TSVWG
Internet-Draft
Intended status: Informational
Expires: May 6, 2020

Y. Zhuang
W. Sun
L. Yan
Huawei Technologies Co., Ltd.

November 3, 2019

An Open Congestion Control Architecture for high performance fabrics
draft-zhuang-tsvwg-open-cc-architecture-00

Abstract

This document describes an open congestion control architecture of high performance fabrics for the cloud operators and algorithm developers to deploy or develop new congestion control algorithms as well as make appropriate configurations for traffics on smart NICs in a more efficient and flexible way.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 6, 2020.

Copyright Notice

Table of Contents

1. Introduction

The datacenter networks (DCNs) nowadays is not only providing traffic transmission for tenants using TCP/IP network protocol stack, but also is required to provide RDMA traffic for High Performance Computing (HPC) and distributed storage accessing applications which requires low latency and high throughput.

Thus, for datacenter application nowadays, the requirements of latency and throughput are more critical than the normal internet traffics, while network congestion and queuing caused by incast is the point that increases the traffic latency and affect the network throughput. With this, congestion control algorithms aimed for low latency and high bandwidth are proposed such as DCTCP[RFC8257], [BBR] for TCP, [DCQCN] for [RoCEv2].

Besides, the CPU utilization on NICs is another point to improve the efficiency of traffic transmission for low latency applications. By offloading some protocol processing into smart NICs and bypassing CPU, applications can directly write to hardware which reduces the latency of traffic transmission. RDMA and RoCEv2 is currently a good example to show the benefit of bypassing kernel/CPU while TCP offloading is also under discussion in [NVMe-oF].

In general, one hand, the cloud operators or application developers are working on new congestion control algorithms to fit requirements of applications like HPC, AI, storage in high performance fabrics; while on the other hand, smart NIC vendors are working on offloading functions of data plane and control plane onto hardware so as to reduce the process latency and improve the performance. In this case, it comes up with the question that how smart NICs can be optimized by offloading some functions onto the hardware while still being able to provide flexibility to customers to develop or change their congestion control algorithms and run their experiments more easily.

That said, it might be good to have an open and modular-based design for congestion control on smart NICs to be able to develop and deploy new algorithms while take the advantage of hardware offloading in a generic way.

This document is to describes an open congestion control architecture of high performance fabrics on smart NICs for the cloud operators and application developers to install or develop new congestion control algorithms as well as select appropriate controls in a more efficient and flexible way.

It only focus on the basic functionality and discuss some common interfaces to network environments and also administrators and application developers while the detailed implementations should be vendors' specific designs and are out of scope.

Discussions of new congestion control algorithms and improved active queue management (AQM) are also out of scope for this document.

2. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Abbreviations

IB - InfinitBand

HPC - High Performance Computing

ECN - Explicit Congestion Notification

AI/HPC - Artificial Intelligence/High-Performance computing

RDMA - Remote Direct Memory Access

NIC - Network Interface Card

AQM - Active Queue Management

4. Observations in storage network

Besides the benefits of easing the development of new congestion control algorithms by developers while taking advantage of hardware offloading improvement by NIC vendors, we notice that there are also benefits to choose proper algorithms for specific traffic patterns.

As stated, there are several congestion control algorithms for low latency high throughput datacenter applications and the industry is still working on enhanced algorithms for requirements of new applications in the high performance area. Then, a question might be asked, how to select a proper congestion algorithm for the network, or whether a selected algorithm is efficient and sufficient to all traffics in the network.

With this question, we use a simplified storage network as a use case for study. In this typical network, it mainly includes two traffic types: query and backup. Query is latency sensitive traffic while backup is high throughput traffic. We select several well-known congestion control algorithms (including Reno[RFC5681], Cubic[RFC8312], DCTCP[RFC8257], and BBR[BBR]) of TCP for this study.

Two set of experiments were run to see the performance of these algorithms for different traffic types (i.e. traffic patterns). The first set is to study the performance when one algorithm is used for both traffic types; the second set is to run the two traffics with combinations of congestion algorithms. The detailed experiments and testing results can be found in appendix A.

According to the result in first experiment set, BBR performs better than others when applied for both traffics; while in the second experiment set, some algorithm combinations show better performance than the same one for both, even compared with BBR.

As such, we think there are benefits for different traffic patterns to select their own algorithm in the same network to achieve better performance. This can also be a reason from cloud operation perspective to have an open congestion control on the NIC to select proper algorithms for different traffic patterns.

5. Requirements of the open congestion control architecture

According to the observation, the architecture design is suggested to follow some principles:

- Can support developers to write their congestion control algorithms onto NICs while keep the benefit of congestion control offloading provided by NIC vendors.
- Can support vendors to optimize the NIC performance by hardware offloading while allow users to deploy and select new congestion control algorithms.
- Can support settings of congestion controls by administrators according to traffic patterns.
- Can support settings from applications to provide some QoS requirements.
- Be transport protocol independent, for example can support both TCP and RoCE.

6. Open Congestion Control (OpenCC) Architecture Overview

The architecture shown in Figure 1 only states the congestion control related components while components for other functions are omitted. The OpenCC architecture includes three layers.

The bottom layer is called the congestion control engine which provides common function blocks independent of transport protocols which can be implemented in hardware, while the middle layer is the congestion control platform in which different congestion control algorithms will be deployed here. These algorithms can be installed by NIC vendors or can be developed by algorithm developers. At last, the top layer provides all interfaces (i.e. APIs) to users, while the users can be administrators that can select proper algorithms and set proper parameters for their networks, applications that can indicate their QoS requirements which can be further mapped to some runtime settings of congestion control parameters, and the algorithm developers that can write their own algorithms.

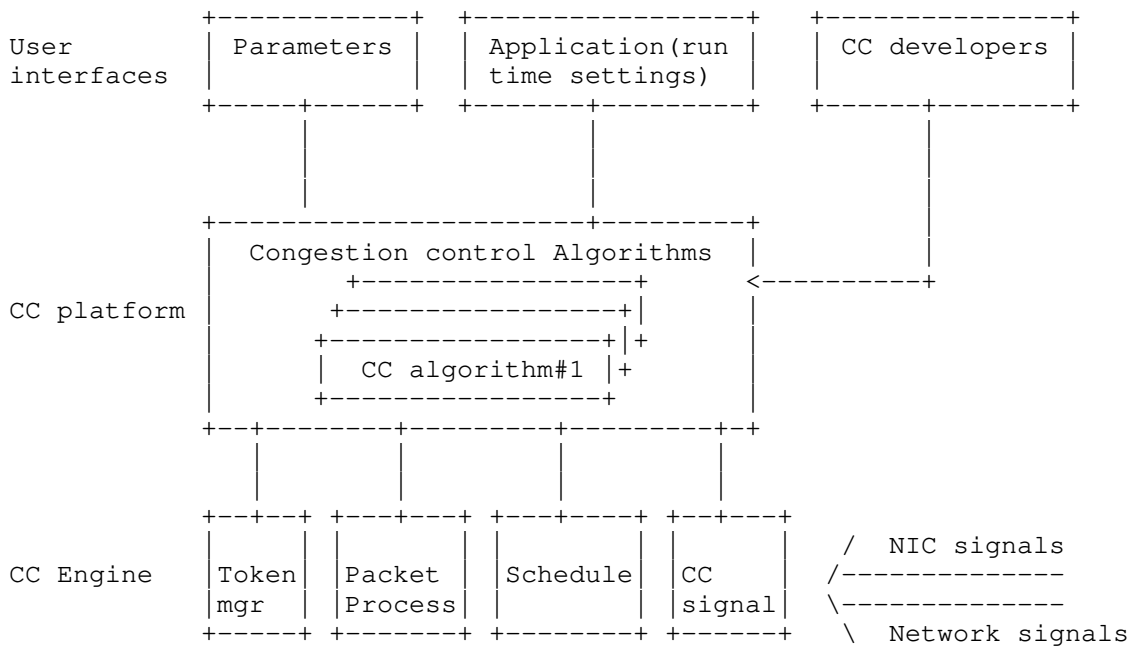

Figure 1. The architecture of open congestion control

6.1. Congestion Control Platform and its user interfaces

The congestion control platform is a software environment to deploy and configure various congestion control algorithms. It contains three types of interfaces to the user layer for different usage.

One is for administrators, which is to select proper congestion control algorithms for their network traffics and configure corresponding parameters of the selected algorithms.

The second one can be an interface defined by NIC vendors or developers that provide some APIs for application developers to define their QoS requirements which will be further mapped to some runtime configuration of the controls.

The last one is for algorithm developers to write their own algorithm in the system. It is suggested to have a defined common language to write algorithms which can be further compiled by vendor specific environments (in which some toolkits or library can be provided) to generate the platform dependent codes.

6.2. Congestion Control Engine (CCE) and its interfaces

Components in the congestion control engine can be offloaded to the hardware to improve the performance. As such, it is suggested to provide some common and basic functions while the upper platform can provide more extensibility and more flexibility for more functions.

The CCE includes basic modules of packet transmission and corresponding control. Several function blocks are illustrated here while the detailed implementation is out of scope for this document and left for NIC vendors. A token manager is used to distribute tokens to traffics while the schedule block is to schedule the transmission time for these traffics. The packet process block is to edit or process the packet before transmission. The congestion control signal block is to collect or monitor signals from both network and other NICs which will be fed to congestion control algorithms.

As such, an interface to get congestion control signal in the congestion control should be defined to receive signals from both other NICs and networks for existing congestion control algorithms and new extensions. These information will be used as inputs of control algorithms to adjust the sending rate and operate the loss recovery et.al.

7. Interoperability Consideration

7.1. Negotiate the congestion control algorithm

Since there will be several congestion control algorithms, the host might negotiate their supported congestion control capability during the session setup phase. However, it should use the existing way of congestion control as default to provide compatibility with legacy devices.

Also, the network devices on the path should be capable to indicate their capability of any specific signals that the congestion control algorithm needs. The capability negotiation between NICs and Switches can be considered either some in-band ECN-like negotiations or out-of-band individual message negotiations.

Alternatively, the system can also use a centralized administration platform to configure the algorithms on NICs and network devices.

7.2. Negotiate the congestion control parameters

The parameters might be set by administrators to meet their traffic patterns and network environments or be set by mappings from application requirements. Hence, these parameters might be changed after the session is set up. As such, hosts should be able to negotiate their parameters when changed or be configured to keep consistent.

8. Security Considerations

TBD

9. Manageability Consideration

TBD

10. IANA Considerations

No IANA action

11. References

11.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

11.2. Informative References

[BBR] Cardwell, N., Cheng, Y., and S. Yeganeh, "BBR Congestion Control", <https://tools.ietf.org/html/draft-cardwell-iccrg-bbr-congestion-control-00>.

[DCQCN] "Congestion Control for Large-Scale RDMA Deployments.", <https://conferences.sigcomm.org/sigcomm/2015/pdf/papers/p523.pdf>.

[NVMe-oF] "NVMe over Fabrics", <https://nvmexpress.org/wp-content/uploads/NVMe_Over_Fabrics.pdf>.

[RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <https://www.rfc-editor.org/info/rfc5681>.

[RFC8257] Bensley, S., Thaler, D., Balasubramanian, P., Eggert, L., and G. Judd, "Data Center TCP (DCTCP): TCP Congestion Control for Data Centers", RFC 8257, DOI 10.17487/RFC8257, October 2017, <https://www.rfc-editor.org/info/rfc8257>.

[RFC8312] Rhee, I., Xu, L., Ha, S., Zimmermann, A., Eggert, L., and R. Scheffenegger, "CUBIC for Fast Long-Distance Networks", RFC 8312, DOI 10.17487/RFC8312, February 2018, <https://www.rfc-editor.org/info/rfc8312>.

[RoCEv2] "Infiniband Trade Association. InfiniBandTM Architecture Specification Volume 1 and Volume 2.", <https://cw.infinibandta.org/document/dl/7781>.

Appendix A. Experiments

This section includes two sets of experiments to study the performance of congestion control algorithms in a simplified storage network. The first set is to study one algorithm applied for both query and backup traffics while the second set is to study the performance when different algorithms are used for query traffic and backup traffic. The metrics include throughput of backup traffic, average completion time of query traffic and 95% percentile query completion time.

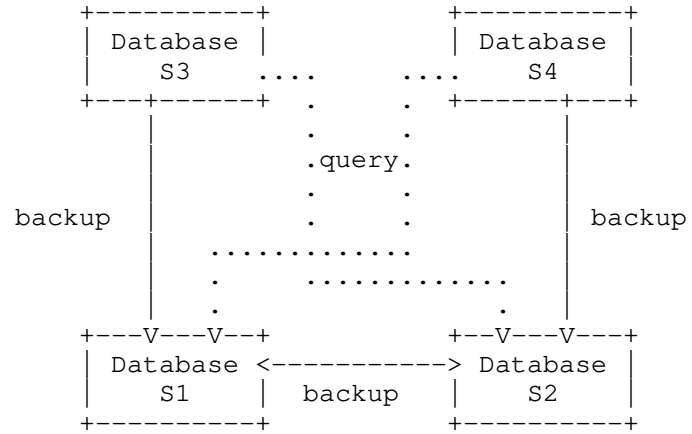

Figure 2. Simplified storage network topology

All experiments are a full implementation of congestion control algorithms on NICs, including Reno, Cubic, DCTCP and BBR. Our experiments includes 4 servers connecting to one switch. Each server with a 10Gbps NIC connected to a 10Gbps port on the switch. However, we limit all ports to 1Gbps to make congestion points. In the experiments, the database server S1 receives backup traffics from both S3 and S2 and one query traffic from S4. The server S2 gets back traffics from S1 and S4 and one query traffic from S3.In the experiments, three traffic flows are transmitted to S1 from one egress port on the switch, which might cause congestion.

In the first experiment set, we test one algorithm for both traffics. The result is shown below in table 1.

	reno	cubic	bbr	dctcp
Throughput MB/s	64.92	65.97	75.25	70.06
Avg. comp ms	821.61	858.05	85.68	99.90
95% comp ms	894.65	911.23	231.75	273.92

Table 1. Performance when use one cc for both query and backup traffics

As we can see, the average completion time of BBR and DCTCP is 10 times better than that of reno and cubic. BBR is the best to keep high throughput.

In the second set, we test all the combinations of algorithms for the two traffics.

1. Reno for query traffic

reno@query

@backup	cubic	bbr	dctcp	reno
Throughput MB/s	66.00	76.19	64.00	64.92
Avg. comp ms	859.61	81.87	18.38	821.61
95% comp ms	917.80	149.88	20.38	894.65

Table 2. reno @ query and cubic, bbr, dctcp @ backup

It shows that given reno used for query traffic, bbr for backup traffic gets better throughput compared with other candidates. However, dctcp for backup traffic gets much better average completion time and 95% completion time, almost 6 times better than those of bbr even its throughput is less than bbr. The reason for this might be bbr does not consider lost packets and congestion levels which might cause much retransmission. In this test set, dctcp for backup traffic gets better performance.

2. Cubic for query traffic

cubic@query

@backup	reno	bbr	dctcp	cubic
Throughput MB/s	64.92	75.02	65.29	65.97
Avg. comp ms	819.23	83.50	18.42	858.05
95% comp ms	902.66	170.96	20.99	911.23

Table 3. cubic @ query and reno, bbr, dctcp @ backup

The results of cubic for query traffic are similar to those of reno. Even with less throughput, dctcp has almost 6 times better than bbr in average completion time and 95% completion time, and nearly 10 times better than those of reno and cubic.

3. Bbr for query traffic

bbr@query

@backup	reno	cubic	dctcp	bbr
Throughput MB/s	64.28	66.61	65.29	75.25
Avg. comp ms	866.05	895.12	18.49	85.68
95% comp ms	925.06	967.67	20.86	231.75

Table 4. bbr @ query and reno, cubi, dctcp @ backup

The results still match those we get from reno and cubic. In the last two columns, dctcp for backup shows better performance even when we compared with bbr used for backup. It indicates that bbr @ query and dctcp @ backup is better than bbr @ query and backup.

4. Dctcp for query traffic

dctcp@query

@backup	reno	cubic	bbr	dctcp
Throughput MB/s	60.93	64.49	76.15	70.06
Avg. comp ms	2817,53	3077.20	816.45	99.90
95% comp ms	3448.53	3639.94	2362.72	273.92

Table 5. dctcp @ query and reno, cubi, bbr @ backup

The results for dctcp@query look worse than others in completion time, since we don't introduce L4S in the experiments which means dctcp will back off most of the time when congestion happens which makes the query traffic bares long latency. The best performance in this test set happens at dctcp@backup. In this setting, both traffics have use the same mechanism to back off their traffics. However, the number is still worse than when other algorithms are used for query and dctcp used for backup.

Authors' Addresses

Yan Zhuang
Huawei Technologies Co., Ltd.
101 Software Avenue, Yuhua District
Nanjing, Jiangsu 210012
China

Email: zhuangyan.zhuang@huawei.com

Wenhao Sun
Huawei Technologies Co., Ltd.
101 Software Avenue, Yuhua District
Nanjing, Jiangsu 210012
China

Email: sam.sunwenhao@huawei.com

Long Yan
Huawei Technologies Co., Ltd.
101 Software Avenue, Yuhua District
Nanjing, Jiangsu 210012
China

Email: yanlong20@huawei.com