only after the coded packet is requested [20], [22], [24], termed the *on-demand* coding scheme. This would introduce a high coding delay when applied in DCN. Since the RTT in DCN is very short (e.g., 250us with empty buffer queue [9]), to make the coding strategy more effective than simple retransmission in DCN, we must strictly control the coding delay. Therefore, we propose a *time-scattering* coding scheme to achieve a nearly-zero delay to send the coded packets when they are requested. The basic motivation is that in every gap between packet sending, we can encode one original packet with specific length into the current coded packet (termed the *unit coding*). Figure 1b and 1c demonstrate that 98% unit coding operations can be completed in the sending gap.

Algorithm 1 Sender Algorithm for DC^2-MTCP

Input: Connectional send window CSW
//Triggered at the beginning of flow transmission
1: $R \leftarrow 0$, $Start \leftarrow False$, $Pre \leftarrow True$
//Triggered by the sending of an original packet P on subflow k
2: Set $\gamma \leftarrow \gamma_2$ based on Equ. (11) if $W_{total} <= 5n$. Otherwise, set $\gamma = \gamma_1$ based on Equ. (6).
3: $R = R + \gamma$
4: **if** $Start$ is $True$ **then**
5: Encode the packet P in C when Pre is true. Or encode the packet P^* before P_{last} in C and set P_{last} as P^* when Pre is false.
6: **else**
7: When $R + \alpha\gamma > 1$ and $\psi \leq \delta$, encode the packet P in C and set $Start$ as true, Pre as true.
8: When $R + \beta\gamma > 1$ and $\psi > \delta$, encode the βth packet P_{last} of CSW in C and set $Start$ as true, Pre as false.
9: **end if**
10: When $R > 1$, send the coded packet C on subflow k, $Start \leftarrow False$, $R \leftarrow R - 1$.

The detailed time-scattering coding process is shown in Algorithm 1. In previous Section IV-B, we introduce how to initialize and update the accumulated redundancy R (line 1-3) so that the request to send a coded packet $C = \sum_{i=1}^{n} \alpha_i P_i$ is generated until $R > 1$ (line 10). Now we present when to trigger the start of the coding process before the arrival of the coding request. Since the original packets P_i in CSW are sent continuously in one RTT, the γ changes little during their sending. Consider the current accumulated redundancy $R < 1$. For the pre-coding process (line 7), if $R + \alpha\gamma > 1$, we need to start coding the last packet of CSW so that after α time gaps, the coded packet will be completed and sent immediately as expected. Similarly, for the post-coding process (line 8), if $R + \beta\gamma > 1$, the coding starts to encode the βth packet of CSW. The setting of α and β are discussed in the previous Section IV-A.

V. Decoding Design of Receiver

In DCN, the receiver needs to perform a light-weight decoding to ensure that the performance benefits outweigh the decoding overhead. The state-of-art decoding algorithm for random linear coding is Gaussian-Jordan elimination [24]. It is designed for general networks without any knowledge about the ratio of coding packets against original packets (i.e., the **C-O** ratio). To address this issue, we design a *preemptive decoding* algorithm (PDA) at the receiver side to effectively reduce the decoding overhead by taking full advantage of the low **C-O** ratio feature in DCN.

At the receiver side, the decoding is performed at the connection level. All subflows share a decoding matrix for a multi-path connection. PDA will start progressive decoding once receive any packet by fully utilizing the receiving gaps. When a new packet arrives, the receiver should first check the coding number n in DC^2 option of the packet head. If $n = 0$, it is an original packet, we should first notify its subflow to generate corresponding ACK. Then we take it into the connection-level decoding matrix to help decode the coded ones. If $n > 0$, it is a coded packet. Next, there are three steps of PDA at the receiver side. The first and the third step are the same as those of the Gauss-Jordan elimination. The second step is different when an original packet arrives. At this step, the original packet replaces its own row in a preemptive way if the row is occupied by a combination C. Then the combination C will perform like a newly arrived one as in the process of Gauss-Jordan elimination to transform the decoding matrix into the reduced row echelon form [22]. Once a coded packet recovers a single loss at the connection level, more packets in the receiver buffer can be delivered to the application layer immediately in order.

For a full-rank decoding matrix $\mathcal{M}$ with m original packets and l coded packets, the decoding time complexity of Gauss-Jordan elimination is $O((m+l)^2)$ [24]. Rather than performing elimination on each packet (either a coded packet or an original packet) as Guass-Jordan elimination, our PDA requires no elimination for the original packets while keeping the same number of elimination operations on the coded packets as Gauss-Jordan elimination. Hence PDA will reduce the decoding time complexity to $O(l(m+l))$, i.e., the time complexity of PDA is reduced to $\frac{l}{m+l}$ fraction that of the Gauss-Jordan elimination. In the evaluation section, we will show that $l < 0.05m$, i.e., the reduced fraction (i.e., $\frac{m}{m+l}$) of decoding time complexity can be achieved above 95%.

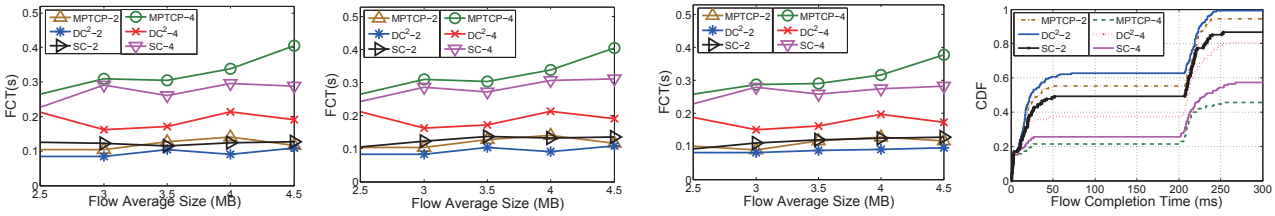
VI. Evaluation

In this section, we implemented DC^2-MTCP and other existing MPTCP schemes in the NS-2 simulation platform. We first present the methodologies of our evaluations and then analyze the simulation results.

A. Evaluation Setup and Methodologies

We simulate the DCN topology as a commonly used 3-layer Fat-Tree with 8 pods [3], where the link bandwidth is set as 1Gbps. The network accommodates totally 128 hosts, where each pair of host has 16 equal-length shortest paths. The default routing function in NS2 adopts a single-path routing for each flow. However, the multiple-path transmission in our study requires each flow to be split into several subflows and route on multiple different paths. To support the ECMP routing protocol for multiple-path transmissions in NS2, we add a hash function in the packet forwarding class *Classifier* to hash the IP head plus the subflow ID of the packets received at each node to one of its next available hops.

The recent work SC-MPTCP [15] proposes a systematic coding to reduce the coding overhead when integrating the coding strategies into MPTCP. Although it is designed for general IP networks, it shows a lower coding overhead than other existing non-systematic coding methods in MPTCP [15]



(a) Overall: Avg FCT (b) Small flow: Avg FCT (c) 99-th percentile: FCT (d) CDF of FCT

Figure 4: Average FCT with random pattern

and is thus more likely to work for DCN. Hence we implement SC-MPTCP to compare with our method and also use IETF-MPTCP to serve as the baseline strategy. For comparative analysis, we evaluate the flow performance of all the three schemes under two traffic patterns. First, to test the flow performance in the average traffic case, we derive 30 random flows from the real traffic traces provided in [23] to serve as the simultaneously running MPTCP flows (see more details about the trace in Section III-A). We simulate the random traffic pattern by adding 128 random TCP flows to serve as the background traffic in the network, where each of them is randomly attached to a host. Based on previous measurements [9], the data center traffic usually consists of delay-sensitive short messages (100KB to 1MB) and throughput-sensitive long flows (1MB to 100MB). Hence we analyze the flow completion time (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.

Second, to test the flow performance under the popular *hot-spot traffic patten* [6] in DCN, we add a small portion of the 100MB MPTCP flows to serve as the *hot flows*. We further add random failures among the subflows to evaluate the robust flow performance on handling the transmission failures (e.g., caused by link/node failures) among the subflows. The coding overhead is evaluated in all the above cases. All the simulations are running ten rounds to obtain the average values for each point in the evaluation figures. We equally scale the average flow traffic size to simulate the varying network loads.

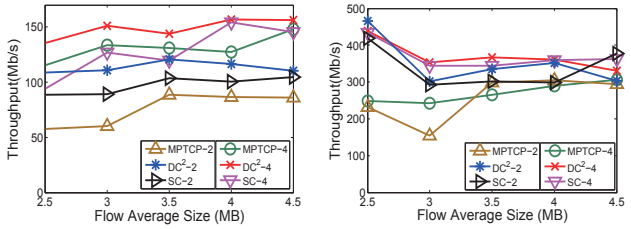
B. Random Traffic Pattern

Flow completion time. Fig. 4 shows the FCT performance under various average flow sizes. Overall, we can see that DC^2-MTCP obtains lower FCT than both the SC-MPTCP and IETF-MPTCP under different cases with the same subflow number. Specifically, compared to MPTCP-2 (i.e., IETF-MPTCP with 2 subflows), the average FCT of DC^2-2 (i.e., DC^2-MTCP with 2 subflows) is reduced by about 25% while that of DC^2-4 is reduced by about 43%. On the other hand, the FCT of SC-MPTCP (i.e., SC-2 and SC-4) is on average only 10% lower than that of IETF-MPTCP with the same subflow number. Moreover, we can see that for each scheme, the case with 2 subflows has lower FCT than that with 4 subflows. This is because most of the flows in the traces are small flows[1]. For a small flow, when the number of subflows increases, its flow completion time becomes longer since it is more likely to suffer from the asynchronous losses. Since the SC-MPTCP does not distinguish between small flows and large flows, it only relies on the estimation of loss rate to adapt the redundancy, which however, is not accurate for small flows (Section IV-B2). Instead, DC^2-MTCP detects the path quality by monitoring the real-time CSW status. The above improvements illustrate the effectiveness of DC^2-MTCP in adapting the coding redundancy when the CSW of flow is small (Section IV-B2), which alleviates the problem of asynchronous losses in the subflows of each small flow.

Flow performance with timeouts. An interesting finding is that in Fig. 4d, each CDF curve has two distinct jumps with an interval of about 200ms. This is due to the timeout duration is 200ms, and the flows without any timeout are all completed within 50ms. Compared to MPTCP-2, DC^2-2 has 5% more flows that are completed without any timeout while SC-2 performs the worst with 5% fewer flows completed. Similar difference between them can be found for the flows completed within one timeout. In the case with 4 subflows, DC^2-4 has 18% more flows without timeout than MPTCP-4, while about 80% flows in DC^2-4 are completed within one timeout while such flows only take 45% in MPTCP-4 and 58% in SC-4. SC-MPTCP fails to allocate enough coding redundancy for small flows because it has a hard time to estimate accurate loss rate when the flow is small. It demonstrates the effectiveness of the coding strategy in DC^2-MTCP to achieve the joint loss recovery among different subflows, especially for the transmission cases with more subflows.

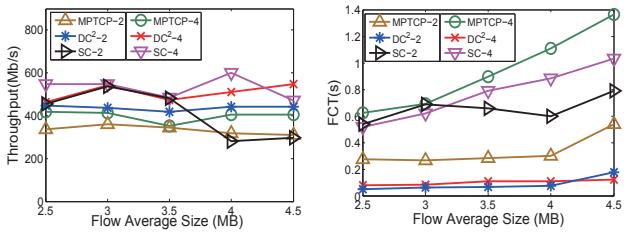
Flow throughout. Fig. 5 shows the throughput performance with different flow sizes. We can see that DC^2-MTCP achieves on average 30% higher throughput than IETF-MPTCP and 15% higher throughput than SC-MPTCP. For large flows, DC^2-4 achieves about 36% higher throughput than MPTCP-4 while DC^2-2 achieves about 28% higher throughput than MPTCP-2. Correspondingly, SC-MPTCP has a 5% lower throughput than DC^2-MTCP. Compared

[1]Based on previous measurements of data center traffic [5], [6], [9], the small flows take the majority of all the DCN flows.



(a) Overall: Throughput (b) Large flow: Throughput

Figure 5: Average throughput with random pattern



(a) Large flow: Throughput (b) Small flow: Avg FCT

Figure 6: Average performance with hot-spot pattern

to the small flows, SC-MPTCP can estimate the loss rate more accurately for large flows, and the unified redundancy setting of SC-MPTCP increases the aggressiveness of large flows. It further demonstrates the advantage of applying the distinguished redundancy adaptation for large flows (Section IV-B1) in DC^2-MTCP to address the reordering issues when experiencing asynchronous losses among the subflows.

C. Hot-spot Traffic Pattern

As Fig. 6 shows, the large flows in DC^2-MTCP achieve on average 30% higher throughput than those in MPTCP. For small flows, the FCT of DC^2 keeps more stable than that of MPTCP and SC-MPTCP when the average flow size increases. Specially, DC^2-2 reduces the FCT by about 75% than MPTCP-2, while DC^2-4 reduces the FCT by about 90% than MPTCP-4. SC-4, however, reduces the FCT by only 10% than MPTCP-4, while SC-2 even doubles the FCT of MPTCP-2. The small flows in SC-MPTCP not only suffers from the congestions caused by hot flows but also the allocation with very little redundancy because most of the redundancy is allocated for hot flows. In contrast, DC^2-MTCP can detect and adapt the coding redundancy to make up the losses online distinguished for the large flows and small flows whenever the losses are caused by the congestions of hot flows or the link failures. Finally, IETF-MPTCP will take a long time after several timeouts to detect the link failure and thus can not make up the losses timely.

Table I: C-O ratio under different traffic patterns

	All-2	Small-2	Large-2	All-4	Small-4	Large-4
Random	0.013	0.02	0.002	0.026	0.04	0.011
Hot-spot	0.0002	0.02	4.19E-05	0.0008	0.04	0.0003

D. C-O Ratio

Table I shows the average ratio between the number of coded packets and the original packets (i.e., the C-O ratio) in DC^2-MTCP. We can see that it has an average C-O ratio of 0.02 under random traffic pattern and 0.005 under the hot-spot pattern. As the comparison, conventional non-systematic coding in MPTCP requires a C-O ratio at 100% [10], while the systematic coding utilized in SC-MPTCP keeps an average C-O ratio about 0.02 under all the cases. Since SC-MPTCP generates two orders of magnitude higher coding delay per packet than DC^2-MTCP (see Fig. 1c), the total coding overhead (in terms of the product of the number of coding packets to insert and the coding delay per packet) of SC-MPTCP is two orders of magnitude larger than that of DC^2-MTCP in the case of random traffic pattern, and three orders of magnitude larger than that of DC^2-MTCP for the hot-spot traffic pattern. While obtaining the highest flow performance in both patterns, such a low C-O ratio helps our DC^2-MTCP solution to run with a low coding overhead. Among all of flows, the average C-O ratio of large flows in DC^2-MTCP is 0.004, which is one magnitude smaller than that of small flows. We can see that the smaller the flow size is, the higher the C-O ratio our coding scheme requires. This is due to our flow differentiated scheme (Section IV-B) in DC^2-MTCP intends to provide higher priority for small flows and thus ensure their high FCT performance.

VII. Conclusion

In this paper, we propose a light-weight coding based multi-path transmission scheme to efficiently address the asynchronous packet losses problem in DCN. At the sender side, we first detect the online losses by monitor the CSW to select the proper packets to be coded. Then we provide a flow differentiated scheme to adapt the coding redundancy for large flows and small flows respectively according to the online path quality. Finally, we propose a time-scattering coding strategy by fully exploiting the packet sending intervals to reduce the coding delay. At the receiver side, we propose a preemptive decoding algorithm to efficiently reduce the decoding overhead. The evaluations show that on average only <5% coded packets are inserted to coordinately recover the losses across subflows in our scheme while it can reduce on average 40% FCT for small flows and improve about 30% throughput for large flows in DCN.

VIII. Acknowledgment

This work is supported in part by National Science Foundation of China (Grant No. 61303250, 61401070), and the Scientific Research Foundation of the Institute of Information Engineering, Chinese Academy of Sciences (Grant No. Y6Z0011105). The corresponding author of this paper is Yan Zhang.

REFERENCES

[1] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "Bcube: a high performance, server-centric network architecture for modular data centers," in *SIGCOMM*, 2009.

[2] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multi-path tcp," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 266–277, 2011.

[3] Y. Cao, M. Xu, X. Fu, and E. Dong, "Explicit multi-path congestion control for data center networks," in *CoNEXT 2013*.

[4] X. Fu, M. Xu, and Y. Cao, "Delay-based congestion control for multi-path tcp," in *ICNP 2012*.

[5] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild." in *Acm Sigcomm Conference on Internet Measurement*, 2010, pp. 267–280.

[6] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *IMC 2009*.

[7] K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, and A. Akella, "Presto: Edge-based load balancing for fast datacenter networks," in *SIGCOMM 2015*.

[8] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz, "Detail: Reducing the flow completion time tail in datacenter networks," *Acm Sigcomm Computer Communication Review*, vol. 42, no. 4, pp. 139–150, 2012.

[9] 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.

[10] Y. Cui, L. Wang, X. Wang, H. Wang, and Y. Wang, "Fmtcp: A fountain code-based multi-path transmission control protocol," *IEEE/ACM Transactions on Networking (ToN)*, vol. 23, pp. 465–478, 2015.

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

[12] J. Zhang, F. Ren, and C. Lin, "Survey on transport control in data center networks," *IEEE Network*, vol. 27, no. 4, pp. 22–26, 2013.

[13] Y. Hwang, B. O. Obele, and H. Lim, "Multi-path transport protocol for heterogeneous multi-homing networks," in *CoNEXT 2010*.

[14] M. Li, A. Lukyanenko, and Y. Cui, "Network coding based multi-path tcp," in *IEEE Conference on Computer Communications Workshops*, 2012, pp. 25–30.

[15] M. Li, A. Lukyanenko, S. Tarkoma, Y. Cui, and A. Ylä-Jääski, "Tolerating path heterogeneity in multipath tcp with bounded receive buffers," *Computer Networks*, vol. 64, pp. 1–14, 2014.

[16] A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar, "Architectural guidelines for multi-path tcp development," *IETF RFC*, 2011.

[17] K. C. Leung, V. O. K. Li, and D. Yang, "An overview of packet reordering in transmission control protocol (tcp): Problems, solutions, and challenges," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 18, no. 4, pp. 522–535, 2007.

[18] S. Sen, D. Shue, S. Ihm, and M. J. Freedman, "Scalable, optimal flow routing in datacenters via local link balancing," in *CoNEXT 2013*.

[19] "http://multipath-tcp.org/pmwiki.php?n=Main.Release91," 2016.

[20] 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.

[21] 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," *Acm Sigcomm Computer Communication Review*, vol. 39, no. 4, pp. 303–314, 2009.

[22] D. T. S. Z. Z. X. Jiyan Sun, Yan Zhang and J. Ge, "Improving tcp performance in data center networks with adaptive complementary coding," in *40th Annual IEEE Conference on Local Computer Networks (LCN 2015)*.

[23] T. Benson, A. Akella, and D. A. Maltz, "http://pages.cs.wisc.edu/~tbenson/IMC10_Data.html," *IMC*, 2010.

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

[25] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "R-fc6824: Tcp extensions for multi-path operation with multiple addresses," 2013.

[26] C. Raiciu, M. Handley, and D. Wischik, "Rfc 6356, coupled congestion control for multi-path transport protocols," 2011.

[27] S. H. Low, "A duality model of tcp and queue management algorithms," *IEEE/ACM Transactions on Networking (ToN)*, vol. 11, no. 4, pp. 525–536, 2003.

[28] G. Wang, D. G. Andersen, M. Kaminsky, K. Papagiannaki, T. Ng, M. Kozuch, and M. Ryan, "c-through: Part-time optics in data centers," in *SIGCOMM 2010*.