Journal of Network and Computer Applications 53 (2015) 103–114

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

DC-Vegas: A delay-based TCP congestion control algorithm for datacenter applications

Jingyuan Wang [a,*], Jiangtao Wen [b], Chao Li [a,c], Zhang Xiong [a], Yuxing Han [d]

[a] School of Computer Science and Engineering, Beihang University, Beijing 100191 China
[b] Department of Computer Science, Tsinghua University, Beijing 100084 China
[c] Research Institute of Beihang University in Shenzhen, Shenzhen 518057 China
[d] TCPEngines Inc., Santa Clara, CA 95054, USA

ARTICLE INFO

Article history:
Received 31 August 2014
Received in revised form
31 December 2014
Accepted 24 March 2015
Available online 13 April 2015

Keywords:
TCP
Congestion control
Datacenter
Deployment cost

ABSTRACT

TCP congestion control in datacenter networks is very different from in traditional network environments. Datacenter applications require TCP to provide soft real-time latency and have the ability to avoid incast throughput collapses. To meet the special requirements of datacenter congestion control, numerous solutions have been proposed, such as DCTCP, D^2TCP, and D^3. However, several deployment drawbacks, including significant modifications to switch hardware, the Operating System protocol stack, and/or upper-layer applications, as well as switch ECN requirements, which are not always available in already existing datacenters, limit deployment of these solutions. To address these deployment problems, in this paper, we proposed a delay-based TCP algorithm for datacenter congestion control, namely DC-Vegas. DC-Vegas combines the performance advantages of DCTCP with the deployment advantages of delay-based TCP Vegas. DC-Vegas can meet both soft real-time and incast avoidance requirements of datacenters, requiring minimal deployment modification to existing datacenter hardware/software (with only sender-side update and without ECN requirements). Experimental results obtained using the real datacenter test bed and an ns-2 simulator demonstrate that DC-Vegas has similar performance with the state-of-the-art Data Center TCP algorithm.

1. Introduction

Traditional TCP congestion control algorithms designed for the Internet environments have encountered many problems in datacenter networks. One problem is introduced by Online Data Intensive (OLDI) applications (Meisner et al., 2011), such as web search and social network services, which operate under "soft" real-time constraints so that any need for transmission latency in datacenters is as minimal as possible (Alizadeh et al., 2012a; Lee et al., 2012; Zhang et al., 2014). The other problem is TCP throughput collapse caused by a special network scenario in datacenters, namely incast. TCP flows in incast scenarios may suffer serious throughput losses (Chen et al., 2009). Incast scenarios widely exist in datacenter applications, such as MapReduce (Wang et al., 2014b; He et al., 2014). To address the soft real-time and incast problems, many improved solutions have been proposed, including fine-grained retransmissions (Vasudevan et al., 2009), ECN* (Wu et al., 2012), and ICTCP (Wu et al., 2010) for mitigating the TCP incast problem, HULL (Alizadeh et al., 2012a), D^3 (Wilson et al., 2011), Detail (Zats et al., 2012), and PDQ (Hong et al., 2012) for reducing low transmission latency, as well as DCTCP (Alizadeh et al., 2010), D^2TCP (Vamanan et al., 2012), L^2DCT (Munir et al., 2015), RDT (Jingyuan et al., 2014) and WDCTCP (Wang et al., 2013a) for both latency reduction and incast avoidance. Although these solutions achieved excellent performance in laboratorial datacenter networks, there still exist some deployment barriers when adopting them in "*existing*" datacenters that were already built all over the world. For solutions such as D^3 and PDQ that are based on custom network protocols and not compatible with TCP, all TCP-based application layer software must be replaced, which is impractical in many commercial systems. Algorithms such as DCTCP that rely on ECN (Explicit Congestion Notification) marking for congestion detection, on the other hand, will require ECN support, a capability which is not universally available even today (Stewart et al., 2011; Bauer et al., 2011). In addition, many solutions require significant modifications to network hardware and/or the OS protocol stack; some even require custom ASICs. Therefore, these existing solutions are very promising for new datacenters, but for the datacenters that already exist, they are not suitable.

In this paper, we propose a high-performance and low-deployment-cost datacenter TCP algorithm named DC-Vegas. DC-Vegas was inspired by an analysis of queue length distribution of delay-based TCP in datacenter networks. The analysis revealed why the delay-based TCP

* Corresponding author.
E-mail address: jywang@buaa.edu.cn (J. Wang).

http://dx.doi.org/10.1016/j.jnca.2015.03.010
1084-8045/

Vegas (Brakmo et al., 1994) algorithm performs well for reducing transmission latency but poorly in incast scenarios. Based on this insight, we design the DC-Vegas algorithm, which combines the low-cost deployment advantage of TCP Vegas and the excellent performance of DCTCP in datacenters. We deployed DC-Vegas in a production datacenter test bed. The deployment only needed to update TCP senders and did not require ECN support, application software and/or network modifications. Experiments using the datacenter test bed show that DC-Vegas (1) reduces queuing delays by two orders of magnitude and reduces the deadlines missed rate of OLDI applications by 80%; (2) significantly avoids incast collapse; and (3) maintains graceful fairness and convergence. The ns-2 simulation experiments show that the performance of DC-Vegas is similar to that of DCTCP.

The rest of this paper is organized as follows. Section 2 analyzes why delay-based TCP Vegas performs well for reducing latency but poorly in incast scenarios. Section 3 describes the algorithm details of DC-Vegas and presents some analysis. Experimental results are presented in Sections 4 and 5. The conclusions of this paper are given in Section 7.

2. Delay-based TCP in datacenters

2.1. Performance of TCP Vegas in datacenters

Congestion detection mechanism is an important module of TCP algorithms (Alrshah et al., 2014; Xu et al., 2011). The most widely used congestion detection mechanism is loss-based detection, which is adopted by many popular TCP algorithms such as TCP Reno (Jacobson, 1988) and CUBIC (Ha et al., 2008; Wang et al., 2013b). Because packet losses occur only after packet queues have built up, loss-based TCP in datacenters always introduces very long queuing delay, and therefore causes degradation of user experience.

The most popular congestion detection method in datacenters is ECN (Explicit Congestion Notification), such as DCTCP, ECN*, and D^{2}TCP. However, ECN is not universally supported by datacenter hardware, especially in datacenters currently in operation. According to the report of Bauer et al. (2011), only 60% end-to-end loops in the Internet are ECN enabled, and as reported in Stewart et al. (2011), finding a datacenter switch that supports ECN to test performance of DCTCP is very difficult. Furthermore, ECN is an IP level protocol, which requires all agents in an end-to-end loop to be ECN enabled, so deploying ECN-based TCP in a non-ECN equipped datacenter means a system-wide software and/or hardware updating. Even though deploying DCTCP over an ECN enabled datacenter, modifications on both sender side and receiver side, as well as parameter setup of switches, are still unavoidable. So the time cost and expense of such deployment is unbearable for production datacenters. As reported by Emerson Network Power (State of the data center, 2011), more than 500 thousand datacenters were built before 2011 around the world. Most of these datacenters are waiting for TCP updating because most of ECN-based datacenter TCP solutions were proposed during 2011–2013.

An alternative for ECN is delay-based TCP congestion control (Brakmo et al., 1994; Wei et al., 2006), which uses end-to-end queuing delay as an indication of network congestion. Because delay-based TCPs aim at keeping packet queue length in network buffers under a reasonable level, queuing delay of delay-based TCP is usually shorter than loss-based algorithms (Wang et al., 2014a). Moreover, delay-based TCPs can proactively reduce their throughput before packet losses appear; therefore, they have potential to mitigate the TCP incast problem (Lee et al., 2012; Wu et al., 2010). However, existing delay-based TCP algorithms cannot meet *both* the low latency and incast avoidance requirements of datacenters. In this section, we use TCP Vegas as a representative of delay-based TCP algorithms to analyze its performance in datacenter networks.

We conducted two sets of experiments on a production datacenter of the Computer Network Information Center (CNIC)[1] in the Chinese Academy of Sciences. First, we investigated queuing delay of TCP Vegas in the datacenter. We let ten long-term TCP flows pass between two hosts that were connected by 1 Gbps links in the datacenter. According to Alizadeh et al. (2010), the typical number of concurrent long-term TCP connections for a host in datacenters is less than four; therefore, ten flows represented a fairly crowded setting. Figure 1(a) plots the RTT (Round-Trip Time) variations between the hosts when TCP Vegas and TCP Reno algorithms were used. As shown in the figure, the queuing delay of TCP Vegas was far smaller (by approximately two orders of magnitude) than that of the loss-based TCP Reno algorithm. Second, we investigated the performance of TCP Vegas in an incast scenario. We used 47 hosts connected by a 1 Gbps switch, where one host was designated the receiver, and the others were used as senders. A subset of the senders simultaneously and repeatedly sent 128 KB data blocks to the receiver. Figure 1(b) shows aggregated throughput of the senders. Regardless of whether TCP Vegas or Reno was used, the throughput of the senders collapsed when the number of senders exceeded a threshold; this is a typical TCP incast collapse phenomenon. The threshold for TCP Vegas was the small value of 20, slightly bigger than TCP Reno and not satisfactory in datacenter applications. The two experiments on TCP Vegas in datacenter networks demonstrated that the TCP Vegas algorithm is promising for latency-sensitive datacenter applications, but performs poorly in incast scenarios.

2.2. Analysis of TCP Vegas in datacenters

TCP Vegas uses end-to-end queue length samples to detect congestion. In each RTT, TCP Vegas estimates current queue length as

$$q = \frac{w}{RTT} \cdot (RTT - RTT_{min}), \tag{1}$$

where w is the size of the congestion control window, RTT is the sampled RTT, and RTT_{min} is the minimum observed RTT, which is used as an estimate of network propagation delay. TCP Vegas adjusts the window size in each RTT by comparing q with a threshold K_v. If $q > K_v$, TCP Vegas considers the network congested and reduces the window but instead increases the window if $q < K_v$. In other words, TCP Vegas uses a binary decision to detect network congestion.

The binary congestion detection of TCP Vegas works well for traffic on the Internet but not for datacenter networks because network queue behaviors on the Internet are very different from those in datacenter networks. Figure 2 shows network buffer queue fluctuation of datacenter networks and the Internet. Datacenter queue fluctuation is observed between two servers of the CNIC datacenter using 10 TCP Vegas flows, and the Internet queue fluctuation is observed from an Amazona EC2 server to a computer in our laboratory. For the sake of comparison, we normalize the queue length samples using $Q_s/((\sum_{i=1}^{i=S} Q_i)/S)$, i.e., the ratio between the current queue sample Q_s and the average of all such values of S samples. The value of S is 200,000 in our experiments, among which 1000 samples are plotted in Fig. 2. As shown in Fig. 2(a), there are many impulses in the Internet queue fluctuation. A probable cause of these impulses is network congestion; if so, TCP Vegas can easily distinguish them from non-congestion network states by using simple binary quantization. On the other hand, the queue length variation of the datacenter, which is shown in Fig. 2(b), is very

[1] http://english.cnic.cas.cn/

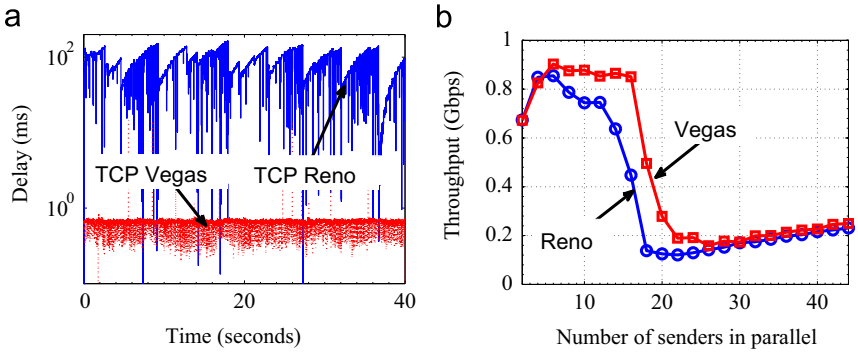

Fig. 1. TCP Vegas in datacenter networks. (a) Queuing delay and (b) the incast scenario.

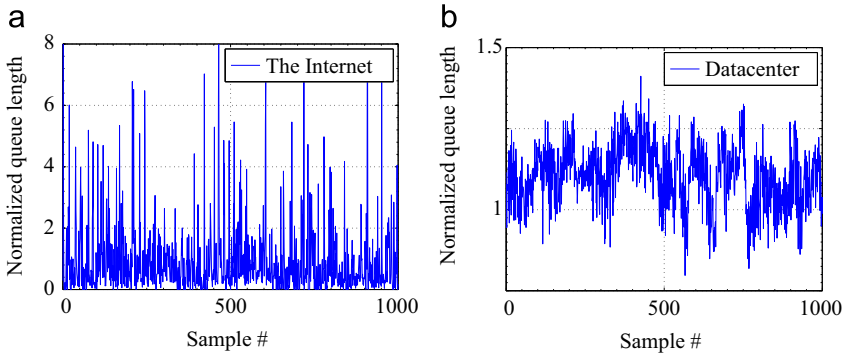

Fig. 2. Queuing length variation. (a) The Internet and (b) datacenter networks.

Table 1
Notations of the HMM.

Notations	Definition
S_c, S_n	The congestion state (hidden) and the non-congestion state (hidden)
Q_c, Q_n	The observable queue length of the congestion and non-congestion states
O_v, O_{dc}, O_{dcv}	Observations of the HMM for Vegas, DCTCP and DC-Vegas
$p_c^v, p_c^{dc}, p_c^{dcv}$	Probability of the network in state S_c for Vegas, DCTCP and DC-Vegas
$p_c^V(i)$	Probability of the network in state S_c for the Viterbi algorithm
K_v, K_{dc}	The congestion threshold of Vegas and DCTCP
$Q(i)$	The ith sample of the network queue length
$q^m(i)$	The ith sample of the queue length for TCP session m

uniform. It is very difficult to detect congestion in this type of queue for the binary detection of TCP Vegas.

To analyze the congestion detection of TCP Vegas in depth, we adopt a Hidden Markov Model (HMM) (Rabiner and Juang, 1986a) with Gaussian observations to model TCP congestion detection systems. The notations adopted by the HMM are listed in Table 1. As illustrated in Fig. 3, a network has two (hidden) states, the congestion state S_c and the non-congestion state S_n; the observable network buffer queue length of the congestion and non-congestion states are Q_c and Q_n, respectively. For the sake of modeling, we assume that $Q_c \sim \mathcal{N}_c(\mu_c, \sigma_c^2)$ and $Q_n \sim \mathcal{N}_n(\mu_n, \sigma_n^2)$ follow a Gaussian distribution and that the queue length observation of the hidden states follows a Gaussian mixture distribution $k_c\mathcal{N}_c + k_n\mathcal{N}_n$.

We use the Baum–Welch algorithm (Welch, 2003) to calculate parameters of the HMM, and use the queue fluctuation data in Fig. 2 as the training data. The PDF of learned $\mathcal{N}_c$, $\mathcal{N}_n$, and the queue fluctuation data are shown in Fig. 4. As shown in the figures, the distribution of the Internet queue has a long tail that is covered mostly by $\mathcal{N}_c$, indicating the queue fluctuation impulses of the Internet, which correspond to the PDF long tail, are indeed caused by network congestion. Except for the long tail, the queue length distribution lies under $\mathcal{N}_n$ and is highly concentrated. For the queue distribution of the Internet, threshold-based binary decisions can easily cut off the network congestion-caused long tail, so the congestion detection of TCP Vegas is reasonable for the Internet traffic. On the other hand, for the datacenter network, the distribution shown in Fig. 4(b) is much more uniform. $\mathcal{N}_c$ and $\mathcal{N}_n$ are significantly mixed, making it very difficult to distinguish congestion states from non-congestion states if using only a binary decision.

We further model the TCP Vegas algorithm using the Hidden Markov Model framework. For the HMM illustrated in Fig. 3, the congestion detection of a TCP algorithm can be modeled as "decoding" the sequence $P = \{p_c(1), \ldots, p_c(i)\}$ using the given observation sequence $O = \{Q_c(1), \ldots, Q_c(i)\}$, where $Q(i)$ is the ith sample of the network queue length, and $p_c(i)$ is the probability that the network is in S_c for $Q(i)$. Thus, for a network containing M TCP Vegas flows, the congestion detection model is

$$\begin{aligned} &\text{Input}: \quad I_v = \{q^m(1), \ldots, q^m(i), \ldots\}, \\ &\text{Ouput}: \quad p_c^v(i) = \begin{cases} 1, & q^m(i) > K_v, \\ 0, & q^m(i) \le K_v, \end{cases} \end{aligned} \tag{2}$$

where q^m is the queue length sample of the TCP Vegas flow m. We

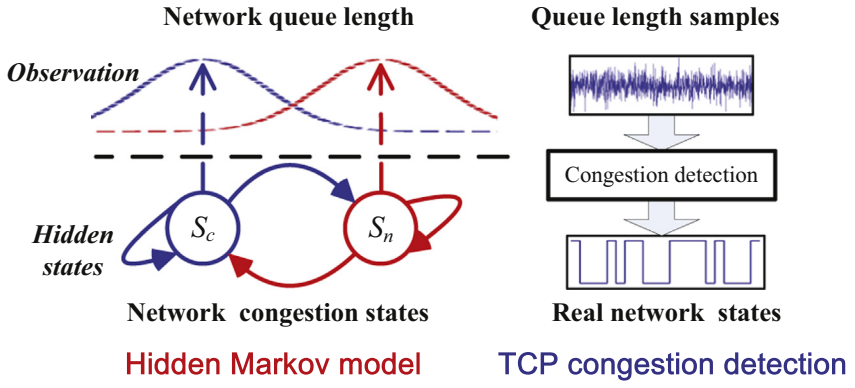

Fig. 3. HMM in TCP congestion detection.

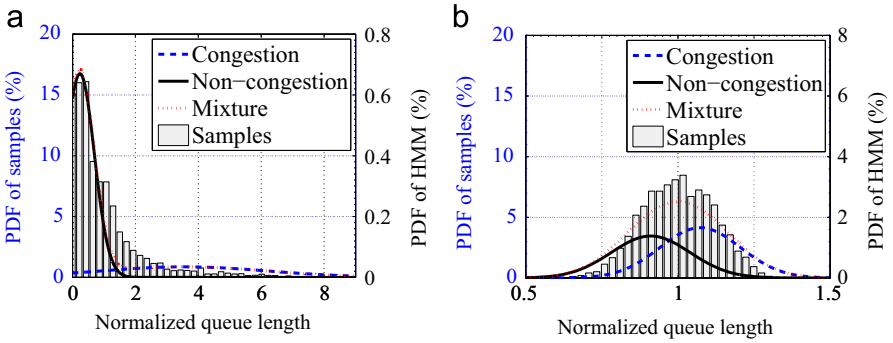

Fig. 4. PDF of queuing length and HMM. (a) The Internet and (b) datacenter networks.

assume $q^m(i) = Q(i)/M + \epsilon^m(i)$, where $\epsilon^m(i)$ is a random signal and $\sum_{m=1}^{M} \epsilon^m(i) = 0$.

We compare performance of model (2) with the Viterbi algorithm (Rabiner and Juang, 1986b) to evaluate the congestion detection of TCP Vegas. The Viterbi algorithm is a dynamic programming algorithm for finding the most likely sequence of hidden states of HMM. Input of the Vitertib algorithm is a sequence of observed states and the output is the most likely sequence of hidden states. We use the sequence of $Q(i)$ as input of the Vitertib algorithm, and get a sequence of probability that the hidden network state for $Q(i)$ belongs to congestion state, i.e., $p_c^V(i)$. Because the Viterbi algorithm is an optimized algorithm for HMM hidden state sequence decoding, we use the Viterbi decoding result $p_c^V(i)$ as an approximate "real" solution of network congestion probability. For the i-th input $q^m(i)$, if $p_c^v(i) - p_c^V(i) > 0.5$, i.e., the congestion probability calculated by TCP Vegas is much bigger than the "real" solution $p_c^V(i)$, which means a severe overstating of network congestion occurs. We treat the congestion report $p_c^v(i)$ as a false alarm, i.e., a false congestion detection. On the contrary, we treat the congestion report $p_c^v(i)$ as a false non-congestion if $p_c^V(i) - p_c^v(i) > 0.5$, i.e., a congestion state leaking.

The false congestion and non-congestion rates of TCP Vegas using the queue fluctuation data in Fig. 2 are shown in Fig. 5. As shown in the figure, the false non-congestion rate of TCP Vegas in the datacenter networks reaches 60%, which is significantly higher than in the Internet. False non-congestion detections means underestimation of network congestion. 60% network congestion underestimation of TCP Vegas in datacenter networks causes the congestion detection mechanism of TCP Vegas to be invalid in datacenter networks. For incast scenarios, this problem becomes more serious, because short TCP connections of incast servers do not have enough time to collect more queue samples and re-detect network states. That is why TCP Vegas performs poorly in incast scenarios.

2.3. Motivation for the DC-Vegas algorithm

The congestion detection mechanism of the DC-Vegas algorithm was inspired by DCTCP. The DCTCP algorithm has three main components (Alizadeh et al., 2010): (1) *ECN marking at the switch*: the switch compares its packets queue length in buffers with a threshold K_{dc} and marks the ECN bit in IP packet headers if the queue length is greater than K_{dc}. (2) *ECN-echo at the receiver*: the ECN flags in packets are then copied by the DCTCP receiver to corresponding ACKs and echoed to the sender when packets carrying the ECN flag are received. (3) *Congestion detection at the sender*: the sender calculates the fraction of ACKed packets with marked ECN flags in a window, i.e.,

$$F_{dc} := \frac{\#\ of\ packets\ with\ ECN\ mark}{Total\ \#\ of\ packets\ in\ a\ window},$$

and filtered by an exponential moving average (EMA) filter to find

$$\alpha_{dc} := (1-g) \times \alpha_{dc} + g \times F_{dc}, 0 < g < 1. \quad (3)$$

Then, in each RTT, DCTCP increases the value of the congestion control window by one when $F_{dc} = 0$; otherwise, the window w_{dc} is reduced to

$$w_{dc} := w_{dc} - w_{dc} \times \frac{\alpha_{dc}}{2}.$$

We also use the HMM defined in Section 2.2 to model DCTCP. Because DCTCP switches mark the ECN bit of in-flight packets when $Q(j) > K_{dc}$, so expectation of F_{dc} is equivalent to the probability that $Prob\{Q(j) > K_{dc}, \forall j\}$, where $j \in$ is in the indices set of all $Q(j)$ in the same RTT with $Q(i)$. Hence, the congestion detection of DCTCP at the sender side is

$$\begin{aligned} &\text{Input}: \quad I_{dc} = \{Q(1), \ldots, Q(i), \ldots\}, \\ &\text{Output}: \quad p_c^{dc}(i) = 1 - P\{Q(j) \le K_{dc}, \forall j\}. \end{aligned} \quad (4)$$

Similar to the TCP Vegas case, we calculated false non-congestion

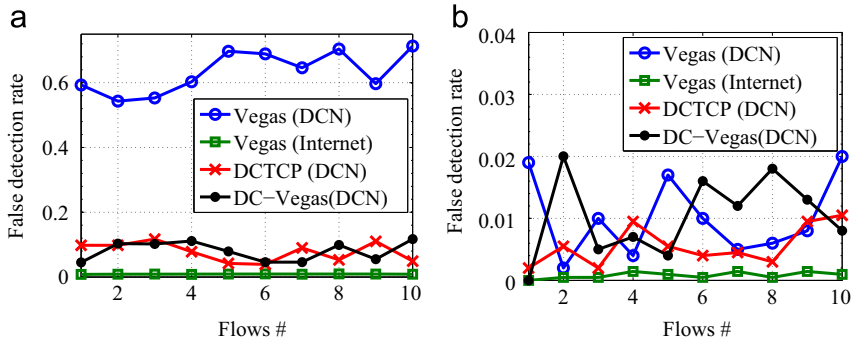

Fig. 5. False congestion/non-congestion detection rate.

and false congestion rates for DCTCP. As shown in Fig. 5, the false rate of DCTCP is significantly lower than that of TCP Vegas, especially the false non-congestion rate, which is critical for incast congestion avoidance.

As mentioned previously, a major deployment drawback of the DCTCP algorithm is the requirement for ECN support and both sender and receiver modifications. On the other hand, deployment of TCP Vegas only modifies TCP senders and does not need ECN. However, the poor incast performance of TCP Vegas limits its application in datacenter networks. It is desirable to combine the deployment advantage of TCP Vegas and the performance advantage of DCTCP in the same algorithm. To achieve this goal, we replace Q in (4) of DCTCP with q^m:

$$\text{Input}: \quad I_{dcv} = \{q^m(1), \ldots, q^m(i)\},$$
$$\text{Output}: \quad p_c^{dcv}(i) = 1 - P\left\{q^m(j) \le \frac{K_{dc}}{M}, \forall j\right\}, \tag{5}$$

where j is in the index set of all $q^m(j)$ in the same window with $q^m(i)$. The model is equivalent to the DCTCP detection model (4) but replaces the global queue observation Q^m with local queue observation q^m. The false congestion and false non-congestion rates of the model (5) for the datacenter queue samples are plotted in Fig. 5 as 'DC-Vegas (DCN)'. The results show that the detection accuracy of the model (5) is very close to DCTCP. This result motivates us to design the DC-Vegas algorithm using the model (5) as the congestion detection mechanism.

3. The DC-Vegas algorithm

3.1. Algorithm description

In DC-Vegas, when an ACK arrives, the sender first estimates current network queue length q using Eq. (1) and then compares q with a threshold K_{dcv}. When all packets in the same window are acknowledged, DC-Vegas calculates

$$F_{dcv} := \frac{\#\ of\ ACKs\ with\ q > K_{dcv}}{Total\ \#\ of\ ACKs\ in\ a\ window} \tag{6}$$

and applies an EMA filter to find $\alpha_{dcv} := (1-g) \times \alpha_{dcv} + g \times F_{dcv}$, where $0 < g < 1$. DC-Vegas uses α_{dcv} to measure network congestion level. Obviously, values of F_{dcv} closer to 1 indicate higher congestion. According to the model defined in (5), if we let $F_{dcv} \approx Prob\{q(j) > K_{dcv}, \forall j\}$, the congestion detection mechanism of DC-Vegas in (6) follows the model (5).

DC-Vegas updates its window w_{dcv} in each RTT according to the network congestion level indicated by F_{dcv}. The specific algorithm is

$$w_{dcv} := \begin{cases} w_{dcv} - w_{dcv} \times \dfrac{\alpha_{dcv}}{2} & \text{if } F_{dcv} > 0, \\ w_{dcv} + 1 & \text{if } F_{dcv} = 0. \end{cases}$$

Thus, when network congestion is at a low level, DC-Vegas reduces its window by a small value; otherwise, the window is reduced significantly to alleviate high network congestion. Similar to the traditional TCP Reno algorithm, DC-Vegas halves its window when a packet loss event is detected. Other parts of the standard TCP algorithm framework, including slow start, fast recovery, and fast retransmission, are left unchanged.

3.2. Analysis

We introduce an approximate model to analyze behaviors of DC-Vegas. The analysis of DC-Vegas in this section uses the modeling method of TCP Reno in Padhye et al. (1998). The modeling method only considers the Congestion Avoid part of TCP algorithms, and assumes that the window size is not limited by the receiver's advertised flow control window. The round trip time of the model is assumed to be independent of the window size, and the time needed to send all the packets in a window is assumed to be shorter than the round trip time. The model assumes N long-term DC-Vegas flows sharing a single bottleneck link with bandwidth B and RTT D. As shown in Fig. 6, variation of the window w of a TCP DC-Vegas flow has a "sawtooth" shape, so we define a period between two w size reductions as a Sawtooth Period (STP). Notations used in the model are summarized in Table 2.

DC-Vegas senders calculate F_{dcv} for a window of packets. The average F_{dcv} of STP_i is

$$F_i = \frac{W_i}{Y_i}. \tag{7}$$

The window of STP_i increases by one after each RTT and decreases by a factor $(1-\alpha_i/2)$ when queue length is greater than K_{dcv}. We therefore have

$$\begin{aligned} Y_i &= \sum_{k=0}^{L_i-1} \left[\left(1 - \frac{\alpha_{i-1}}{2}\right) W_{i-1} + k\right] \\ &= \left(1 - \frac{\alpha_{i-1}}{2}\right) W_{i-1} L_i + \frac{L_i(L_i-1)}{2}. \end{aligned} \tag{8}$$

In addition, the window size at the end of STP_i is

$$W_i = \left(1 - \frac{\alpha_{i-1}}{2}\right) W_{i-1} + L_i - 1. \tag{9}$$

From (7)–(9), expectations of the random variables Y, L, and F form the following equation set:

$$\begin{cases} E[F] = E[\alpha] = \dfrac{E[W]}{E[Y]} \\ E[Y] = E[L]\left(\dfrac{(2 - E[\alpha])E[W] + E[L] - 1}{2}\right) \\ E[L] = \dfrac{E[W]E[\alpha]}{2} + 1 \end{cases} \tag{10}$$

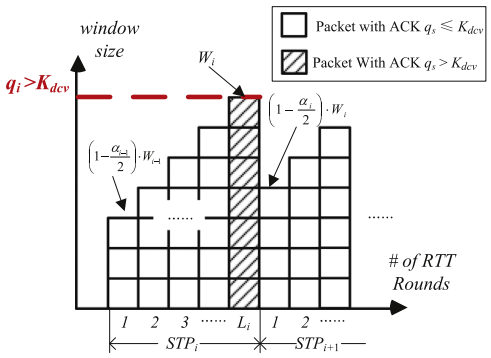

Fig. 6. Window size variation during a STP.

Table 2
Notations of STP.

Notation	Definition
STP_i	The ith sawtooth period (STP)
W_i	The w size at the end of STP_i
α_i	The α of STP_i
q_i	The max queue length of STP_i
Y_i	The number of packets sent in STP_i
L_i	The RTT round where $q_i > K$ in STP_i

From (10), we obtain

$$\frac{2}{E[W]} = E[\alpha]^2\left(1-\frac{E[\alpha]}{4}\right)+2\left(\frac{E[\alpha]}{E[W]}-\frac{E[\alpha]^2}{4E[W]}\right). \tag{11}$$

Assuming that α is small enough, then $E[\alpha]\approx\sqrt{2/E[W]}$. Assuming furthermore that the N flows are synchronized, which follows the assumptions in Alizadeh et al. (2010), then

$$E[W] = \frac{BD}{N}+K_{dcv}.$$

The expected window size oscillation period (OP) and amplitude (OA) are

$$E[OP] = E[OA] = \frac{E[W]E[\alpha]}{2} \approx \frac{1}{2}\sqrt{2E[W]} = \sqrt{\frac{BD+NK_{dcv}}{2N}}.$$

Because we assume that the N flows are synchronized and aggregated amplitude of synchronous TCP flows is equal to or larger than aggregated amplitude of asynchronous, the amplitude analysis here is the amplitude upper bounders.

Moreover, the expected queue length of a DC-Vegas flow is

$$E[Q] = \frac{E[Y]}{E[L]}-\frac{BD}{N}\approx K_{dcv}+1-\sqrt{\frac{BD+NK_{dcv}}{8N}}.$$

A comparison of DC-Vegas and DCTCP is given in Table 3. As shown in the table, the DC-Vegas algorithm is very similar to DCTCP in terms of window variation and queue length. The average queue length, oscillation period, and amplitude for the DC-Vegas and DCTCP algorithms will be of identical form if we let $K_{dc} = N\times K_{dcv}$, which is very reasonable because K_{dc} of DCTCP is observed by an ECN-enabled switch for a total of N flows, whereas K_{dcv} of DC-Vegas is estimated by each sender. In other words, DC-Vegas therefore can be regarded as a distributed DCTCP.

On the other hand, DC-Vegas also can be regarded as a higher-precision TCP Vegas algorithm. Through adopting a finer granularity and more accurate congestion detection, DC-Vegas becomes more suitable for datacenter networks than is traditional TCP Vegas. Moreover, DC-Vegas adopts an AIMD (Additive Increase and Multiplicative Decrease) window size adjustment, which has faster network congestion back-off speed than AIAD (Additive Increase Additive Decrease) of traditional Vegas. This feature is very important for small-buffer networks such as datacenter networks. In summary, DC-Vegas is an optimized hybrid of DCTCP and TCP Vegas.

3.3. Parameter settings

The maximum queue length q_{max} of DC-Vegas is $K_{dcv}+1$. The expected minimum value of the queue occupancy sawtooth pattern is given by

$$q_{min} = q_{max}-OA = K_{dcv}+1-\sqrt{\frac{BD+NK_{dcv}}{2N}}. \tag{12}$$

To ensure networks are fully utilized even when there is only one DC-Vegas flow in the network, we let $Q_{min}>0$ and $N=1$; therefore, the setting of K_{dcv} should be $K_{dcv} > (\sqrt{1+8BD}-3)/4$. For the parameter g, we use the same method as DCTCP to compute g in our implementation, that is, $g < 1.386/\sqrt{2(BD+K_{dc})}$.

3.4. Granularity requirement of RTT measurement

Accurate RTT measurement is an important issue for delay-based TCP algorithms. DC-Vegas measures network queue length through $q = T/Packet\_Size\times(RTT-RTT_{min})$. Therefore, to achieve "one packet" queue length measurement precision, RTT measurement precision must satisfy

$$G \leq \frac{Packet\_Size}{T}. \tag{13}$$

Table 4 lists the RTT measurement precision requirements of several typical networks. The highest precision requirement is 1.13 μs for 10 Gbps links, which is available for both Linux, for which the RTT measurement precision is 1 μs, and Windows OS, for which is 100 ns (Wu et al., 2010). In fact, RTT precision of (13) in DC-Vegas can be further relaxed because DC-Vegas only needs to know whether $q > K$. The actual queue precision requirement is "K packets" but not "one packet". As shown in Table 4, the highest RTT precision requirement for "K packets" is 23.8 μs for typical network settings, which can be easily achieved by mainstream OSs.

4. Evaluation on a production datacenter test bed

We used a subset of the machines in the CNIC datacenter as a test bed to evaluate DC-Vegas. In the test bed, 47 machines were connected by a Dell Force10 S50N datacenter switch that does not have ECN. The bandwidth between the switch and the machines was 1 Gbps. The OS of the machines was Red Hat Enterprise Linux

Table 3
Comparison of DC-Vegas and DCTCP.

Algorithms	$E[Q]$	OP/OA
DC-Vegas	$K_{dcv}+1-\sqrt{\frac{BD+NK_{dcv}}{8N}}$	$\sqrt{\frac{BD+NK_{dcv}}{2N}}$
DCTCP (Alizadeh et al., 2010)	$\frac{K_{dc}}{N}+1-\sqrt{\frac{BD+K_{dc}}{8N}}$	$\sqrt{\frac{BD+K_{dc}}{2N}}$

Table 4
The RTT measurement granularity requirements.

Precision	10 Gbps	1 Gbps	0.1 Gbps
1 packet (μs)	1.13	11.3	113
K packets (μs)	23.8	67.9	339

Server release 5.3 with kernel version 2.6.18. The switch and the machines used their default setup settings. The CPU, memory and storage space of the machines were not bottlenecks in the experiments. Unless otherwise noted, the experiments used the parameter settings listed in Section 3.3. We could not include algorithms such as DCTCP due to the lack of ECN in the test bed.

4.1. Basic properties

We first evaluated the throughput of DC-Vegas on long-term flows when the recommended value of parameter K_{dcv} was used. Two machines in the test bed, one as a sender and the other as a receiver, were used in the experiment. The bandwidth between the sender and receiver was 1 Gbps, and the average propagation delay of the link was approximately 50 μs. According to Section 3.3, $K_{dcv} = 6$ for this network condition. In the experiment, the sender was allowed to send data as quickly as it could. The results show that DC-Vegas achieved more than 90% bandwidth utilization when $K \geq 5$. Moreover, we further tested DC-Vegas under a 10 Gbps and 50 μs delay setup by connecting two machines using 10 Gbps NICs directly. The recommended parameter is $K_{dcv} = 21$ for this condition. The experimental results show that DC-Vegas achieved more than 90% bandwidth utilization when $K \geq 16$. The results indicate that long-term DC-Vegas flows with the recommended K_{dcv} can fully saturate datacenter network bandwidth. The recommended K_{dcv} is a reasonable setup.

Next, we tested the influence of DC-Vegas on network bottleneck links. In the experiment, one machine in the test bed was used as a receiver and the other machines were used as senders. We let several senders established TCP connections to the receiver and sent data as quickly as they could. The end-to-end RTT and the packet loss rate of the TCP flows in the bottleneck experiment were sampled for cases with 2 and 32 senders. The cumulative distribution function (CDF) of RTT and packet loss for different algorithms are plotted in Fig. 7. As shown in Fig. 7(a), the *RTT*s of 2 and 32 DC-Vegas flows were stable at approximately 80 μs and 500 μs, respectively, while the Vegas flows were stable at approximately 80 μs and 1000 μs. In contrast, RTTs incurred by the Reno flows were two orders of magnitude greater than were incurred by both DC-Vegas and TCP Vegas. As shown in Fig. 7(b), the packet loss rate incurred by DC-Vegas was significantly lower than TCP Reno and Vegas. As shown in Table 3, DC-Vegas has very low expected in-flight queue length, so it can keep bottleneck buffers at a very low occupancy rate, and then avoid long queuing delay and packet loss, which is why DC-Vegas has low RTT and packet loss in this experiment.

4.2. Performance in datacenter scenarios

In this subsection, the incast performance and queuing delay of DC-Vegas in datacenter networks was evaluated. Most of the experimental setups in this subsection follow previous datacenter TCP studies (Alizadeh et al., 2010; Wu et al., 2010; Vasudevan et al., 2009; Vamanan et al., 2012).

4.2.1. Incast

In the incast experiments, several servers in the test bed synchronously and periodically sent small data chunks to a receiver. The total amount of data sent by all of the servers in each round was fixed and uniformly divided among the servers as in Alizadeh et al. (2010); Wu et al. (2010) and Vasudevan et al. (2009). Figure 8 shows the results when the total data size was set to 1 MB and 4 MB; the number of servers varied from 2 to 46. For each data size and each number of servers, the experiment was repeated 1000 times. Both the means and the standard deviations of the servers' aggregate throughput are plotted in the figures. As shown in Fig. 8, when the number of senders was small and TCP Reno and Vegas did not experience incast collapse, the throughput of DC-Vegas was slightly lower. As the number of senders increased, TCP Reno and Vegas experienced incast-caused throughput collapse, DC-Vegas still achieved a stable high throughput. On average, network utilization achieved by DC-Vegas is much higher than that achieved by the other two algorithms. In addition, we test the performance of 1-win TCP, which fixes its window size at 1, as a reference. According to the analysis in Hwang et al. (2012), when the sender number is large, the performance of 1-win TCP can be considered an approximation of the TCP performance upper bound in incast scenarios. The results in Fig. 8 show that the throughput of DC-Vegas is very close to the upper bound approximated by 1-win TCP when the number of senders was greater than 20. These experimental results indicate that DC-Vegas is an effective algorithm for TCP incast collapse avoiding.

An all-to-all incast scenario was used in our experiments to investigate the behavior of DC-Vegas when incast congestion occurred simultaneously on multiple switch ports. In the all-to-all experiments, 41 machines were used. Each machine requested 25 KB of data from the remaining 40 machines (totaling 1 MB of data) simultaneously, resulting in 40 simultaneous incasts. The throughput CDF of experiments repeated 1000 times is shown in Fig. 9. As shown in the figure, the capacity of the 41 ports was nearly fully utilized for DC-Vegas, while TCP Reno and Vegas performed very poorly. 95% and 89% of the TCP Reno and Vegas connections experienced at least one timeout, respectively. The corresponding percentage for the DC-Vegas connections was only 0.6%. Moreover, performance of 1-win TCP was also tested in the all-to-all incast scenario. The results in Fig. 9 show that the performance of DC-Vegas is very close to the 1-win TCP.

According to the analysis given in Section 3.2, DC-Vegas is a distributed DCTCP, and its congestion behaviors are very close to DCTCP, which is why DC-Vegas achieves excellent incast collapse avoidance performance in experiments of this subsection.

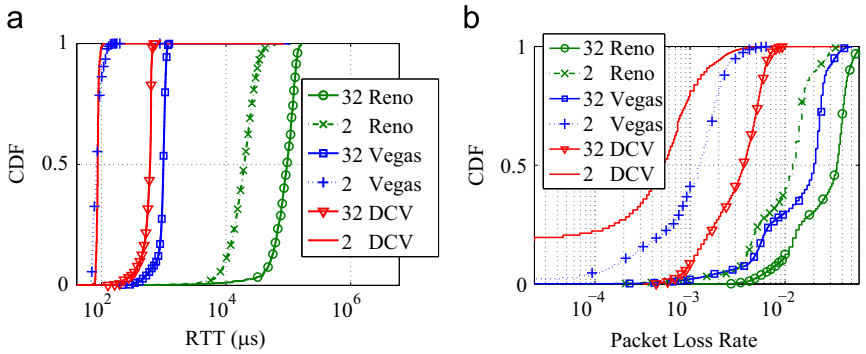

Fig. 7. RTT and PLR in the bottleneck. (a) CDF of RTT and (b) CDF of PLR.

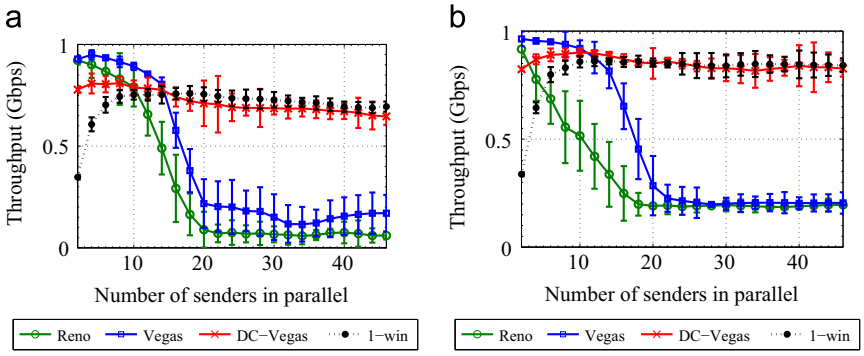

Fig. 8. Incast experiments. (a) Total traffic=1 MB and (b) Total traffic=4 MB.

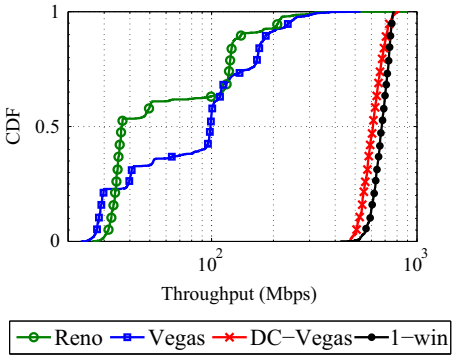

Fig. 9. All-to-all incast.

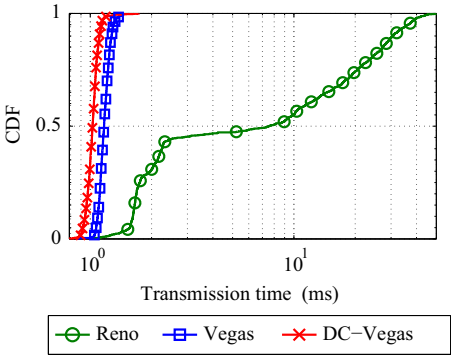

Fig. 10. Transmission time of mouse.

4.2.2. Queuing delay

In datacenter networks, extensive queuing delay incurred by long-term flows may cause severe performance impairment to delaying sensitive small flows of OLDI applications. We used four machines of the test bed, with one used as a receiver and the other three as senders, to investigate the interaction between throughput-sensitive long-term flows (elephant flows) and delay-sensitive short-term flows (mouse flows) for DC-Vegas in datacenter networks. As reported in Alizadeh et al. (2010), the 75th percentile of mouse flows coexist with two elephant flows; therefore, we let two elephant flows, which send packets as fast as they can, start from two of the senders, with the other sender establishing a mouse connection to transmit 20 KB data blocks to the receiver. Figure 10 shows the CDF for transmission times of 1000 data blocks. As shown in the figure, the transmission times of DC-Vegas and TCP Vegas were much lower than TCP Reno. Table 5 lists the average aggregate throughput of the two elephant flows. As shown in the table, the mouse flow for DC-Vegas and TCP Vegas did not affect the throughput of elephant flows. On the contrary, the elephant flows of TCP Reno were degraded by the mouse flow. The standard deviation of the elephant flows' aggregate throughput is also given in Table 5. The transmission is more stable with DC-Vegas than with the other algorithms.

Further experimental results of DC-Vegas queuing delay tests are given in Figs. 11 and 12. In these experiments, one of the machines was designated as the client, with 40 other machines as servers. A mouse flow and an elephant flow were established between each server and the sole client. The client repeatedly downloaded a 100 MB file using the elephant flow and a 100 KB/1 MB small file using the mouse flow from each server. The time interval between two downloads using elephant flows was 1 s, and it was 100 ms for mouse flows. We set 5 ms and 50 ms deadlines for mouse flow downloading 100 KB and 1 MB files, respectively. The deadline missing rates of different algorithms for different fan-in servers are shown in Fig. 11. The missed deadline rate is markedly reduced in the DC-Vegas case, which could significantly improve the user experience of OLDI applications (Wilson et al., 2011; Vamanan et al., 2012). Figure 12 plots the aggregated throughput of elephant flows for different algorithms. The figure shows that DC-Vegas achieved above-80% network capacity, or approximately full utilization (plus the throughput of mouse flows), implying that the improved throughput for the mouse flows using DC-Vegas did not come at the cost of the elephant flows.

Table 5
Mean and standard deviation of elephant flows throughput.

Indicator	Reno	Vegas	DC-Vegas
Mean (Mbps)	514	865	861
Deviation (Mbps)	162	38	21

As analyzed in Section 2, the stable state queue length of DC-Vegas is very low. This feature forbids elephant flows in the experiments from building long in-flight packets queue in network buffer. Moreover, the multiplicative window size decrease of DC-Vegas ensures elephant flows to make timely backoff when a mouse flow arrived. These two features enable DC-Vegas to perform well in the experiments of this section.

4.3. Fairness and convergence

To investigate fairness and convergence of DC-Vegas in datacenter networks, we tested DC-Vegas in a sequential starting/stopping experiment. In the experiments, six hosts in the test bed were used, with one as the receiver and the others as senders. TCP flows from the senders to the receiver were started and stopped sequentially at 150-s intervals. Figure 13 shows the time series of throughput variations of different algorithms in the sequential starting/stopping experiments. As the figures show, DC-Vegas quickly converged to a fair sharing point as flows joined and left. TCP Vegas converged to fairness quickly with high-level variations. The fairness and convergence of Reno were much worse than were those of the two delay-based algorithms. In datacenter networks,

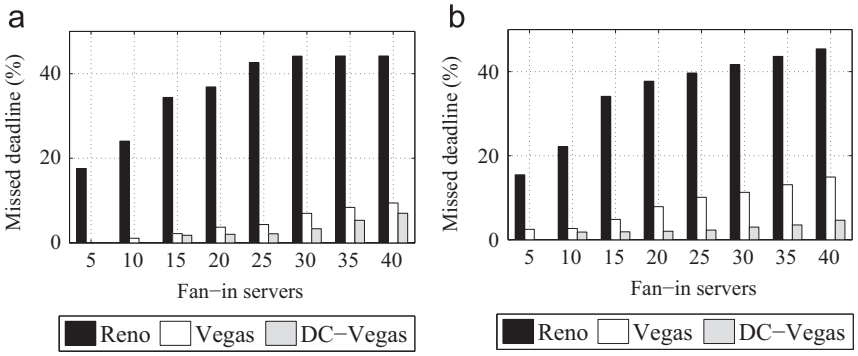

Fig. 11. Missed deadline rate of mouse flows. (a) 100 KB, 5 ms deadline and (b) 1 MB, 50 ms deadline.

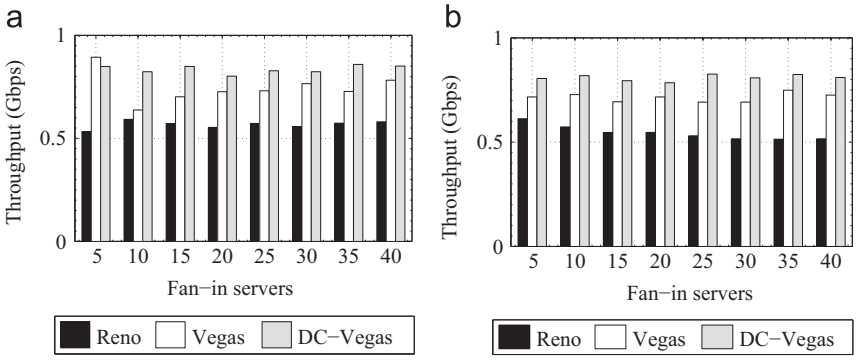

Fig. 12. Throughput of elephant flows. (a) 100 KB and (b) 1 MB.

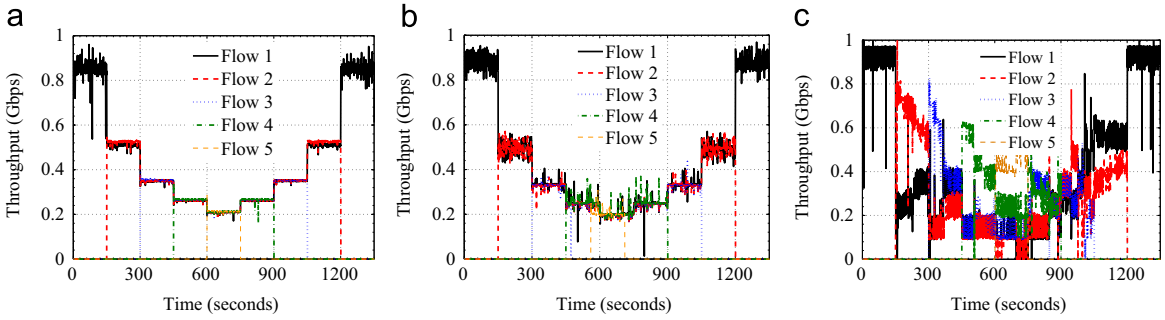

Fig. 13. Fairness and convergence of sequential starting experiments. (a) DC-Vegas, (b) TCP Vegas and (c) TCP Reno.

the loss-based TCP Reno algorithm introduces many burst packet losses which are very detrimental to fairness and convergence. In contrast, DC-Vegas and TCP Vegas do not introduce high queuing delay or packet loss during their data transmissions and therefore can achieve improved fairness and convergence. Moreover, low queuing packet length also can improve the fairness and convergence of DC-Vegas. These factors enable DC-Vegas to achieve excellent fairness and convergence in experiments of this subsection.

4.4. *RTT noise*

Delay-based TCPs are more susceptible to latency noise than loss-based and ECN-based. The RTT noise harm the congestion measurement accuracy in two ways: (1) *Impulse noise*: RTT impulse caused by random noise sometimes are one or two orders of magnitude larger than regular RTT samples, so it can lead to serious congestion false alarms. (2) *Uniform noise*: Random noise that uniformly distributes over all timelines can cause a system-level harm to delay-based TCP congestion detection. When we designed DC-Vegas, we took the drawback of delay-based TCP into consideration, and introduced features to protect DC-Vegas from RTT noise: First of all, DC-Vegas uses the fraction of over-threshold queuing delay samples, F_{dcv}, to indicate network congestion level. This congestion indication mechanism can filter influences of large impulse noise. Moreover, DC-Vegas uses an exponential moving average to smooth F_{dcv}, which can further weaken influences of uniform RTT noise. Figure 14 shows a throughput stability comparison of TCP Vegas and DC-Vegas over a 100 Mbps bandwidth link. RTT of the link follows a normal distribution with 10 ms mean and 100 standard deviation. As shown in the figure, the robustness of DC-Vegas against RTT noise is much better than that of TCP Vegas.

5. NS-2 simulation experiments

The experiments described in Section 4 were conducted in a real datacenter test bed which does not support ECN, so algorithms such as DCTCP could not be deployed. To compare DC-Vegas with state-of-the-art datacenter algorithms such as DCTCP, we tested the performance of DC-Vegas using the ns-2 simulator in this section. The ns-2 code for DCTCP was taken from

http://www.stanford.edu/alizade/Site/DCTCP.html. In all experiments, the RTT of the link was 50 μs, and the buffer size was equal to one-half the network bandwidth delay product (BDP).

First, DC-Vegas was compared with DCTCP, TCP Vegas and Reno in incast scenarios. Sixty TCP servers synchronously sent a volume of data to a client through a 1 Gbps/10 Gbps link. The total traffic volume size was 1 MB for the 1 Gbps link experiments and 10 MB for the 10 Gbps experiments. The experimental results in Fig. 15 show that DC-Vegas performed comparably to DCTCP and much better than others. Then, for low latency applications, 40 mouse flows and 40 elephant TCP flows shared a link of 1 Gbps/10 Gbps bandwidth. The mouse flows sent 100 KB (for 1 Gbps simulations) or 1 MB (for 10 Gbps) files 1000 times, while the elephant flows sent data as quickly as they could. The transmission deadline of the experiment was set at 5 ms for the 1 Gbps experiments and 50 ms for 1 Gbps. The rates of missed deadlines are plotted in Fig. 16, again showing very similar performances of DC-Vegas and DCTCP, both of which were observably better than others. Finally, we evaluated DC-Vegas in the multi-bottleneck topology shown in Fig. 17, which was used in Alizadeh et al. (2010). There are two bottlenecks in the configuration: the 10 Gbps link passed by S1 and S2 flows, and the 1 Gbps switch passed by S2 and S3 flows. Table 6 lists the throughput, fairness, and timeout packet loss rate of different algorithms. As shown in the table, the performances of DCTCP and DC-Vegas were still very similar. As shown in Section 2, the congestion control behavior of DC-Vegas is very close to DCTCP and can be regarded as a distributed version of DCTCP. This feature causes the performance of DC-Vegas in the ns-2 testbed which is very close to performance of the state-of-the-art Data Center TCP algorithm.

6. Related work

In recent years, many solutions have been proposed to address the challenges of using TCP in data center networks (Kushwaha and Gupta, 2014; Lin et al., 2013). These solutions can be classified into two categories: TCP-based solutions and non-TCP solutions.

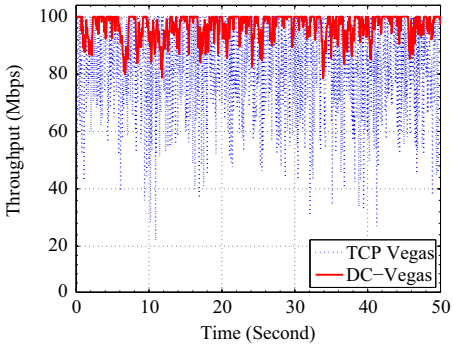

Fig. 14. Throughput stability comparison of TCP Vegas and DC-Vegas.

TCP-based solutions: Fine-grained TCP retransmissions (Vasudevan et al., 2009) are proposed to reduce the incast penalty of timeouts by using microsecond granularity retransmission timers, which can reduce the timeout waiting time when incast congestion occurs, but cannot avoid incast congestion and address the long queuing delay problem. Data Center TCP (DCTCP) (Alizadeh et al., 2010) is an ECN-based TCP congestion control algorithm proposed to avoid incast timeout and achieve low queuing delay, which achieves very good performance in data center netowrks. ECN* (Wu et al., 2012) tries to achieve a performance comparable with DCTCP by only modifying switch ECN marking strategy. Deadline-aware Data Center TCP (D^2TCP) (Vamanan et al., 2012) adds deadline awareness to DCTCP using a gamma correction function. L^2DCT (Munir et al., 2015) introduces Least Attained Service (LAS) scheduling into DCTCP to reduce completion times of short flows. DCTCP, ECN*, D^2TCP, and L^2DCT make important contributions for data center congestion control, but all require ECN, which are not always available in existing production data centers. ICTCP (Wu et al., 2010) proposes a receiver driven TCP congestion control algorithm to avoid incast congestion happened in the last-hop of a TCP session. The ICTCP uses delay-based congestion detection, and only modifies TCP protocol stack on the receiver side. The benefit of DC-Vegas as compared with ICTCP is that DC-Vegas can avoid incast congestion happened on any bottleneck link of a network, while ICTCP only works for last hop congestion. Lee et al. (2012) recommend delay-based congestion detection as a future direction for data center congestion control. Although they did not give any specific algorithms, their work is very inspiring.

Non-TCP solutions: Krevat et al. (2007) discuss potential application-level approaches for resolving the incast problem in the paper. FQCN avoids incast throughput collapse of TCP using Quantized Congestion Notification (QCN) by improving the fairness of multiple flows sharing a bottleneck. HCF proposed a switch-based approach to prevent data centers from incast throughput collapse. HULL (Alizadeh et al., 2012a) is a new network congestion control architecture that tradeoffs a little network bandwidth for ultra-low latency. The design targets of HULL are for ultra-low latency applications such as high-frequency trading and RAM-Cloud, which have microsecond-level latency requirements. HULL uses phantom queues for congestion detection, ECN-based congestion control, and custom packet pacing hardware to achieve this performance. DeTail (Zats et al., 2012) avoids missed deadline application flows by reducing the flow completion time tail. DeTail constructed a cross-layer network stack that involves quick congestion detection at the link layer and lower-congestion paths routing at the network layer to achieve its target. The approach used in pFabric (Alizadeh et al., 2012b) is TCP-compatible and provides near-optimal performance in terms of completion time for high-priority flows and network utilization. pFabric achieves the target by redesigning data center fabric. D^3 (Wilson et al., 2011) introduces deadline awareness into data center congestion control, and analogously, PDQ (Hong et al., 2012) introduces preemptive scheduling, which are very important contributions for data center congestion

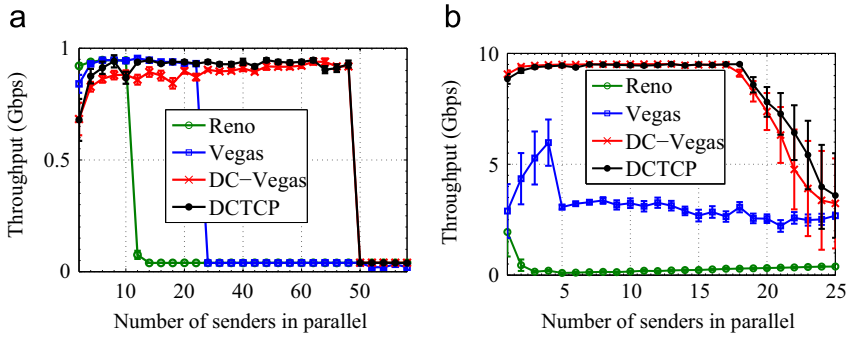

Fig. 15. Incast experiments over ns-2. (a) 1 MB, 1 Gbps and (b) 10 MB, 10 Gbps.

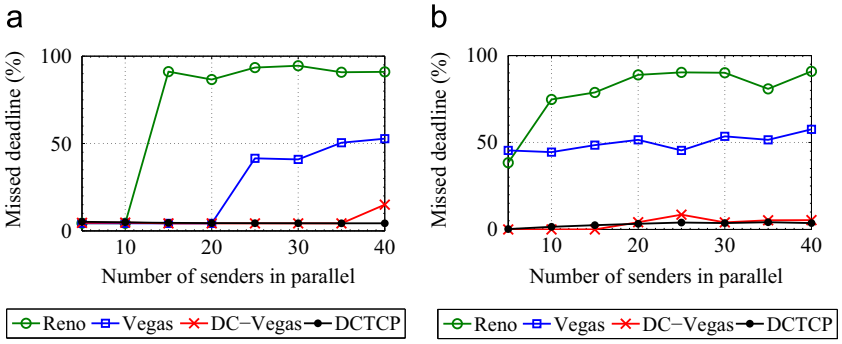

Fig. 16. Low latency experiments over ns-2. (a) 100 KB, 5 ms deadline, 1 Gbps and (b) 1 MB, 50 ms deadline, 10 Gbps.

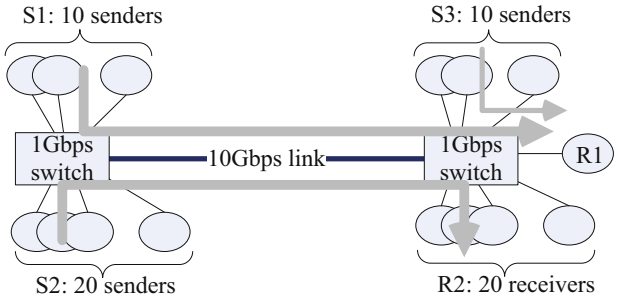

Fig. 17. Multi-bottleneck experiments topology.

Table 6
Multi-bottleneck experiments.

Indicator	Reno	Vegas	DC-Vegas	DCTCP
Throughput (**S1**+**S2**)	7.43 Gbps	8.87 Gbps	9.51 Gbps	9.88 Gbps
Throughput (**S1**+**S3**)	0.67 Gbps	0.81 Gbps	0.95 Gbps	0.92 Gbps
Jain's index (**S1**, **S3**)	0.778	0.923	0.925	9.922
Jain's index (**R1**, **R2**)	0.633	0.965	0.981	0.987
TimeOut rate (**S1**)	0.011%	0.001%	0%	0%
TimeOut rate (**S2**)	0.025%	0.002%	0%	0%
TimeOut rate (**S3**)	1.002%	0.346%	0.002%	0.003%

control. However, both D^3 and PDQ require modifications to the switch and are incompatible with TCP, presenting difficulties for deployment in existing data centers.

Although these TCP/non-TCP solutions achieved very important contributions for datacenter congestion control, when these are deployed in existing production data centers, ECN requirements, modifications to the switch software/hardware, network protocol stacks, and/or application software are unavoidable, leading to high deployment cost and complexity.

The deployment complexity of congestion control technologies in datacenters attracted the attention of researchers in recent years. In the paper (Goswami et al., 2014), Bharti and Pattanaik implement traffic shaping mechanism in the edge switch at source that acts proactively and prevents the propagation of ill effects due to sustained burst. In the paper, Bharti and Pattanaik (2014) propose a dynamic distributed flow scheduling mechanism for effective link utilization and load balancing. These papers provide good technique for congestion mitigation as the technique is relatively simple to implement and yet it is quite impactful.

7. Conclusions

In this paper, DC-Vegas, a new delay-based TCP algorithm for datacenter congestion control, is proposed. DC-Vegas can achieve a performance comparable to state-of-the-art datacenter TCP algorithms and requires only minimal system modifications, i.e., only the update of TCP senders. Receiver side modifications, ECN support, and/or updates to other parts of datacenter networks are unnecessary, which is very valuable for existing production data centers. Experiments conducted in a production datacenter test bed as well as ns-2 simulations confirmed the performance (including throughput, latency, fairness, and convergence) of the proposed algorithm.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant no. 61202426), the National Science Fund for Distinguished Young Scholars of China (Grant no. 61125102), the State Key Program of National Natural Science of China (Grant no. 61133008), and National High Technology Research and Development Program of China (No. 2013AA01A601).

References

A DCTCP implementation for ns-2. ⟨http://www.stanford.edu/alizade/Site/DCTCP.html⟩.
Alizadeh M, Greenberg A, Maltz D, Padhye J, Patel P, Prabhakar B, et al. Data center TCP (DCTCP). In: Proceedings of ACM SIGCOMM 2010; 2010. p. 63–74.
Alizadeh M, Kabbani A, Edsall T, Prabhakar B, Vahdat A, Yasuda M. Less is more: trading a little bandwidth for ultra-low latency in the data center. In: Proceedings of the ninth USENIX NSDI; 2012. p. 19.
Alizadeh M, Yang S, Katti S, McKeown N, Prabhakar B, Shenker S. Deconstructing datacenter packet transport. In: Proceedings of ACM HotNets 2012; 2012. p. 133–38.
Alrshah MA, Othman M, Ali B, Hanapi ZM. Comparative study of high-speed linux TCP variants over high-BDP networks. J Netw Comput Appl 2014;43:66–75.
Bauer S, Beverly R, Berger A. Measuring the state of ECN readiness in servers, clients, and routers. In: Proceedings of the ACM IMC 2011. p. 171–80.
Bharti S, Pattanaik K. Dynamic distributed flow scheduling for effective link utilization in data center networks. J High Speed Netw 2014;20(1):1–10.
Brakmo LS, O'Malley SW, Peterson LL. TCP Vegas: new techniques for congestion detection and avoidance. In: Proceedings of ACM SIGCOMM 2002; 1994. p. 24–35.
Chen Y, Griffith R, Liu J, Katz R, Joseph A. Understanding TCP incast throughput collapse in datacenter networks. In: Proceedings of ACM WREN 2009; 2009. p. 73–82.
Goswami A, Pattanaik K, Bharadwaj A, Bharti S. Loss rate control mechanism for fan-in-burst traffic in data center network. Proc Comput Sci 2014;32:125–32.
Ha S, Rhee I, Xu L. CUBIC: a new TCP-friendly high-speed TCP variant. ACM SIGOPS Oper Syst Rev 2008;42(5):64–74.
He Y, Tan H, Luo W, Feng S, Fan J. MR-DBSCAN: a scalable mapreduce-based DBSCAN algorithm for heavily skewed data. Front Comput Sci 2014;8(1):83–99.
Hong C, Caesar M, Godfrey P. Finishing flows quickly with preemptive scheduling. In: Proceedings of ACM SIGCOMM 2012; 2012. p. 127–38.
Hwang J, Yoo J, Choi N. IA-TCP: a rate based incast-avoidance algorithm for TCP in data center networks. In: Proceedings of the 2012 IEEE international conference on communications; 2012.
Jacobson V. Congestion avoidance and control. In: Proceedings of ACM SIGCOMM 1988; 1988. p. 314–29.
Jingyuan W, Jiang Y, Chao L, Ouyang Y, Xiong Z. Improving the incast performance of datacenter TCP by using rate-based congestion control. IEICE Trans Fundam Electron Commun Comput Sci 2014;97(7):1654–8.

Krevat E, Vasudevan V, Phanishayee A, Andersen D, Ganger G, Gibson G, Seshan S. On application-level approaches to avoiding TCP throughput collapse in cluster-based storage systems. In: Proceedings of the petascale data storage workshop, supercomputing 2007; 2007. p. 1–4.

Kushwaha V, Gupta R. Congestion control for high-speed wired network: a systematic literature review. J Netw Comput Appl 2014;45:62–78.

Lee C, Jang K, Moon S. Reviving delay-based TCP for data centers. In: Proceedings of ACM SIGCOMM 2012; 2012. p. 111–12.

Lin J, Zha L, Xu Z. Consolidated cluster systems for data centers in the cloud age: a survey and analysis. Front Comput Sci 2013;7(1):1–19.

Meisner D, Sadler C, Barroso L, Weber W, Wenisch T. Power management of online data-intensive services. In: Proceeding of the 38th annual international symposium on computer architecture; 2011. p. 319–30.

Munir A, Qazi I, Uzmi Z, Mushtaq A, Ismail S, Iqbal M, et al. Minimizing flow completion times in data centers. In: Proceedings of IEEE INFOCOM 2013. p. 2157–165.

Padhye J, Firoiu V, Towsley D, Krusoe J. Modeling TCP throughput: a simple model and its empirical validation. In: Proceedings of ACM SIGCOMM 1998; 1998. p. 303–14.

Rabiner L, Juang B. An introduction to hidden Markov models. IEEE ASSP Mag 1986a;3(1):4–16.

Rabiner L, Juang B-H. An introduction to hidden Markov models. IEEE ASSP Mag 1986b;3(1):4–16.

State of the data center. ⟨http://www.emersonnetworkpower.com/en-US/Solutions/infographics/Pages/2011DataCenterState.aspx⟩; 2011.

Stewart R, Tüxen M, Neville-Neil G. An investigation into data center congestion control with ECN. In: Proceedings of BSDCan 2011; 2011.

Vamanan B, Hasan J, Vijaykumar T. Deadline-aware datacenter TCP (D^2TCP). In: Proceedings of ACM SIGCOMM 2012; 2012. p. 115–26.

Vasudevan V, Phanishayee A, Shah H, Krevat E, Andersen D, Ganger G, Gibson G, Mueller B. Safe and effective fine-grained TCP retransmissions for datacenter communication. In: Proceedings of ACM SIGCOMM 2009; 2009. p. 303–14.

Wang J, Jiang Y, Ouyang Y, Li C, Xiong Z, Cai J. TCP congestion control for wireless datacenters. IEICE Electron Express 2013a;10(12):20130349.

Wang J, Wen J, Han Y, Zhang J, Li C, Xiong Z. Cubic-fit: a high performance and tcp cubic friendly congestion control algorithm. IEEE Commun Lett 2013b;17(8):1664–7.

Wang J, Wen J, Han Y, Zhang J, Li C, Xiong Z. Achieving high throughput and tcp reno fairness in delay-based tcp over large networks. Front Comput Sci 2014a;8(3):426–39.

Wang H, Li F, Zhou X, Cao Y, Qin X, Chen J, Wang S. HC-store: putting MapReduces foot in two camps. Front Comput Sci 2014b;8(6):859–71.

Wei DX, Jin C, Low SH, Hegde S. FAST TCP: motivation, architecture, algorithms, performance. IEEE/ACM Trans Netw 2006;14(6):1246–59.

Welch LR. Hidden Markov models and the Baum–Welch algorithm. IEEE Inf Theory Soc Newslett 2003;53(4):10–3.

Wilson C, Ballani H, Karagiannis T, Rowtron A. Better never than late: meeting deadlines in datacenter networks. In: Proceedings of ACM SIGCOMM 2012; 2011. p. 50–61.

Wu H, Feng Z, Guo C, Zhang Y. ICTCP: incast congestion control for TCP in data center networks. In: Proceedings of ACM CoNEXT 2010; 2010. p. 13–25.

Wu H, Ju J, Lu G, Guo C, Xiong Y, Zhang Y. Tuning ECN for data center networks. In: Proceedings of ACM CoNEXT 2012; 2012. p. 25–36.

Xu W, Zhou Z, Pham D, Ji C, Yang M, Liu Q. Hybrid congestion control for high-speed networks. J Netw Comput Appl 2011;34(4):1416–28 (Advanced Topics in Cloud Computing).

Zats D, Das T, Mohan P, Borthakur D, Katz R. DeTail: reducing the flow completion time tail in datacenter networks. In: Proceedings of ACM SIGCOMM 2012; 2012. p. 139–50.

Zhang Y, Liao X, Jin H, Lin L, Lu F. An adaptive switching scheme for iterative computing in the cloud. Front Comput Sci 2014;8(6):872–84.