Performance Evaluation of TCP Congestion Control Algorithms in Data Center Networks

Truc Anh N. Nguyen*, Siddharth Gangadhar*, and James P.G. Sterbenz*[†][‡]
*Information and Telecommunication Technology Center
Department of Electrical Engineering and Computer Science
The University of Kansas, Lawrence, KS 66045, USA
[†]School of Computing and Communications (SCC) and InfoLab21
Lancaster University, LA1 4WA, UK
[‡]Department of Computing
The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong
{annguyen|siddharth|jpgs}@ittc.ku.edu www.ittc.ku.edu/resilinets

ABSTRACT

TCP congestion control has been known for its crucial role in stabilizing the Internet and preventing congestion collapses. However, with the rapid advancement in networking technologies, resulting in the emergence of challenging network environments such as data center networks (DCNs), the traditional TCP algorithm leads to several impairments. The shortcomings of TCP when deployed in DCNs have motivated the development of multiple new variants, including DCTCP, ICTCP, IA-TCP, and D^2TCP, but all of these algorithms exhibit their advantages at the cost of a number of drawbacks in the Global Internet. Motivated by the belief that new innovations need to be established on top of a solid foundation with a thorough understanding of the existing, well-established algorithms, we have been working towards a comprehensive analysis of various conventional TCP algorithms in DCNs and other modern networks. This paper presents our first milestone towards the completion of our comparative study in which we present the results obtained by simulating multiple TCP variants: NewReno, Vegas, HighSpeed, Scalable, Westwood+, BIC, CUBIC, and YeAH using a fat tree architecture. Each protocol is evaluated in terms of queue length, number of dropped packets, average packet delay, and aggregate bandwidth as a percentage of the channel bandwidth.

CCS Concepts

•Networks → Transport protocols; Network simulations; Network performance analysis;

Keywords

TCP, dekay-based, loss-based, hybrid congestion control, data center, NewReno, HighSpeed, STCP, YeAH, BIC, CUBIC, Westwood+, Vegas, fat tree, ns-3, performance evaluation, queue buildup, incast, outcast, buffer pressure, Future Internet

1. INTRODUCTION

The scalability and robustness of the Transmission Control Protocol (TCP) have been proven through its dominant role in the Internet, with its congestion control algorithm contributing to the stability of this global network. However, the evolution of networking technologies with the emerging of modern network environments such as satellite channels and data center networks (DCNs), each with distinct characteristics have been challenging the well-established TCP. For DCNs, even the newest TCP variants designed to balance among various constraints of a state-of-the-art congestion control algorithm can suffer from performance in general, since what is considered as an ideal design goal in one environment may not suit the others.

Measurements have shown that the majority of traffic within data centers is TCP flows that comprise of a mixture of 3 different types: mice, cat, and elephant traffic [9, 32]. Query traffic or mice flows are small in size (less than 100 KB) and come from those applications that require fast response time such as Google search and Facebook updates. These flows tend to experience TCP incast impairment in which the flows collide at an upper-level switch, exhausting the switch memory or its buffer capacity and resulting in packet losses due to the simultaneous transfer from multiple hosts/workers. The second type of traffic in DCNs is delay-sensitive short messages (cat traffic) that have a size between 100 KB and 5 MB generated by applications such as YouTube or Picasa. The last traffic type is throughput-sensitive long flows (elephant traffic) of more than 5 MB in size generated by those applications such as software updates. A data center also needs to process data coming from outside or another data center, for example when an end user uses a cloud application.

In addition to the incast impairment, TCP in DCNs also suffers from outcast, queue buildup, and buffer pressure problems. TCP outcast occurs when a small set of traffic flows looses their throughput share to a larger set when they arrive at the same output port due to the characteristic of drop-tail queues [32]. Queue buildup happens when short traffic flows experience long queuing delay because their

CFI'16, June 15-17, 2016, Nanjing, China

DOI: http://dx.doi.org/10.1145/2935663.2935669

packets are buffered behind those of longer flows while all of them traverse the same queue [9]. Lastly, the buffer pressure issue arises when the greedy elephant flows occupy most of the bottleneck queue buffer space, leaving little room for the mice traffic, resulting in packets being dropped [9]. The impairments of TCP in DCNs have motivated the development of multiple TCP variants to address TCP issues, including DCTCP [9], ICTCP [36], IA-TCP [23], and D^{2}TCP [33]. However, these variants gain their advantages at the cost of many drawbacks in the general wide-area Global Internet [32]. An ideal congestion control for DCNs should be able to efficiently utilize the available network bandwidth and experience low packet latency with small buffer occupancy. Other design constraints such as scalability, robustness, and deployment complexity also need to be considered. Moreover, as an end-to-end algorithm, the protocol should be able to cooperate with commodity switches off-the-shelf at its full potential without any requirements to modify the network devices.

We believe that every new TCP variant development needs to be established on top of a comprehensive understanding of the existing mechanisms in order to arrive at a better solution without reinventing the wheel. Unfortunately, the research community still lacks such a comprehensive comparative study of conventional TCP congestion control algorithms in the context of modern networking. Although this paper only presents our initial set of results in DCNs, it sets our first milestone towards our ambitious goal of a thorough analysis of existing general-purpose TCP variants under different network environments, including satellite networks, in addition to further extend the DCN study. We hope that our work will provide useful detailed insights into these algorithms, serving as a reference for the research community towards the development of better TCP congestion control algorithms that are adaptable to the characteristics of the underlying network while satisfying the application requirements specific to a particular environment.

In this paper, we use the open-source network simulator ns-3 [1] due to its effective scalability for large-scale simulations with low memory usage [34]. We study the performance of three TCP congestion control categories: loss-based with NewReno, HighSpeed, Scalable, Westwood+, BIC, and CUBIC, delay-based with Vegas, and hybrid with YeAH-TCP in DCNs using a 6-ary fat tree architecture. For each protocol, we measure the queue length, total number of packet drops, average packet delay, and aggregate throughput as a percentage of the available network bandwidth.

The remainder of our paper is organized as follows: Section 2 briefly presents the theoretical background of the TCP variants studied in our paper. Section 3 explains our simulation setup and an analysis of the obtained results. Section 4 concludes our paper with directions for future work.

2. BACKGROUND AND RELATED WORK

This section provides the theoretical background of the conventional TCP congestion control algorithms evaluated in our paper, including NewReno, Vegas, HighSpeed, Scalable, Westwood+, BIC, CUBIC, and YeAH. The key features of these protocols are summarized in Table 1.

2.1 TCP Congestion Control Algorithms

Our paper evaluates three different categories of TCP congestion detection algorithms, including loss-based, delay-based, and hybrid. The loss-based algorithms (NewReno, Westwood+, HighSpeed, Scalable, BIC, and CUBIC) rely on packet loss events to detect congestion in network channels while the delay-based variants (Vegas) interpret the increasing delay due to queue overflow as an indication of congestion. All hybrid mechanisms (YeAH) incorporate the features from the first two categories.

2.1.1 NewReno

NewReno [21] implements the standard AIMD (additive increase multiplicative decrease) congestion control algorithm, known as the TCP Reno variant described in RFC 5681 [10]. Specifically, NewReno increases its congestion window (cwnd) exponentially during the slow start phase, which is equivalent to the cwnd increment in Equation 1 upon every new ACK arrival until a loss happens.

$$\text{cwnd} = \text{cwnd} + 1 \tag{1}$$

When the cwnd reaches the slow start threshold (ssthresh) value, NewReno slows down its window growing rate, allowing the increment of cwnd by 1 only every RTT, or by 1/cwnd every ACK receipt until a loss happens as in Equation 2.

$$\text{cwnd} = \text{cwnd} + \frac{1}{\text{cwnd}} \tag{2}$$

When a loss is detected through the arrival of three duplicate ACKs (dupACKs), NewReno enters fast retransmit, resends the lost segment, halves its ssthresh and modifies its cwnd according to Equations 3 and 4, respectively.

$$\text{ssthresh} = \frac{\text{cwnd}}{2} \tag{3}$$

$$\text{cwnd} = \text{ssthresh} + 3 \tag{4}$$

Fast recovery then governs the data transmission until a new ACK arrives signifying the recovery of the lost segment. NewReno augments Reno's fast recovery phase by introducing a mechanism for responding to *partial acknowledgments*, those ACKs that acknowledge only part of the data transmitted before the loss detection. NewReno remains in the recovery phase, trying to recover additional missing segments when these ACKs are received. This mechanism allows the protocol to recover from multiple packet losses happening in a single sending window when TCP selective acknowledgment (SACK) option [29] is not available.

2.1.2 Vegas

TCP Vegas [12] is a pure delay-based congestion control algorithm that proactively prevents the loss of data segments. In its steady state, Vegas linearly increases/decreases its sending rate to ensure a very small number of extra packets queued up at the bottleneck at any given time (backlog). Vegas bases its congestion window modification on its measurements of the RTT and the actual throughput achieved by a connection because the changes in these metrics reflect the network dynamics, especially when it approaches congestion. Vegas estimates its backlog (the diff rate) using Equation 7, where the actual and expected throughputs are calculated in Equations 5 and 6, respectively.

$$\text{actual} = \frac{\text{cwnd}}{\text{RTT}} \tag{5}$$

$$\text{expected} = \frac{\text{cwnd}}{\text{BaseRTT}} \tag{6}$$

$$\text{diff} = \text{expected} - \text{actual} \tag{7}$$

BaseRTT (Eq. 6) denotes the minimum of all RTT measurements during a connection lifetime.

2.1.3 HighSpeed

HighSpeed TCP (HSTCP) [14] improves the performance of standard TCP in high bandwith-×-delay product (BDP) networks by making the additive increase and multiplicative decrease factors functions of the current window size when the congestion window grows large (beyond the value of Low_Window). Specifically, the increase and decrease parameters are calculated using Equations 8 and 9, respectively. The packet drop rate $p(w)$ is also a function of current congestion window size as shown in Equation 10. A number of other parameters are used in these calculations, including High_Decrease with the usual default value of 0.1, W or Low_Window with the default value of 38, and W_1 or High_Window with the default value of 83000.

$$a(w) = w^2 \times p(w) \times 2 \times \frac{b(w)}{(2 - b(w))} \tag{8}$$

$$b(w) = \frac{(\text{High\_Decrease} - 0.5)(\log(w) - \log(W))}{\log(W_1) - \log(W)} + 0.5 \tag{9}$$

$$p(w) = \frac{0.078}{w^{1.2}} \tag{10}$$

2.1.4 Scalable

Scalable TCP (STCP) [25] is a simple modification to standard TCP additive increase and multiplicative decrease factors to enhance performance for bulk data transfers in high-speed wide area networks. STCP increases its congestion window aggressively during its steady state to allow full utilization of high BDP links while slowly decreasing the window to shorten TCP recovery time after the occurrence of a loss. During its congestion avoidance phase, STCP increases its cwnd by 0.01 for every new ACK until some data loss is detected as in Equation 11. On an occurrence of congestion, the ssthresh is reduced by a factor of 0.125 as in Equation 12.

$$\text{cwnd} = \text{cwnd} + 0.01 \tag{11}$$

$$\text{ssthresh} = \text{cwnd} - \lceil 0.125 \times \text{cwnd} \rceil \tag{12}$$

2.1.5 Westwood+

Similar to TCP Westwood [27], Westwood+ [28] employs a bandwidth estimation mechanism to adjust its congestion window. However, instead of performing the sampling every ACK receipt as in Westwood, Westwood+ measures the bandwidth every RTT to alleviate the protocol's aggressiveness in the presence of ACK compression [30].

2.1.6 BIC

BIC (Binary Increase Congestion Control) [37] improves TCP RTT fairness while effectively utilizing the available bandwidth of high BDP network environments. At the heart of BIC are four algorithms: binary search increase, additive increase, slow start, and fast convergence. The binary search increase allows BIC to achieve TCP friendliness when the window size is small with its logarithmic increase function. The additive increase enhances the performance of long-RTT flows when they compete with connections that have shorter delay. The slow start probes for a new maximum window size that is used in the binary search and fast convergence algorithms when the current window reaches the saturation point. Finally, the fast convergence reduces the increase rate of larger window flows to allow those with smaller windows to catch up.

2.1.7 CUBIC

CUBIC [20] simplifies BIC congestion control algorithm complexity while further enhancing BIC friendliness when competing with standard TCP connections. CUBIC window growth is a cubic function of the elapsed time since the last time it reduces its sending rate due to a loss event (Equation 13), resulting in a concave followed by convex graph shape.

$$W(t) = C(t - K)^3 + W_{\max} \tag{13}$$

In the equation, C is the CUBIC parameter, K is the elapsed time from the last window reduction following a loss, and $W_{\max}$ is the congestion window size when the loss happened.

2.1.8 YeAH

YeAH (Yet Another Highspeed) [11] is a high-speed TCP that tries to achieve various design goals of an ideal congestion control algorithm. Vegas' method for estimating the network condition is employed to determine YeAH operation mode: the *fast* mode with more aggressive congestion window increment or the *slow* mode that is equipped with a precautionary decongestion algorithm. YeAH also stands out for its capability to detect the presence of legacy Reno flows and adjust its own sending rate properly to ensure a fair competition for network resources. The calculations of the backlog at the bottleneck queue in Equation 14 and the congestion level in Equation 15 are used to determine the transition between YeAH's operating modes.

$$Q = \text{RTT}_{\text{queue}} \times G = (\text{RTT}_{\text{min}} - \text{RTT}_{\text{base}}) \times \frac{\text{cwnd}}{\text{RTT}_{\text{min}}} \tag{14}$$

$$L = \frac{\text{RTT}_{\text{queue}}}{\text{RTT}_{\text{base}}} \tag{15}$$

The value of Q (Eq. 14) is also used to adjust the ssthresh upon the receipt of three dupACKs when YeAH is not competing with Reno flows as shown in Equation 16. Otherwise, ssthresh is halved to ensure friendliness.

$$\text{ssthresh} = \min[\max(\frac{\text{cwnd}}{8}, Q), \frac{\text{cwnd}}{2}] \tag{16}$$

Protocol	Target environment	Category	Design goals	Descriptions
NewReno	wired, low bandwidth, small delay	loss-based	recover multiple losses occuring in a single sending window	standard AIMD (AI factor = 1, MD factor = 1/2); exit fast recovery upon full ACK receipt
Vegas	wired, low bandwidth, small delay	delay-based	increase throughput while still competing fairly with Reno flows	proactively prevent packet losses; linear increase/linear decrease algorithm based on queue backlog
HighSpeed	high BDP	loss-based	fully utilize high bandwidth links without unrealistic requirement of a very low packet drop rate	modified AIMD with AI and MD factors functions of current window size under large congestion windows
Scalable	high-speed WANs	loss-based	better utilize high bandwidth links	modified AIMD with AI factor = 0.01 and MD factor = 0.125
Westwood+	wireless networks with high number of random losses	loss-based	improve throughput in the presence of corruption-based losses	adjust congestion window based on estimated bandwidth; perform bandwidth sampling every RTT
BIC	high BDP	los-based	tackle RTT unfairness, especially with the use of drop tail queues while still ensure friendliness and scalability	use binary search method to identify the sending rate
CUBIC	high BDP	loss-based	simplify BIC window control; enhance BIC friendliness	window growth a cubic function
YeAH	high BDP	hybrid	balance multiple design constraints: efficiency, intra fairness, friendliness, random loss robustness	dual operating modes based on queue backlog and network congestion level

Table 1: Summary of TCP congestion control algorithms

2.2 Related Work

To the best of our knowledge, this paper is the first that provides an evaluation of multiple conventional TCP congestion control algorithms in data center network environments. Furthermore, there have been very few papers comparing many TCP variants [19, 22].

3. EVALUATION

In this section, we present our simulation topology and parameters in detail followed by an analysis of the collected results that demonstrate the performance of each TCP algorithm using queue length, total number of dropped packets, average throughput as a percentage of the ideal bandwidth, and average packet delay as the metrics.

3.1 Simulation Topology and Parameters

Our simulation topology is the fat-tree architecture developed to support large-size data-center networks at a low cost, allowing the use of commodity switches while still being able to achieve full aggregation bandwidth and backward compatible with the current networking technologies and protocols (Ethernet, IP, and TCP) [8].

To simplify the illustration, we present a 4-ary fat tree with k equal to 4 in Figure 1. This tree consists of 4 pods. Each pod contains 2 level of switches: edge and aggregation with 2 ($k/2$) switches at each level. All edge switches are directly connected to the aggregation switches above and the hosts below them in the hierarchy. The aggregation switches connect to 2 ($k/2$) groups of core switches, allowing these upper-layer switches to communicate with every single pod. A 4-ary fat tree supports a total of 16 ($k^3/4$) hosts.

In the simulations for this paper, we use k equal to 6, resulting in a topology of 54 hosts. Although it is small comparing to the size of DCNs in practice, it is sufficient to

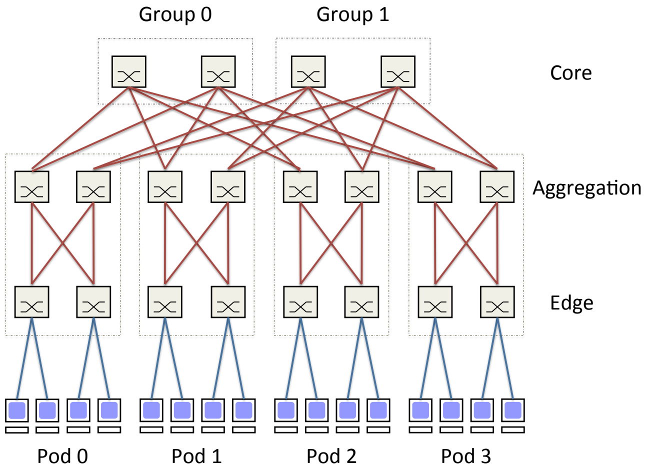


Figure 1: Simple fat-tree topology with $k = 4$

provide us helpful insights for our initial evaluation of the various TCP congestion control algorithms. Table 2 summarizes the parameters of the 6-ary fat tree.

k	6
Number of pods	6
Number of edge switches per pod	3
Number of aggregation switches per pod	3
Number of group of core switches	3
Number of core switches per group	3
Number of ports per switch	6
Total number of hosts in tree	54
Total number of edge switches in tree	18
Total number of aggregation switches in tree	18
Total number of core switches in tree	9

Table 2: Fat tree parameters with $k = 6$

Our test script is a TCP-version (in the current ns-3.25 standard release) of the publicly available ns-3 fat-tree implementation [5] under the NTU-DSI-DCN project [3]. Similar to the simulation setup discussed in Wong et. al. paper [35], traffic flow in the network is generated by randomly selecting (with uniform probability) pairs of communication hosts across all 54 end systems, resulting in 54 TCP connections simultaneously sending data and ACK segments across the entire network. In each pair, one host plays the role of a sender that transmits data at a rate of 50 Mb/s with an on-off behavior to the other destination host. The traffic flow has an exponential random distribution. Each data packet is 1500 bytes in size. All device channels have a capacity of 100 Mb/s with a delay of 0.001 ms. We also use the Nix-Vector routing protocol that was developed for simulating large-scale network topologies [6] as in the original setting of the source code. We modify the default initial slow-start threshold in ns-3 to 2 MB to allow all TCP protocols to enter their congestion avoidance phases shortly after the start of the simulations, during which they exhibit their distinct behaviors. The timestamp [24] and window scaling [24] options are both enabled. No loss model is used due to the reliable characteristic of DCN channels. Each simulation has a duration of 60 seconds. Table 3 summarizes our simulation parameters. We utilize the ns-3 Flow Monitor module [4] to collect the performance statistics with some of our own modifications to calculate the average throughput percentage of the ideal bandwidth and the average packet delay. While NewReno and HighSpeed are part of the standard release, Vegas, Scalable, Westwood+, and YeAH are our contributions to the ns-3 community [15, 31]. BIC and CUBIC implementations are obtained from the ns-3 SOCIS-2014 project [2].

3.2 Results and Analysis

Figure 2 displays the total number of packet drops occurring at all 45 switches of the network. The pure delay-based Vegas experiences no drops as it is designed to maintain a very small number of extra packets queued up at any given time during a connection. YeAH-TCP, with the incorporation of Vegas mechanism for monitoring network conditions, particularly when the network approaches congestion, also does not cause many queue drops. All of the loss-based congestion control algorithms, especially those designed to fully utilize the available bandwidth of high BDP links suffer badly from packet losses at the network queues, with CUBIC and STCP performing the worst due to their aggressiveness. The number of packet drops in Westwood+ is slightly smaller than HighSpeed and BIC, which can be attributed to its bandwidth estimation mechanism.

Parameter	Values
Device channel bandwidth	100 Mb/s
Device channel delay	0.001 ms
Packet MTU size	1500 B
Delayed ACK count	2 segments
Delayed ACK timeout	200 ms
Traffic flow pattern	exponential random
Simulation time	60 s
Routing protocol	Nix-Vector
Queue type	drop tail

Table 3: Simulation parameters

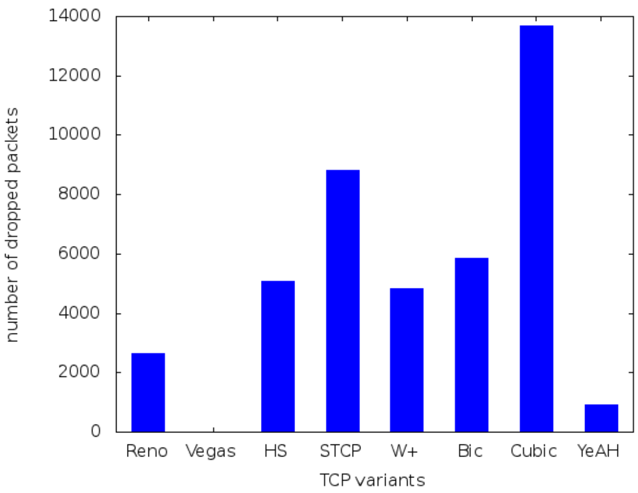
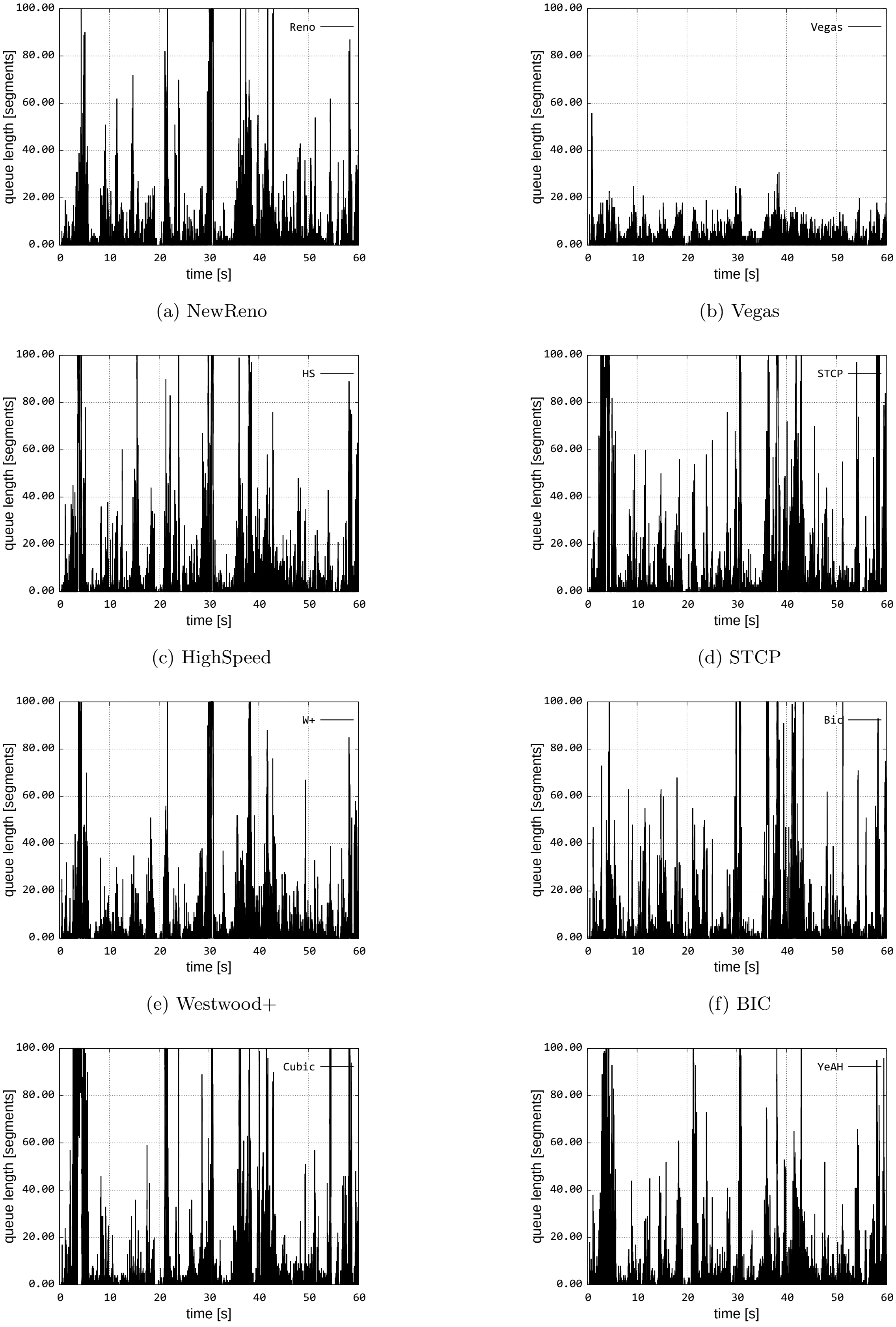


Figure 2: Total number of dropped packets

To illustrate the level of queue occupancy, we utilize the tracing system [7] in ns-3 to trace the queue length of one of the core switches during the connection lifetime. Figure 3 displays the collected results. While the other TCP variants tend to overflow the queue, Vegas maintains a small number of packets of less than 20 in its queue most of the time. CUBIC and STCP cause many queue-overflow instances with extended drop periods resulting in burst of packet losses. Multiple packet drops occurring in a single sending window trigger RTO (retransmission timeout), which resets the congestion window to its initial value. Our results illustrate the TCP incast impairment in DCNs when multiple mice flows transmit data simultaneously and experience collision at the upper-level core switches.

Table 4 presents the aggregate throughput achieved by each TCP variant as the percentage of the ideal channel bandwidth (100 Mb/s) and the average packet delay. The throughputs of all studied algorithms are far from optimal in our simulation scenario. Overall, the delay-based Vegas performs the best with higher throughput and smaller delay; a bit higher than 1/5 of BIC delay. As illustrated in the previous plots, when Vegas is used, the queue length is very small, enabling traffic flows to traverse through the network



(a) NewReno

(b) Vegas

(c) HighSpeed

(d) STCP

(e) Westwood+

(f) BIC

(g) CUBIC

(h) YeAH

Figure 3: Core switch queue length

	NewReno	Vegas	HSTCP	STCP	Westwood+	BIC	CUBIC	YeAH
% of ideal throughput	5.61	8.02	5.64	5.32	5.64	5.23	5.42	5.74
Average packet delay (ms)	14.17	3.36	13.01	16.07	13.01	16.04	14.54	13.58

Table 4: Percentage of ideal throughput and average delay

switches with low queueing delay, thus smaller end-to-end latency. The small delay of DCNs (0.001 ms in our simulation) also contributes to the poor performance of TCP variants that were intentionally designed to handle long-latency channels.

4. CONCLUSIONS

In our paper, we perform an evaluation of multiple conventional TCP congestion control algorithms, including the standard NewReno, Vegas, HighSpeed, STCP, Westwood+, BIC, CUBIC, and YeAH in data-center networks using a 6-ary fat tree topology in ns-3. We learn that while Vegas has been known for its inability to compete for network resource with other greedy, loss-based flows in other environments due to its proactive pure-delay based scheme, the same mechanism contributes to Vegas better performance when comparing with other loss-based variants (NewReno, HighSpeed, STCP, Westwood+, BIC, and CUBIC) and even with the hybrid scheme YeAH-TCP in DCNs. Our finding is consistent with the recent study that proposes the revival of delay-based TCP for data centers in which the authors show Vegas potentials when comparing it with DCTCP [26]. We agree that future development of TCP algorithms for DCNs should consider incorporating a delay-based mechanism similar to Vegas to reduce queue occupancy and overcome the queue buildup and buffer pressure problems so that all types of traffic flows can enjoy a shorter end-to-end delay. The aggressiveness of high-speed TCP variants trigger many packet drops at the queues while not helping them to achieve high throughput in this type of network, at least under our simulation setting presented in this paper.

As mentioned earlier, this paper is only our initial evaluation of TCP performance in DCNs. To gain more insights into the congestion control algorithms, for our future work, we plan to extend our simulation model, not only in the TCP variant dimension, but also in the architecture (Dcell [18], BCube [17], and VL2 [16]), application (MapReduce [13]), and performance metrics. In addition, we are aware of the need to evaluate the algorithms in an environment with mixed traffic types (mice, cat, and elephant) since application diversity is a natural characteristic of DCNs.

5. ACKNOWLEDGMENTS

The authors would like to thank the members of the ResiliNets group for discussions that led to this work. We would like to thank the anonymous reviewers for their helpful feedback on this paper. We would also like to thank Natale Patriciello and the ns-3 development team for their help with the ns-3 platform and for the BIC and CUBIC implementations. This work was funded in part by NSF grant CNS-1219028 (Resilient Network Design for Massive Failures and Attacks).

6. REFERENCES

[1] The ns-3 Network Simulator. http://www.nsnam.org, July 2009.

[2] ns-3 SOCIS 2014: TCP versions for satellite communications. https://www.nsnam.org/wiki/SOCIS2014TCP, 2014.

[3] An open-source ns-3 simulation framework for data center network architectures. http://ntu-dsi-dcn.github.io/ntu-dsi-dcn/, 2015.

[4] ns-3 Flow Monitor Module Documentation. https://www.nsnam.org/docs/models/html/flow-monitor.html, 2015.

[5] ns-3 Implementation of Fat-tree Architecture. https://github.com/ntu-dsi-dcn/ntu-dsi-dcn, 2015.

[6] ns-3 Nix-Vector Routing API, Usage, and Implementation. https://www.nsnam.org/docs/release/3.16/doxygen/group__nixvectorrouting.html, 2015.

[7] ns-3 Tracing System Manual. https://www.nsnam.org/docs/manual/html/tracing.html, 2015.

[8] M. Al-Fares, A. Loukissas, and A. Vahdat. A Scalable, Commodity Data Center Network Architecture. *SIGCOMM Comput. Commun. Rev.*, 38(4):63–74, Aug. 2008.

[9] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data Center TCP (DCTCP). In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, pages 63–74, New York, NY, USA, 2010. ACM.

[10] M. Allman, V. Paxson, and E. Blanton. TCP Congestion Control. RFC 5681 (Draft Standard), Sept. 2009.

[11] A. Baiocchi, A. P. Castellani, and F. Vacirca. YeAH-TCP: yet another highspeed TCP. In *Proc. PFLDnet*, volume 7, pages 37–42, 2007.

[12] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson. TCP Vegas: new techniques for congestion detection and avoidance. *SIGCOMM Comput. Commun. Rev.*, 24(4):24–35, 1994.

[13] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM*, 51(1):107–113, Jan. 2008.

[14] S. Floyd. HighSpeed TCP for Large Congestion Windows. RFC 3649 (Experimental), Dec. 2003.

[15] S. Gangadhar, T. A. N. Nguyen, G. Umapathi, and J. P. Sterbenz. TCP Westwood Protocol Implementation in ns-3. In *Proceedings of the ICST SIMUTools Workshop on ns-3 (WNS3)*, Cannes, France, March 2013.

[16] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. VL2: A Scalable and Flexible Data Center Network. In *Proceedings of the ACM*

SIGCOMM Conference on Data Communication, pages 51–62, Barcelona, Spain, 2009.

[17] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu. Bcube: A high performance, server-centric network architecture for modular data centers. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, SIGCOMM '09, pages 63–74, New York, NY, USA, 2009. ACM.

[18] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu. Dcell: A Scalable and Fault-tolerant Network Structure for Data Centers. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, SIGCOMM '08, pages 75–86, New York, NY, USA, 2008. ACM.

[19] S. Ha, Y. Kim, L. Le, I. Rhee, and L. Xu. A step toward realistic performance evaluation of high-speed tcp variants. In *Fourth International Workshop on Protocols for Fast Long-Distance Networks (PFLDNet06)*, 2006.

[20] S. Ha, I. Rhee, and L. Xu. CUBIC: A New TCP-friendly High-speed TCP Variant. *SIGOPS Oper. Syst. Rev.*, 42(5):64–74, July 2008.

[21] T. Henderson, S. Floyd, A. Gurtov, and Y. Nishida. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 6582 (Standards Track), 2012.

[22] S. Henna. A Throughput Analysis of TCP Variants in Mobile Wireless Networks. In *Next Generation Mobile Applications, Services and Technologies, 2009. NGMAST '09. Third International Conference on*, pages 279–284, Sept 2009.

[23] J. Hwang, J. Yoo, and N. Choi. IA-TCP: A rate based incast-avoidance algorithm for TCP in data center networks. In *Communications (ICC), 2012 IEEE International Conference on*, pages 1292–1296, June 2012.

[24] V. Jacobson, R. Braden, and D. Borman. TCP Extensions for High Performance. RFC 1323 (Proposed Standard), May 1992.

[25] T. Kelly. Scalable TCP: Improving performance in highspeed wide area networks. *ACM SIGCOMM Computer Communication Review (CCR)*, 33(2):83, Apr. 2003.

[26] C. Lee, K. Jang, and S. Moon. Reviving Delay-based TCP for Data Centers. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '12, pages 111–112, New York, NY, USA, 2012. ACM.

[27] S. Mascolo, C. Casetti, M. Gerla, M. Sanadidi, and R. Wang. TCP westwood: Bandwidth estimation for enhanced transport over wireless links. In *Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 287–297. ACM, 2001.

[28] S. Mascolo, L. Grieco, R. Ferorelli, P. Camarda, and G. Piscitelli. Performance evaluation of Westwood+ TCP congestion control. *Performance Evaluation*, 55(1-2):93–111, Jan. 2004.

[29] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgment Options. RFC 2018 (Proposed Standard), Oct. 1996.

[30] J. C. Mogul. Observing TCP dynamics in real networks. *SIGCOMM Comput. Commun. Rev.*, 22(4):305–317, Oct. 1992.

[31] T. A. N. Nguyen, S. Gangadhar, M. M. Rahman, and J. P. Sterbenz. An Implementation of Scalable, Vegas, Veno, and YeAH Congestion Control Algorithms in ns-3 (extended). ITTC Technical Report ITTC-FY2016-TR-69921-04, The University of Kansas, Lawrence, KS, April 2016.

[32] R. P. Tahiliani, M. P. Tahiliani, and K. C. Sekaran. TCP Variants for Data Center Networks: A Comparative Study. In *Cloud and Services Computing (ISCOS), 2012 International Symposium on*, pages 57–62, Dec 2012.

[33] B. Vamanan, J. Hasan, and T. Vijaykumar. Deadline-aware Datacenter TCP (D2TCP). *SIGCOMM Comput. Commun. Rev.*, 42(4):115–126, Aug. 2012.

[34] E. Weingartner, H. vom Lehn, and K. Wehrle. A Performance Comparison of Recent Network Simulators. In *Communications, 2009. ICC '09. IEEE International Conference on*, pages 1–5, June 2009.

[35] D. Wong, K. T. Seow, C. H. Foh, and R. Kanagavelu. Towards reproducible performance studies of datacenter network architectures using an open-source simulation approach. In *Global Communications Conference (GLOBECOM), 2013 IEEE*, pages 1373–1378, Dec 2013.

[36] H. Wu, Z. Feng, C. Guo, and Y. Zhang. ICTCP: Incast Congestion Control for TCP in Data Center Networks. In *Proceedings of the 6th International COnference*, Co-NEXT '10, pages 13:1–13:12, New York, NY, USA, 2010. ACM.

[37] L. Xu, K. Harfoush, and I. Rhee. Binary increase congestion control (BIC) for fast long-distance networks. In *INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 4, pages 2514–2524, 2004.