An Incast-Coflow-Aware Minimum-Rate-Guaranteed Congestion Control Protocol for Datacenter Applications

Zhijun Wang
The University of Texas at Arlington
Arlington, USA
zhijun.wang@uta.edu

Yunxiang Wu
Purple Mountain Laboratories
Nanjing, China
wuyunxiang@pmlabs.com.cn

Rosenkrantz Stoddard
The University of Texas at Arlington
Arlington, USA
todd.rosenkrantz@uta.edu

Ning Li
The University of Texas at Arlington
Arlington, USA
ning.li@uta.edu

Minh Nguyen
The University of Texas at Arlington
Arlington, USA
mqnguyen@mavs.uta.edu

Hao Che
The University of Texas at Arlington
Arlington, USA
hche@cse.uta.edu

***Abstract*—Today's datacenters need to meet service level objectives (SLOs) for applications, which can be translated into deadlines for (co)flows running between job execution stages. As a result, meeting (co)flow deadlines with high probabilities is essential to attract and retain customers and hence, generate high revenue. To fill the lack of a transport protocol that can facilitate low (co)flow deadline miss rate, especially in the face of incast congestion, in this paper, we propose DCMRG, an incast-coflow-aware, ECN-based soft minimum-rate-guaranteed congestion control protocol for datacenter applications. DCMRG is composed of two major components, i.e., a congestion controller running on the send host and an incast congestion controller running on the receive host. DCMRG possesses three salient features. First, it is the first congestion control protocol that integrates congestion control with coflow-aware incast control while providing soft minimum flow rate guarantee. Second, DCMRG is readily deployable in datacenter networks. It only requires software upgrade in the hosts and minimum assistance (i.e., ECN) from in-network nodes. Third, DCMRG is backward compatible with and, by design, friendly to the widely deployed, standard-based transport protocols, such as DCTCP. The results from large-scale datacenter network simulation demonstrate that in the absence of incast congestion, DCMRG can reduce flow deadline miss rates by 3x and 1.6x compared to D^{2}TCP and MRG, respectively. Moreover, DCMRG further reduces the coflow deadline miss rate by more than 40% and 60% and lowers the packet drop probability by 60% and 80%, in the face of incast congestion, compared to D^{2}TCP with ICTCP and MRG with ICTCP, respectively.**

***Index Terms*—Congestion control, incast congestion, datacenter, minimum rate guarantee**

I. Introduction

Today's datacenter applications can be broadly classified into two categories, i.e., user-facing interactive applications and background batch applications. While background batch applications, such as big data analytics, may have to meet a mean-latency Service Level Objective (SLO) [1], user-facing interactive applications, such as web searching and social networking, generally call for tail-latency SLO guarantee [2], [3]. A major challenge to satisfy SLOs for datacenter applications is how to ensure that given (co)flow deadlines are met. The majority of the datacenter applications are scale-out by design. Namely, each job may involve multiple rounds of parallel task processing by a group of machines and the intermediate results, in the form of a collection of flows, known as a coflow [4], to be sent to another group of machines serving the next round of parallel task processing. Oftentimes, the next round cannot start until all the flows in the coflow of the current round finishes. Hence, to meet a given job SLO, all the associated coflows must meet their respective coflow deadlines. As coflows often cause incast congestion, i.e., multiple flows from a coflow fan-in at a machine in a burst, a key challenge is how to meet coflow deadlines in the presence of incast congestion [5], [6].

To meet coflow deadlines or minimize coflow completion time (CCT), the existing coflow scheduling solutions working at the application layer, e.g., [4], [7]–[10], generally call for explicit flow rate allocation for all the flows in a coflow. This, however, must be done by an underlying transport layer protocol, with or without the assistance of the IP layer (i.e., the network nodes).

To provide flow rate/deadline guarantee, some transport protocols with significant assistance of the IP layer have been developed, e.g., based on priority queuing [11] or clean-slate, custom solutions [12]–[15]. As a result, they may incur substantial queuing parameter tuning costs and/or require new switch hardware design. A notably example of flow-deadline-aware transport protocol that require minimum assistance from the network nodes is D^{2}TCP [16], based on single-bit marking for explicit congestion notification (ECN) [17]. However, D^{2}TCP may suffer from high flow deadline miss rate even at modest load, especially in the presence of incast congestions. To the best of our knowledge, the only end-to-end

978-1-5386-5541-2/18/$31.00 ©2018 IEEE

(i.e., without the assistance of the IP layer) transport protocol that provides soft minimum flow rate guarantee is MRG [18]. MRG is backward compatible with TCP, i.e., it degenerates to the TCP Reno, when the minimum guaranteed rate is set to zero. However, it inherits all the drawbacks of TCP when applied to datacenter networks.

Meanwhile, the existing solutions that address incast congestion, e.g., [5], [6], [19]–[21], exclusively focus on how to improve link utilization for elastic flows, not providing coflow deadline guarantee. In fact, none of the above transport protocols address the impact of incast congestion on meeting the coflow deadlines.

To fill the lack of a transport protocol that can facilitate effective flow rate allocation for deadline-aware (co)flow scheduling, especially in the presence of incast congestion, in this paper, we put forward an incast-and-coflow-deadline-aware transport protocol, called DataCenter transport with soft Minimum Rate Guarantee (DCMRG). DCMRG is composed of a congestion controller running on the send host side and a coflow-aware incast congestion controller on the receive host side. DCMRG possesses following desirable features:

- It is the first transport protocol that integrates the congestion control with coflow-aware incast control. In particular, it provides software minimum rate guarantee and (co)flow-deadline-aware flow rate allocation in the face of incast congestion, one of the most challenging issues for datacenter network flow rate allocation;
- It is highly scalable and readily deployable in datacenter networks without (i.e., end-to-end) or with minimum assistance (i.e., ECN) from the IP layer. Its implementation only requires software upgrade at hosts and does not require any in-network node software/hardware modification.
- It is backward compatible with TCP, DCTCP and MRG. Specifically, in the absence of incast control, DCMRG degenerates to MRG, when ECN is turned off; to DCTCP, when the minimum guaranteed flow rate is set to zero; and to TCP, when ECN is turned off and the minimum guaranteed flow rate is set to zero.

We conduct the evaluation of DCMRG based on large-scale datacenter network simulation. The simulation results demonstrate that in the absence of incast congestion, DCMRG can achieve 3x and 1.6x lower flow and coflow deadline miss rate, while offering comparable average flow completion time (FCT) for flows without deadline, compared to D^{2}TCP and MRG, respectively. Moreover, DCMRG further reduces the coflow deadline miss rate by more than 40% and 60% and lowers the packet drop probability by 60% and 80%, in the presence of incast congestion, compared to D^{2}TCP with ICTCP [6] and MRG with ICTCP, respectively.

II. DCMRG CONTROLLERS

Datacenter networks are significantly different from the public Internet. The round trip time (RTT) for a datacenter network is usually much smaller (less than 200 μs versus tens to hundreds of milliseconds in the public Internet). As a result, a data flow may consume a huge amount of bandwidth at very low packet latency. TCP is particularly ineffective in such a high-bandwidth-low-latency environment, causing excessive packet queuing delay and packet losses, especially in the presence of incast congestion [22].

On one hand, DCTCP [22] manages to address the above shortcomings of TCP by employing the ECN mechanism for earlier congestion detection, allowing switches to maintain short queues and hence, reducing packet latency. However, DCTCP is flow deadline unaware. On the other hand, MRG [18] improves over TCP by incorporating minimum-rate-guaranteed features. However, as an end-to-end transport protocol like TCP, it inherits all the above shortcomings from TCP. Although flow deadline aware and ECN-based, D^{2}TCP [16] is found to give high flow deadline miss rate, even at modest load. Moreover, these protocols do not provide incast congestion control and are coflow-unaware. To fill the lack of a transport protocol that can facilitate low (co)flow deadline miss rate, especially in the presence of incast congestion, and that is readily deployable in any existing datacenter networks, in what follows, we develop DCMRG, a transport protocol that integrates congestion control and coflow-aware incast control for low (co)flow deadline miss rate and that requires minimum assistance of the IP layer. In the following, we first briefly introduce the utility-based optimal control law and then derive the DCMRG congestion controller, and finally design the DCMRG incast controller.

A. Optimal Control Law

Both DCMRG and MRG are underpinned by the work in [23], which solves the network utility maximization (NUM) problem for any concave user utility functions in the form of a family of fully distributed flow control laws. In the follows, we first summarize the results in [23].

For any given concave utility function, $U_i(x_i)$, of flow rate x_i for user flow i, for $i = 1, 2, ..., N$, where N is the total number of active flows, the NUM problem can be stated as follows,

$$max \sum_{i}^{N} U_i(x_i) \tag{1}$$

subject to the link bandwidth constraints for all links in the network and the minimum flow rate constraints for inelastic flows, i.e., flows with minimum rate, or equivalently, deadline requirements (the equivalence of the two will be explained later). In general, NUM solutions can lead to unfair/unpredictable flow rate allocation where different flows may take on different user utilities, i.e., $U_i(x_i)$ may be different for different flows x_i. Our recent work [24] proposes a variation of NUM framework that addresses this issue. In this paper, however, we limit ourself to the case where all flows share a single TCP utility function, to be defined shortly. The optimal control law that maximizes the system utility for any given flow x (the subscript i is skipped for simplicity), satisfying Eq.(1) is given as follows,

$$\dot{x} = z(x, t, cg)[f(x) - (1 - \overline{cg}r(x))] \tag{2}$$

with

$$f(x) = 1 - e^{-\partial U(x)/\partial x}, \quad (3)$$

here cg is the path congestion indicator, taking value 1, if the path is congested, and 0, otherwise. $\overline{cg}$ is the logical negation of cg. $z(x,t,cg)$ can be any positive piecewise continuous scaler function. $r(x)$ is a scalar parameter associated with the minimum rate requirement. Assume that a flow has a minimum rate requirement, i.e., $x \geq \theta$, then $r(x)$ is given as follows,

$$r(x) \quad = \quad \begin{cases} 1 & if \quad x \geq \theta \\ r_{cos} & if \quad x < \theta \end{cases} \quad (4)$$

with $r_{cos} > 1$, a design parameter. We select $r_{cos} = 3$ in throughout the paper based on our analysis and simulation results (not shown in the paper due to lack of space).

The above optimal control law for each flow, x, is a distributed control law, because it is only dependent on the utility function, $U(x)$, of the flow, and the only nonlocal information needed as input is a single bit, cg, indicating whether the flow forwarding path is congested or not. Clearly, this information may be either source inferred, as in the case of end-to-end TCP or explicitly conveyed with minimum assistance from the IP layer, e.g., via ECN. Moreover, since $z(x,t,cg)$ is an arbitrary piecewise continuous positive scaler function, the control law above represents, in fact, a family of control laws that solve the NUM problem.

As discussed in detail for MRG in [18], [25], a user utility function $U(x)$ is in general composed of an inelastic, convex lower part and an elastic, concave upper part separated by an inflection point $x = \theta$. For example, for adaptive video streaming, θ may represent the minimum video encoding rate below which the user utility quickly diminishes. As a result, allocating at least θ amount of bandwidth to a flow is essential to the longevity of the video service. Any additional rate allocation further increases the user utility, but is no longer essential, just like any other concave user utilities for elastic applications whose elastic flows may be well served by TCP. So an important decision made in the design of MRG is to let $x \geq \theta$ be the flow constraint in the NUM problem and $U(x)$ be the TCP utility to be shared by all the flows. By doing so, all the flows, including TCP flows, will share network resources fairly, provided that the minimum rate requirements, i.e., $x \geq \theta$, are met.

TCP utility function is derived by reverse engineering the TCP Reno control law in [18]. Let ωx and μ be the multiplicative and additive increase rates, and βx be the multiplicative decrease rate of TCP, respectively. The TCP utility function can then be derived from Eq. (2), which is given as follows: in the slow start phase,

$$U(x) = xlog(1 + \omega/\beta), \quad (5)$$

and $z(x,t,cg)$ given as,

$$z(x,t,cg) = z_{tcp}(x,t,cg) = (\omega + \beta)x, \quad (6)$$

and, in the congestion avoidance phase,

$$U(x) = (\mu/\beta + x)[log(\mu + \beta x) - 1] - x[log(\beta x) - 1], \quad (7)$$

with $z(x,t,cg)$ given as,

$$z(x,t,cg) = z_{tcp}(x,t,cg) = \mu + \beta x, \quad (8)$$

B. DCMRG congestion controller

The congestion controller in DCMRG runs on the send host. In Eq. (2), $z(.)$ is an arbitrary positive piecewise continuous scalar function. The actual function selected determines how well the control law can track the optimal operational point, an ever moving target in a dynamically changing network environment. Since DCTCP is able to address the shortcomings of TCP by incorporating ECN, we adopt the similar idea to improve over MRG. In what follows, we show that this can be done by selecting a different control law in the same family of control laws that MRG belongs to, i.e., through a specific selection of $z(.)$ only. More specifically, we use α defined in DCTCP (i.e., the fraction of packets that are marked in one RTT) as a measurement of the degree of congestion and define $z(.)$ as,

$$z(x,t,cg) = z_{tcp}(x,t,cg)(\overline{cg} + \alpha cg). \quad (9)$$

The defined $z(.)$ function is discontinuous at instants the congestion status change (i.e., from congestion to non-congestion and vice versa). As in practice, the congestion status cannot change until the end of each RTT epoch, $z(.)$ thus defined is indeed a piecewise continuous positive function. Substituting $z(.)$ above and the TCP utility function in Eqs. (5) and (7) into Eqs. (3) and (2), we arrive at the DCMRG congestion controller as follows,

$$\dot{x} \quad = \quad \begin{cases} r(x)\omega x + (r(x) - 1)\beta x & if \quad cg = 0 \\ -\alpha\beta x & if \quad cg = 1, \end{cases} \quad (10)$$

for the slow start phase and,

$$\dot{x} \quad = \quad \begin{cases} (r(x) - 1)\omega x + \mu r(x) & if \quad cg = 0 \\ -\alpha\beta x & if \quad cg = 1, \end{cases} \quad (11)$$

for the congestion avoidance phase. In the case where in-network nodes are not ECN-capable, α=1 and this DCMRG congestion controller degenerates to MRG. The DCMRG congestion controller possesses some salient features as we now explain.

First, We note that DCMRG degenerates to DCTCP at $r(x)$=1, i.e., θ=0 (for the ease of comparison, the readers may also refer to the window-based DCMRG presented in the next subsection). This is true because the rate decrease cases in Eqs. (10) and (11) are the same as the one in DCTCP and the rate increase case is also the same as DCTCP, which coincides with TCP [22]. The performance improvement of DCMRG over MRG or DCTCP over TCP, in practice, comes from faster and finer-grained adaptation to congestion via ECN signaling.

Second, since the DCMRG congestion controller degenerates to MRG at α=1, when ECN is turned off (i.e., using source inferable information only), which in turn, degenerates to TCP at $r(x)$=1, DCMRG is a parent control law from which the other three control laws can be "derived" as special cases. In other words, DCMRG is backward compatible with MRG, DCTCP and TCP.

C. Window based DCMRG congestion controller

The DCMRG congestion controller presented above is fluid-flow based. Now we give its window-based version, as the window-based version is easier to implement and backward compatible with MRG, DCTCP and TCP. First, to apply the DCMRG congestion controller to a flow with a deadline, T_d. T_d must be mapped to a minimum flow rate, θ. Following [11], [14], for a flow with deadline, assume that the flow size, S_f, is a given. Then, we have, $\theta = S_f/T_d$. For window-based flow control, the flow rate is considered as a constant within a RTT, hence the minimum required rate θ can be transformed into a minimum requested window size W_{min} as,

$$W_{min} = \frac{\theta RTT}{MSS} = \frac{S_f RTT}{T_d MSS}, \tag{12}$$

where MSS is the maximum segment size. For a flow without deadline, W_{min}=0.

Next, we need to determine ω, β and μ. These parameters are the coefficients of multiplicative increase rate and multiplicative decrease rate, respectively, for TCP Reno. In window based TCP, the rate is a constant within each RTT, and its TCP congestion window size is doubled and reduced by half every RTT without and with congestion, respectively. So ω and β can be approximated as $1/RTT$ and $1/2RTT$, respectively. μ is the rate corresponding to 1 packet increase per RTT, i.e., $\mu = 1 \times MSS/RTT$. The rate change in a RTT is $\Delta x = \dot{x}RTT$. In the window based protocol, the window size change ΔW_c is calculated as $\Delta W_c = \Delta x RTT/MSS$, and the congestion window size (W_c) adjustment is $W_c = W_c + \Delta W_c$.

With the above preparation, now we can write the window update algorithm for the congestion window, W_c, after each RTT in DCMRG as follows: in the slow start phase,

$$W_c = \begin{cases} (\frac{3}{2} r_{cos} + \frac{1}{2}) W_c & if\ cg = 0\ \&\ W_c < W_{min} \\ 2W_c & if\ cg = 0\ \&\ W_c \geq W_{min} \\ W_c(1 - \alpha/2) & if\ cg = 1, \end{cases} \tag{13}$$

and in the congestion avoidance phase,

$$W_c = \begin{cases} r_{cos} W_c + 1 & if\ cg = 0\ \&\ W_c < W_{min} \\ W_c + 1 & if\ cg = 0\ \&\ W_c \geq W_{min} \\ W_c(1 - \alpha/2) & if\ cg = 1. \end{cases} \tag{14}$$

From these equations, we know that the flow increase rate is determined by W_{min} which is associated with flow deadline. For a flow with a given deadline and flow size, W_{min} is given. For a flow with deadline but unknown flow size, e.g., when a flow starts, while the task execution for the task associated with the flow is still producing more data to be transmitted, the minimum rate can be set at the data generation rate or the flow is treated as an elastic flow by setting $W_{min} = 0$ until all the data are generated, when the W_{min} is updated and enforced.

D. DCMRG incast controller

The congestion caused by many-to-one traffic is called incast congestion [6], [20]. As datacenter applications are predominantly scale-out by design, how to meet incast coflow deadlines in the face of incast congestion becomes a critical challenge. For instance, even with only 10% flow deadline miss rate, the coflow deadline miss rate for a coflow with only 10 flows in it will reach 65%! So it is imperative to further enhance flow-deadline-aware transport protocols to enable coflow-deadline-aware control for fan-in coflows. Unfortunately, the existing solutions that address incast congestion, e.g., [5], [6], [19]–[21], exclusively focus on how to improve link utilization for elastic flows, not providing coflow deadline guarantee. In this section, we introduce the DCMRG incast controller running on the receive host that aims at providing high fan-in coflow deadline guarantee, while preserving and/or enhancing all the nice features of the DCMRG congestion controller. Again, DCMRG can serve as a transport layer flow rate allocation protocol to facilitate (co)flow scheduling for the existing [4], [7]–[10] and future (co)flow scheduling solutions.

The DCMRG incast controller not only makes DCMRG coflow-deadline-aware but also help improve the convergence speed of DCMRG in tracking the optimal operational point where the sum of TCP-based user utilities is maximized. The DCMRG incast controller only involves software upgrade at the end hosts, requiring no additional assistance from the network nodes.

As the sending window size (W_{send}) in the send host is the minimum of congestion window size (W_c) and receive window size (W_{rcv}), the idea is to regulate the receive window size to bound flow rate. Although the idea of the using receive window size to bound flow rate is not new, e.g., ICTCP [6], none of the existing solutions that deal with incast congestion can provide minimum rate guarantee. Specifically, let x denote the flow rate bound for a given incoming flow. Then the receive window size, W_{rcv} is given by $W_{rcv} = xRTT/MSS$. However, as RTT is usually measured at the send host and the receive host may not know the value of RTT, in DCMRG, a fake RTT= RTT_0 is used, and the receive window size $W^0_{rcv} = xRTT_0/MSS$ is conveyed to the send host, which then convert W^0_{rcv} to the actual receive window size W_{rcv} as $W_{rcv} = W^0_{rcv} RTT/RTT_0$ by using the measured RTT at the send host. Now the question is how the receive host calculates x for an incoming flow.

The DCMRG incast controller requires that a send host who has an inelastic flow convey the minimum rate, θ, and the ID of the coflow the flow belongs to, to the receive host. This can be done by piggybacking the two values using the Option field in TCP header in the SYN packet.

The DCMRG incast controller at the receive host tries to evenly allocate the port bandwidth to all the flows coming into the same port, provided that the required minimum rates for inelastic flows are satisfied. Clearly, this design objective maximizes the sum of TCP-based user utilities, subject to network resource and minimum rate constraints. In what follows, we discuss the details how to achieve this design objective.

We first define the needed notations. At any given instant in time, assume that there are N incoming flows sharing a port of the receive host with port bandwidth C. Out of these

N flows, there are N_{in} inelastic flows, i.e., the inelastic flows with minimum rates, θ_i, for i=1,...,N_{in}, and the rest, $N_{el} = N - N_{in}$, flows are elastic flows. Further, let $R_{rv} = \sum_{i=1}^{N_{in}} \theta_i$ represent the reserved rate at the port of the receive host. We must have, $R_{rv} \leq C$. Further, define $R_{ab} = C - R_{rv}$ as the available bandwidth to accommodate more inelastic flows.

With the above preparation, now we describe how the receive host allocates the port bandwidth among all the flows sharing the port. The available rate, R_{ab}, is partitioned into two parts, R_{el} and $R_{ab} - R_{el}$. R_{el} is evenly allocated to all elastic flows, i.e., each elastic flow receives, $r_{el} = R_{el}/N_{el}$. Here R_{el} is determined by the following condition,

$$R_{ab} - R_{el} = \sum_{i=1}^{N_{in}} (r_{el} - \theta_i) I(r_{el} - \theta_i) \tag{15}$$

where $I(x)$ is an indicator function and it takes value 1 if $x \geq 0$, and 0, otherwise. Allocating R_{el} above to the elastic flows ensures that $R_{ab} - R_{el}$ amount of bandwidth is just enough to allow $(r_{el} - \theta_i)I(r_{el} - \theta_i)$ additional amount of bandwidth to be assigned to inelastic flow i. This ensures that the flow rate for an inelastic flow is r_{el} if $\theta_i < r_{el}$ and θ_i if $\theta_i > r_{el}$, and hence the sum of the TCP-based user utilities is maximized.

Clearly, the bandwidth allocation must be adjusted upon a flow arrival or departure. Upon the arrival or departure of an elastic flow, R_{el} must be updated through Eq. (15) and the receive windows for all the affected flows must be adjusted. Upon the departure of an inelastic flow, both R_{rv} and R_{el} must be updated. Now upon the arrival of an inelastic flow, if the minimum rate of the flow, $\theta \leq R_{ab}$, it will be admitted as an inelastic flow and again, both R_{rv} and R_{el} must be updated. Then the receive window for the flow will be calculated. If, on the other hand, $\theta > R_{ab}$, then the flow will be treated as an elastic flow. Moreover, if this flow is associated with an fan-in coflow and some other flows in the coflow are already using the port with minimum rates reserved, the minimum rate reservations for all those flows will be released and those flows will also be treated as elastic flows. The rationale for doing so is based on the understanding that a coflow deadline is determined by the slowest flow in the coflow. Since the flow that just arrives cannot meet its deadline due to bandwidth shortage, there is no point to attempt to meet the deadlines of other flows in the coflow. Releasing the reservations for those flows will only help the future inelastic flows or coflows to meet their deadlines. After releasing the reservations, both R_{rv} and R_{el} are updated and then the receive windows for all the affected flows are updated.

III. Evaluation of DCMRG

We evaluate DCMRG by ns-3 simulator using real datacenter workloads, i.e., Web-search [22] and Data-mining [26].

A. Testing of the DCMRG congestion controller

We first test the performance of the DCMRG congestion controller (i.e., without activating in-cast controller) against MRG [18] and D^{2}TCP [16], as they are the only existing lightweight (i.e., end-to-end and ECN-based, respectively), flow-deadline-aware congestion control protocols. The other existing deadline-aware congestion control protocols, e.g., [11]–[14], require significant network node involvement and/or new hardware support, and hence are not compared.

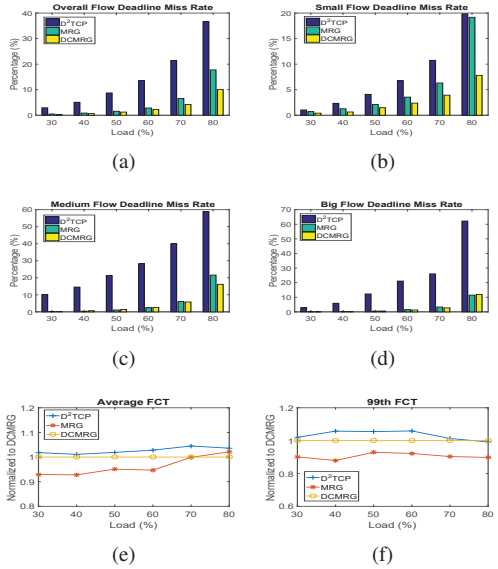

Fig. 1. Performance comparison of Congestion controller (Web-search workload).

Consider an 8x8 leaf-spine network topology with each rack having 32 hosts. The bandwidth/propagation delay is set at 10Gbps/10μs between a host and a leaf node and 20Gbps/20μs between a leaf node and a spine node. The queue size for the 10/20 Gbps links are set at 150/300 kbyes (i.e., 100/200 packets) and the ECN marking threshold is set at 65/130 packets, the typical value used in DCTCP and D^{2}TCP. The flow arrival process is assumed to be Poisson. We adjust the average flow arrival interval to generate different network loads. When a flow arrives, a send host is randomly selected, and another host in a different rack is randomly selected as the receive host. In this experiment, we assume that there is no coflow. The flows are classified into small (size<=100 Kbytes), medium (100<size<=1 Mbytes) and big (size>1 Mbytes) flows. The flow deadline miss rates of overall, small, medium and big inelastic flows as well as the average (including both elastic and inelastic) FCT and the 99th percentile of FCT are used as the performance metrics.

We first consider the web-search workload [22], and randomly select 20% flows as inelastic flows whose deadlines are set at: (i) 1ms+flow size/2Gbps for both small and medium flows; and (ii) flow size/2.5Gbps for big flows.

Figure 1 presents the results for the three protocols. We see that DCMRG performs much better in terms of flow deadline miss rates. DCMRG reduces the overall flow miss rate by as much as 3x and 1.6x, compared to D^{2}TCP and MRG, respectively, at high load. We also note that MRG achieves performance comparable with DCMRG for medium and big flows, but much poorer performance for small flows. This is

because for small flows, fast flow rate increase helps little on improving FCT as the flow is over in a few RTTs. Instead, FCT for a small flow is mainly determined by RTT, i.e., the shorter the RTT, the smaller the FCT. Just like TCP, MRG tends to cause longer switch queue, and hence, longer RTT and FCT. In contrast, like DCTCP [22], DCMRG incurs much smaller queuing delays, and hence, is able to maintain small flow deadline miss rate even for small flows. Overall, DCMRG outperforms D^{2}TCP by big margins across the entire load region tested. These results clearly demonstrate that DCMRG can effectively support inelastic flows.

Among the three protocols, MRG gives the smallest average FCT at light and medium network loads, but larger average FCT than DCMRG at the heavy load. At the light and medium loads, where the congestion occurs infrequently and the queue length is short, MRG experiences a smaller number of slow-down times due to the full use of the port buffer and hence lower average FCT. But at heavy load, MRG experiences more packet losses due to congestion and more retransmissions, and hence higher average FCT than DCMRG. The 99th FCT is determined by the big elastic flows. MRG benefits big flows and hence gives the best 99th FCT. Since DCMRG can provide better flow deadline guarantee, the tail FCT becomes less important because flows with tail constraints can be enforced as flows with deadlines. Again, DCMRG almost outperforms D^{2}TCP across the entire load region tested.

Now we test DCMRG with the Data-mining workload [26]. We randomly select 30% flows as inelastic flows whose deadlines are set at 0.6ms+flow size/4Gbps for small flows, and the same deadlines for medium and big flows as those in the Web-search workload case. We set a tighter deadline for small flows in this case because most small flows in the Data-mining workload have very small sizes (a few packets).

Figure 2 indicates that the simulation results. Similarly to the results in the Web-search workload case, DCMRG performs much better for big flows than small flows. DCMRG performs 2x and 3x better than D^{2}TCP in terms of the overall and big flow deadline miss rate at high load. DCMRG reduces the overall and small flow deadline miss rates by 2x as much as these in MRG. Similarly, DCMRG outperforms D^{2}TCP in terms of the average FCT for all load levels tested. DCMRG offers smaller 99th FCT in medium and high loads, but longer 99th FCT at high loads (70% and above) than D^{2}TCP. DCMRG achieves big flow deadline miss rate comparable with and a little bit larger deadline miss rate for medium flow than MRG. However, MRG performs poorly in terms of small flow deadline miss rate, even worse than D^{2}TCP.

The above case study clearly demonstrates that DCMRG congestion controller achieves overall best performance among all the three protocols, especially in providing high performance for inelastic flows.

B. Testing of both DCMRG congestion and incast controllers

As the majority of datacenter applications are scale-out by design, we evaluate the performance of DCMRG with the both congestion controller and incast controller by simulating

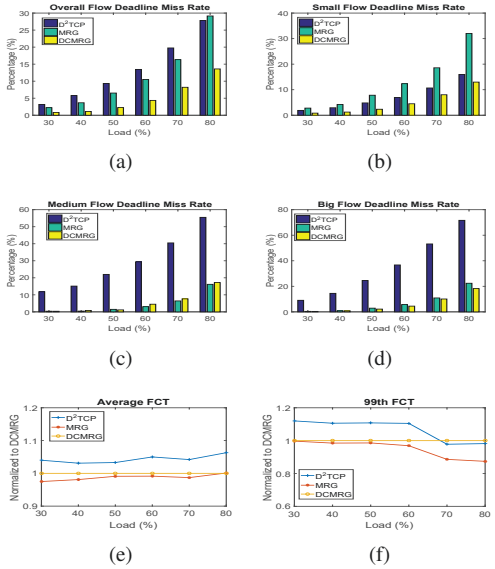

Fig. 2. Performance Comparison of Congestion Controller (Data-ming workload).

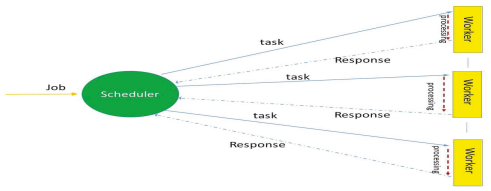

Fig. 3. A Fork-Join process.

a Fork-Join process [1], as depicted in Figure 3. A job arriving at a job scheduler running in a server spawns F tasks, which are dispatched to F workers to be processed. The data as a result of task processing are returned as data flows from the workers to the scheduler to be merged. The F fan-in data flows forming an incast coflow at the link between the top of rack switch and the server in which the job scheduler resides, called incast link hereafter. The slowest flow in the coflow determines the job response time. A job SLO can be translated into a coflow deadline. For simplicity, we consider the case where each job must meet a given deadline.

We use the same network topology as the one in the previous section. Assume that there are four job schedulers running in four different servers in each rack, resulting in a total of 32 schedulers. Jobs arrive following a Poisson process. When a job with fanout/fan-in degree F and deadline T_D arrives, it is randomly assigned to a scheduler. Upon receiving the job, the selected scheduler randomly selects F workers (i.e., servers which do not run schedulers) to dispatch the tasks to. Upon arrival at a worker, a task is queued and then processed before the resulting data are returned to the same scheduler the task is dispatched. Assume that the task time, i.e., the queuing time plus task processing time, follows a truncated exponential

[1]We test DCMRG in many-to-one incast congestion control, but it can also be applied to many-to-many incast congestion control without modification.

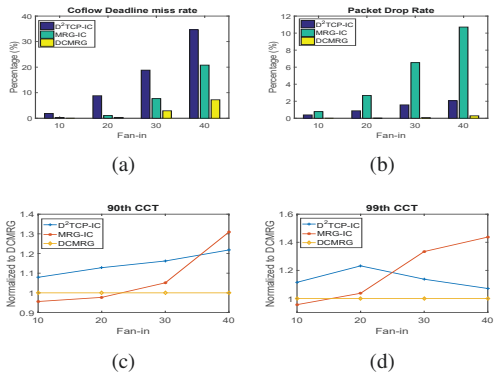

Fig. 4. Coflows with same fanout/fan-in and deadline.

distribution with average time T_P and cutoff time T_{cut}, where $T_{cut} < T_D$. After a response flow is generated at time T_r ($T_r <= T_{cut}$), it must be completed within $T_D - T_r$ amount of time to meet its deadline. As the user face applications [16], [27] usually have small flow sizes and deadline requirements, we further assume that each task to be dispatched has a small fixed size of 2K bytes and the size of each response flow is linearly distributed from 1K to 400K bytes.

Some background flows among servers without schedulers are also generated based on a mixture of the web-search [22] and Data-ming [26] workloads. Again, the background flows arrive following a Poisson process. When a background flow arrives, a server is randomly picked as the send host and another server in a different rack is randomly selected as the receive host. The background traffic load is fixed at 60% of the spine switch load.

In this study, we consider four performance metrics (i) the coflow deadline miss rate; (ii) the packet drop rate at the incast port of the server a scheduler resides in; (iii) the 90th percentile of coflow completion time (CCT); and (iv) the 99th percentile of CCT. To be fairly compare the performance of incast solution, we compare DCMRG against D^{2}TCP and MRG with incast solution ICTCP [6] (called as D^{2}TCP-IC and MRG-IC, respectively). We compare DCMRG with D^{2}TCP-IC and MRG-IC for the following two cases,

- Case I: all the jobs have the same fanout/fan-in degree and deadline. $T_P = 5ms$, $T_D = 15ms$ and $T_{cut} = 5ms$. Consider the cases with fanout/fan-in degrees ranging from 10 to 40.
- Case II: the fanout/fan-in degree is uniformly distributed from 1 to 50. T_D and T_P pair is randomly selected from a set of values: {(20,10), (30, 20), (40, 25), (50,30)} and $T_{cut} = 5ms$. The incast port load varies from 30% to 80%.

Case I Coflows with same fan-in degree and deadline: The results are presented in Figure 4. We see that the coflow deadline miss rate and packet drop rate at the incast link increase as the fan-in degree increases for all the three protocols. This is because more traffic is generated as fan-in degree increases, resulting in heavier incast congestion.

Among the three protocols, D^{2}TCP-IC has the highest coflow miss rate across the entire fan-in degree range studied. This is because D^{2}TCP-IC only provides relative rate differentiations among flows with different deadlines, rather than explicit rate guarantee. DCMRG achieves the lowest coflow miss rate due to its ability to explicitly handle incast congestion for coflows. The coflow deadline misses in DCMRG is mainly due to bandwidth shortage, which causes some coflows to be downgraded to elastic flows. To provide lower coflow deadline miss rate, a call admission control mechanism may be introduced to limit the inelastic flows and coflows so that R_{rv} is kept below C. But due to unpredictability of the flow start time for each inelastic flow in a coflow, how to do call admission control effectively is a challenging task, which is a subject to be explored as our future work. MRG-IC incurs the highest packet drop rate at the incast link. This is because it is an end-to-end solution and it does not slow down the sending rate before packet drop occurs. The packet drop rate in DCMRG is lowered than D^{2}TCP-IC by more than 70% due to its incast congestion control. Moreover, with the help with both ECN and incast congestion control, DCMRG reduces packet drop rate by more than 90% compared to MRG-IC.

MRG-IC has the shortest 90th percentile of CCT at small fan-in degrees. In this case, the congestion rarely occurs and MRG-IC can fully use the switch queue buffer to improve the link utilization. As the fan-in degree increases, however, the congestion occurs more frequently and CCT for MRG-IC grows faster than that for the other three protocols. D^{2}TCP-IC incurs longer 90th percentile of CCT than DCMRG. This is because the faster the increase rate for inelastic flows, the shorter their FCTs are. For the 99th percentile of CCT, MRG has the similar behavior. But D^{2}TCP-IC approaches DCMRG as the fan-in degree increases. Even at a large fan-in degree, e.g., 40, the coflow deadline miss rate for DCMRG is still within 10%. The 99th percentile of CCT in DCMRG comes from coflows being treated as elastic flows which share very limited bandwidth and hence results in its CCT comparable with that of D^{2}TCP-IC.

Case II Coflows with mixed fan-in degrees and mixed deadlines: Now we consider the case for coflows with mixed fanout/fan-in degrees and mixed deadlines. The results are depicted in Figure 5. Among the three protocols, DCMRG reduces coflow deadline miss rate by more than 40% and 60% compared to D^{2}TCP-IC and MRG-IC, respectively. We also note that the coflow deadline miss rate and packet drop rate are a little bit higher than those in the previous case. Due to different cowflow deadlines, the incast flows belonging to different coflows have higher chance to arrive at the incast link close to one another. For example, a scheduler handles a coflow with deadline T_D=50ms at time 10ms, and a coflow with deadline T_D =20ms at time 40ms. In this case, both coflows have the deadlines at 60ms and hence more flows coming from both coflows will have a good chance to compete with each other for the link bandwidth. This results in more shortage of reserved bandwidth for incast flows and

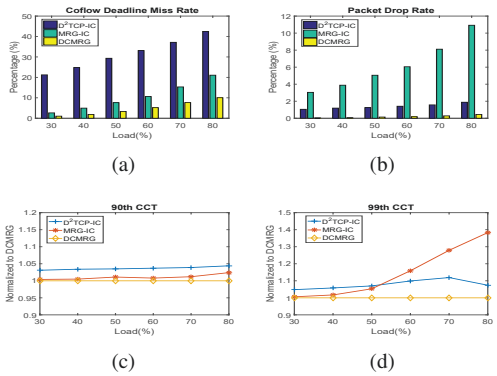

Fig. 5. Coflows with mixed fanout/fan-in degrees and mixed deadlines.

hence higher coflow deadline miss rate.

In summary, we conclude that DCMRG is very effective in reducing incast congestion caused by coflows. The objective of DCMRG is to minimize the coflow deadline miss rate. This may result in longer 99th percentile of CCT by treating some inelastic flows as elastic flows due to the lack of available bandwidth. If the objective is to minimize the 99th CCT, the flows in a coflow should share the bandwidth proportional to their remaining flow size so that all the flows in the coflow can be completed at the similar time to minimize CCT.

IV. Conclusions

In this paper, we proposed an incast-coflow-aware, ECN-based, soft minimum-rate-guaranteed congestion control protocol for datacenter networks, called DCMRG. DCMRG is the first traffic control protocol integrating congestion control and coflow-aware incast congestion control by providing flow deadline guarantee. DCMRG is composed of two major components, i.e., a congestion controller running on the send hosts and an incast congestion controller running on the receive hosts. DCMRG improves over MRG, an end-to-end soft minimum-rate-guarantee transport protocol, by integrating an incast congestion control and leveraging the ECN-mechanism. DCMRG deals with the incast congestion through dynamically regulating the receive window size to bound the overall traffic rate in the incast link while maximizing the overall network utilities. The large scale simulation results demonstrated that DCMRG outperform D^{2}TCP and MRG by big margins. DCMRG is readily to be deployed in existing datacenters with/without ECN support by only end host software upgrading.

Acknowledgment

This work was supported by Alibaba Group through Alibaba AIR Program and the US NSF under Grant No. CCF XPS-1629625, CCF SHF-1704504 and CCF SHF-2008835.

References

[1] D. Serrano, S. Bouchenak, Y. Kouki, T. Ledoux and J. Lejeune, J. Sopena, L. Arantes and O. Sens, "Towards QoS-Oriented SLA Guarantees for Online Cloud services," IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, 2013

[2] J. Brutlag, "Speed matters for google web search," Google, 2009.

[3] M. Jeon, S.Kim, S. Hwang,Y. He, S. Elnikety, A. Cox and S. Rixner, "Predictive Parallelization: Taming Tail Latencies in Web Search," Proceedings of ACM SIGIR, 2014.

[4] M. Choedhury and I. Stoica, "Coflow: A networking abstraction for cluster applications," Proceedings of ACM HotNets, 2012.

[5] A. Phanishayee,E. Krevat, V. Vasudevan, D. Andersen, G. Ganger, and G. Gibson, "Measurement and Analysis of TCP Throughput Collapse in Cluster-based Storage Systems," Proceedings of USENIX FAST, 2008.

[6] H. Wu, Z. Feng C. Guo and Y. Zhang, "ICTCP: incast congestion control for TCP in data-center networks," IEEE/ACM Transactions on Networking, vol. 21, pp. 345-358, 2011.

[7] H. Tan, S. Jiang, Y. Li, X. Li, C. Zhang, Z. Han and F. Lau, "Joint Online Coflow Routing and Scheduling in Data Center Networks," IEEE/ACM Transactions on Networking, vol. 27, pp. 1171-1786, 2019.

[8] M. Chowdhury and I. Stoica, "Efficient coflow scheduling without prior knowledge," Proceedings of ACM SIGCOMM, 2015.

[9] A. Ahmadi, S. Khuller, M. Purohit and S. Yang, "On scheduling coflows," Proceedings of MAPSP, 2017.

[10] S. Agarwal, S. Rajakrishnan, A. Narayan, R. Agarwal, D. Shmoys and A. Vahdat, "Sincronia: Near-Optimal Network Design for Coflows," Proceedings of ACM SIGCOMM, 2018.

[11] L. Chen, K. Chen, W. Bai and M. Alizadeh, "Scheduling Mix-flows in Commodity Datacenters with Karuna," Proceedings of ACM SIGCOMM, 2016.

[12] M. Alizadeh, S. Yang, M. Sharif and S. Katti, "pFabric: Minimal Near-Optimal Datacenter Transport," Proceedings of ACM SIGCOMM, 2013.

[13] C. Hong, M. Caesar and P. Godfrey, "Finishing flows quickly with preemptive scheduling," Proceedings of ACM SIGCOMM, 2012.

[14] C. Wilson, H. Ballani, T. Karagiannis and A. Rowstron, "Better Never than Late: Meeting Deadlines in Datacenter Networks," Proceedings of ACM SIGCOMM, 2011.

[15] G. Kumar, N. Dukkipati, K. Jang, H. Wassel, X. Wu, Xian, B. Montazeri, Y. Wang, K. Springborn, C. Alfeld, M. Ryan and D. Wetherall, "Swift: Delay is Simple and Effective for Congestion Control in the Datacenter, " Proceedings of ACM SIGCOMM, 2020.

[16] B. Vamana, J. Hasan and T. Vijakumar, "Deadline-Aware Datacenter TCP (D^{2}TCP)," Proceedings f ACM SIGCOMM, 2012.

[17] D. Shan and F. Ren, "Improving ECN marking scheme with micro-burst traffic in data center networks," Proceedings of IEEE INFOCOM, 2017

[18] L. Ye, Z. Wang, H. Che, Hao and C. Lagoa, "TERSE:A Unified End-to-End Traffic Control Mechanism to Enable Elastic, Delay Adaptive, and Rate Adaptive Services," IEEE Journal on Selected Areas in Communications, vol. 29, pp938-950, 2011.

[19] H. Almasi, H. Rezaei, M. Chaudhry and B. Vamanan, "Pulser: Fast Congestion Response Using Explicit Incast Notifications for Datacenter Networks," Proceedings of IEEE LANMAN, 2019.

[20] , Y. Zhang and N. Ansai, "On mitigating TCP incast in data center networks," Proceedings of IEEE INFOCOM, 2011.

[21] W. Chen, F. Ren, J. Xie, C. Lin K. Yin and B. Fred, "Comprehensive understanding of TCP Incast problem," Proceedings of IEEE INFOCOM, 2015.

[22] M. Alizadeh, A. Greenberg, D. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta and M. Sridharan, "Data center TCP (DCTCP)," Proceedings of ACM SIGCOMM, 2010.

[23] B. Movsichoff, C. Lagoa and H. Che, "End-to-End Optimal Algorithm for Integrated QoS, Traffic Engineering, and Failure Recovery," IEEE Transactions on Networking, vol. 15, pp813-823, 2007.

[24] Z. Wang, A. Singhal, Y. Wu, C. Zhang, H. Che, Hao, H. Jinag, B. Liu and C. Lagao, "HOLNET: A Holistic Traffic Control Framework for Datacenter Networks, " Proceedings of IEEE ICNP, 2020.

[25] S. Shenker, "Fundamental Design Issues for the Future Internet," IEEE Journal of Selected Areas in Communications, vol. 13, pp. 1176-1188, 1995.

[26] A. Greenberg, J. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel and S. Sudipta, "VL2: A Scalable and Flexible Data Center Network, " Proceedings of ACM SIGCOMM, 2009.

[27] T. Benson, A. Akella and D. Maltz, "Network traffic characteristics of data centers in the wild," Proceedings of ACM SIGCOMM Conference on Internet Measurement, 2010.