Transport Area Working Group H. Dai, Ed.
Internet-Draft B. Fu
Intended status: Informational K. Tan
Expires: 13 October 2021 Huawei
11 April 2021

PFC-Free Low Delay Control Protocol
draft-dai-tsvwg-pfc-free-congestion-control-01

Abstract

Today, low-latency transport protocols like RDMA over Converged Ethernet (RoCE) can provide good delay and throughput performance in small and lightly loaded high-speed datacenter networks due to lossless transport based on priority-based flow control (PFC). However, PFC suffers from various issues from performance degradation and unreliability (e.g., deadlock), limiting the deployment of RoCE to only small scale clusters (~1000).

This document presents LDCP, a new transport that scales loss-sensitive transports, e.g., RDMA, to entire data-centers containing tens of thousands machines, without dependency on PFC for losslessness, i.e., PFC-free. LDCP develops a novel end-to-end congestion control scheme and achieves very low queue occupancy even under high network utilization or large traffic churns, resulting in almost no packet loss. Meanwhile, LDCP allows a new flow to jump start at full speed at the very beginning and therefore minimizes the latency for short RPC-style transactions. LDCP relies on only WRED and ECN, two widely supported features on switches, so it can be easily deployed in existing network infrastructures. Finally, LDCP is simple by design and thus can be easily implemented by programmable or ASIC NICs.

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 13 October 2021.

Table of Contents

1. Introduction 2
1.1. Requirements Language 3
2. LDCP algorithm 3
2.1. ECN 4
2.2. Stable stage algorithm 4
2.3. Zero-RTT bandwidth acquisition 6
3. Reference Implementations 8
3.1. Implementation on programmable NIC 8
3.2. Implementation on commercial NIC 9
4. IANA Considerations 10
5. Security Considerations 10
6. References 10
6.1. Normative References 10
6.2. Informative References 11
Authors' Addresses 11

1. Introduction

Modern cloud applications, such as web search, social networking, real-time communication, and retail recommendation, require high throughput and low latency network to meet the increasing demands from customers. Meanwhile, new trends in data-centers, like resource disaggregation, heterogeneous computing, block storage over NVMe, etc., continuously drive the need for high-speed networks. Recently, high-speed networks, with 40Gbps to 100Gbps link speed, are deployed in many large data-centers.

Conventional software TCP/IP stacks incur high latencies and substantial CPU overhead, and have limited applications from fully utilizing the physical network capacities. RDMA over Converged

Ethernet (RoCE), however, has shown very good low-delay and throughput performance in small and lightly loaded networks, due to the ability of OS bypassing and a lossless transport that performs hop-by-hop flow control, i.e., PFC. Nevertheless, in a large data-center network (with tens of thousands of machines) with bursty traffic, PFC backpressure leads to cascaded queue buildups and collateral damages to victim flows, resulting in neither Low latency nor high throughput [Guo2016rdma]. Therefore, high-speed networks still face fundamental challenges to deliver the three aforementioned goals.

This document describes LDCP, a scalable end-to-end congestion control that achieves low latency even under high network utilization. The key insight behind LDCP is using ACKs to grant to or revoke from senders credits, in order to mimic receiver-driven pulling. LDCP requires data receivers to reply ACKs as fast as possible, preferably one ACK for each data packet received (per-packet ACK). The congestion window is adjusted on the per-ACK basis using a parameterized AIMD algorithm. This algorithm manages to smooth out the traffic burstiness and stabilize the queue size at ultra-low level, preventing queue buildups and preserving high link utilization. A first-RTT bandwidth acquisition algorithm is also proposed to allow new flows to start sending at a large rate, but excessive packets will be actively dropped by WRED if they overwhelm the network, in order to protect on-going flows. When heavy congestion happens due to a large number of concurrent flows contending for the bottleneck link, e.g., large-scale incast, LDCP allows the congestion window to be beneath one packet, so the number of flows that LDCP can endure remarkably increases compared with TCP or DCTCP.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. LDCP algorithm

LDCP involves primarily two algorithms: a fast start algorithm that is used in the first RTT, and a stable stage algorithm that governs the rest lifespan of a flow. Each algorithm works with a separate ECN setting respectively. Because we want to use as fewer priority classes as possible, we leverage the common WRED/ECN [CiscoGuide2012] [RFC3168] feature in commodity switches to support multiple ECN marking policies within one priority class.

2.1. ECN

LDCP employs WRED/ECN at intermediate switches to mark packets when congestion happens [Floyd1993random]. Instead of using the average queue size for marking as in the original RED proposal, LDCP employs instant queue based ECN to give more precise congestion information to end hosts [Alizadeh2010data] [Kuzmanovic2005power]. The switch is configured with four parameters: K_min, K_max, P_max and buf_max, and it is going to mark a packet with a probability function as follows,

if q < K_min, p = 0

if K_min <= q < K_max, p = (q-K_min)/(K_max-K_min)*P_max

if q >= K_max, p = 1

If q is larger than the maximum buffer of the port (buf_max), the packet is dropped. This general ECN model works for both algorithms developed in LDCP but with different sets of parameters, respectively. We will explain below.

2.2. Stable stage algorithm

In stable stage, i.e., rounds after the fast start (sec 2.3), the flow is in the congestion avoidance state, and LDCP works as follows.

The sender maintains a congestion window (cw) to control the sending rate of data packets. The receiver returns ACK packets to confirm the delivery of these data packets. Meanwhile, the CE (Congestion Experienced) flag in data packets are echoed back by ECN-Echo (ECE) flags in the ACKs. An ACK that does not carry an ECE flag (ECE=0) informs the sender that the network is not congested, while an ACK that carries an ECE flag (ECE=1) informs the sender that the network is congested.

There could be two possible ways regarding the number of ACKs generated. The simplest one is to have the receiver to generate an ACK for every received data packet (i.e., per-packet ACK) and set the ECE flag if the corresponding packet has a CE mark. Alternatively, if the receiver is busy, it can also employ delayed ACK to generate an ACK for at most m data packets if they all are not marked, but would generate an ACK with ECE flag immediately once a CE marked packet is received. The goal of this receiver behavior is to ensure that the sender has precise information of CE marking. A similar design is also in [Alizadeh2010data].

An LDCP sender updates the cw upon each ACK arrival according to the ECE marks, namely per-ACK window adjustment (PAWA). An ECE=0 flag increases the cw, while an ECE=1 flag decreases the cw. When per-packet ACK is obeyed on the receiver, the update rule is as follows:

if ECN-Echo = 0, cw = cw + alpha/cw

if ECN-Echo = 1, cw = cw - beta --(1)

where alpha and beta are constants, and cw >= 1 (0 < alpha, beta <= 1).

Eq. (1) reveals that if an incoming ACK does not carry an ECE flag (ECE=0), it grants the sender credits, and cw is increased by alpha/cw; if the ACK carries an ECE flag (ECE=1), it revokes credits from the sender, and cw is decreased by beta.

In essence, Eq. (1) implements an additive increase and multiplicative decrease (AIMD) policy similar to previous work, e.g., DCTCP [Alizadeh2010data]. But PAWA, together with per-packet ACK, has following benefits: Firstly, PAWA reacts to each received ECE mark (or no mark) immediately, rather than employs a RTT-granularity averaging process and reacts only once per RTT (like DCTCP does), so it is more responsive and accurate to congestions. Secondly, along with WRED/ECN, PAWA is able to de-synchronize flows. Instead of cutting a large portion of cw immediately upon the first ECE-marked ACK (like ECN-enabled TCP does), LDCP distributes the window reduction in one round. Such de-synchronization is effective to reduce the window fluctuation and stabilize a low queue at the switches. Not only that, per-packet ACK allows ACK-clocking to better pace out the packets: as each ACK confirms the delivery of one packet, an ACK arrival also clocks out one new packet, hence the packets are almost equispaced. Finally, PAWA has a tiny state footprint, i.e., a single state of cw, and is easy to implement in hardware compared with DCTCP.

Per-packet ACK and PAWA match the principle in discrete control systems: increase the controller’s action rate but take a small control step per action. They are effective in improving the control stability and accuracy.

If delayed ACK is used on the receiver side, an ACK can confirm the delivery of multiple (denoted by n) packets, then Eq. (1) becomes:

if ECN-Echo = 0, cw = cw + n * alpha/cw

if ECN-Echo = 1, cw = cw - n * beta --(2)

In extremely congested cases where a large number of flows contending for the bottleneck link, e.g., heavy incast with thousands of senders, even each flow maintains a window of merely one packet, large queue sizes would still be caused. To handle these situations, LDCP allows cw to further reduce beneath one packet. A flow with a cw < 1 is ticked out by a timer, whose timeout is set as RTT/cw. Accordingly, the cw updating rule is,

if ECN-Echo = 0, cw = cw + gamma

if ECN-Echo = 1, cw = max{gamma,eta * cw}

where cw < 1. We choose eta = 1/2. gamma is the increase step when ACK is not marked ECE, and is also the minimum window size (typical values of gamma include 1/4, 1/8, 1/16).

2.3. Zero-RTT bandwidth acquisition

Setting up an initial rate at the very beginning of a flow is challenging. Since the sender does not ever get a chance to probe the network, it faces a difficult dilemma: If it picks up a too large initial window (IW), it may cause congestion inside network, resulting in large queue buildup or even packet drops; On the other hand, if the sender chooses a too conservative IW, it may lose the transmission opportunities in the first RTT and hurt short flow performance greatly, which could have finished in one round. LDCP resolves this dilemma with a zero-RTT bandwidth acquisition algorithm, which allows the sender to fast start opportunistically without adverse impacts to on-going flows in stable stage. In what follows, the design of the fast start algorithm is firstly described, afterwards an implementation using existing techniques is provided.

Specifically, when a flow starts, the sender chooses a large enough Initial Window (e.g., BDP) and sends out as many packets as possible in the first RTT. (For brevity, packets transmitted by a sender in the first RTT are denoted by first-RTT-packets, and packets transmitted in the congestion avoidance state (sec 2.2) are referred to as stable-stage-packets.) By intention, first-RTT-packets are marked to have lower priority, while stable-stage-packets are marked to have high priority. The two priority classes are controlled by two separate AQM policies.

The first-RTT-packets are controlled by an AQM policy which simply drops packets if they are sent too aggressively, i.e., the queue exceeds a configured threshold K. A network switch receives packets transmitted by the senders and puts them into a queue. The queue distinguishes the first-RTT-packets and the stable-stage-packets according to the marks in the packets. Because first-RTT-packets are

with low priority, they will be dropped if the receiving queue size exceeds the configured threshold, while stable-stage-packets are enqueued as long as the queue size is below the queue capacity. Stable-stage-packets are dropped only when the queue is full.

Senders and switches must cooperate. The sender adds one mark to first-RTT-packets, and the switches identify first-RTT-packets using this mark; the sender adds another mark to stable-stage-packets, and the switches recognize packets sent beyond first RTT based on this mark.

In summary, first-RTT-packets are sent with a large rate, and controlled by a separate AQM, to quickly acquire free bandwidth if there is; and low priority is used to protect on-going long flows if there is not.

The above design can be implemented by leveraging a common feature on modern switches. On a commodity switch, the WERD/ECN feature on an ECN-enabled queue works as follows. ECN-capable packets (the two-bit ECN fields in IP headers are set to '01' or '10') are subject to ECN-marking, while ECN-incapable packets (the two-bit ECN fields in IP headers are set to '00') comply with WRED-dropping, i.e., ECN-incapable packets are dropped if the queue size exceeds a configured threshold K, as in Eq (3).

if q < K, D(q) = no drop

if q >= K, D(q) = drop --(3)

The fast start algorithm makes use of such WERD/ECN feature to distinguish first-RTT and stable-stage packets: the sender sets the low priority first-RTT-packets to ECN-incapable, and sets the high priority stable-stage-packets to ECN-capable. All the packets carry the same DSCP value and are mapped to the same priority queue on switches. This queue is exclusively used by LDCP flows. First-RTT-packets are either dropped or successfully pass the switch. After the first RTT, the sender will count how many in-order packets has been acknowledged using ACKs and takes this as a good estimation of cw and enters the stable stage (sec 2.2).

At first glance, the above design might look counterintuitive. If we want to improve the performance of short flows, why should we drop their packets, instead of queuing them, even with a higher priority? The answer, however, lies in that if we allow blind burst in the first RTT, these first-RTT-packets could build excessively large queues, e.g., in a heavy incast scenario, and eventually these packets may still get dropped. Therefore, an AQM policy is necessary to keep a low queue for the first RTT packets. An additional benefit

of the above strategy is to also give protection to flows in stable stage. Those stable stage flows will experience seldom packet loss and constant performance even facing rather dynamic churns of short flows. Finally, we comment that while we could put the first-RTT-traffic into a separate high priority queue, we believe it is not very necessary. The reason is with LDCP's stable stage algorithm, the queue is already small at the switch, and thus the benefit for a separate priority queue may be limited. Given the limited priority queues in Ethernet, it is a fair choice to map both into one priority queue while applying different WRED/ECN polices to control their behavior.

3. Reference Implementations

3.1. Implementation on programmable NIC

LDCP has been implemented with RoCEv2 on a programmable many-core NIC (referred to as uNIC). uNIC has hardware enhancements for RoCEv2 packet (IB/UDP/IP stack) encapsulation and decapsulation. The RoCEv2 stack, as well as the congestion control algorithm, is implemented by microcode software on uNIC.

Congestion window cw is firstly added to RoCEv2 to limit the in-flight data size. RoCEv2 uses Packet Sequence Number (PSN) to ensure in-order delivery, but PSN can have jump overs if SEND/WRITE requests are interleaved with READ requests, and packets can have different sizes. Therefore, it is difficult for cw to calculate the data size by PSN. A new byte sequence number -- LDCP Sequence Number (LSN) -- is used to slide the window. Packets belonging to READ, SEND/WRITE requests share the same LSN space, while packets of READ Responses have a separate LSN space, coded in a customized header.

In the stable stage of LDCP, cw is updated in the PAWA manner, and the uNIC is programmed to reply an ACK for each data packet it receives (uNIC is able to automatically coalesce ACKs based on its current load), which echoes back the CE mark if the data packet is marked. Note that there is no ACK packets for Read Response in the RDMA protocol, the uNIC is also programmed to reply ACKs for Read Responses to enable congestion control. Because out-of-order delivery of Read Responses can be discovered by the requester, and a repeat read request will be issued, it is not necessary to add a NAK protocol for Read Responses to ensure reliability. The CE-Echo bits are coded in a customized header encapsulated in the ACK.

On switches, fast-start packet needs WRED-dropping while stable-stage packets need ECN-marking, so the packets should carry different flags to be identified by the switches. The WERD/ECN feature on an ECN-enabled queue works as follows: ECN-capable packets are subject to

ECN-marking, while ECN-incapable packets follows WRED-dropping, i.e., packets are dropped if the queue size exceeds the threshold K. Therefore, the WERD/ECN feature is used to tag fast-start and stable-stage packets: uNIC sets fast-start packets to ECN-incapable and stable-stage packets ECN-capable. All the packets are mapped to the same priority queue.

If tail loss happens for the fast-start packets, the sender needs to wait for retransmission timeout. To prevent this problem, the last fast-start packet is set to ECN-capable, that is the IW-th packet if the message is larger than BDP, or the last packet of a message if its size is below BDP. The ECN-capable packet will not be dropped by WRED, so it can pass the switches and arrive at the receiver, allowing the receiver to detect if packet loss happens.

If a new flow does not finish within the fast-start stage, it will transfer to the stable stage. There are two transition conditions: 1) Packet loss is detected in the fast-start stage, which indicates the network is overloaded. cw in stable stage is set to the number of packets that are accumulatively acknowledged before packet loss. The lost packets are retransmitted using go-back-N. 2) When a full IW of packets have all been acknowledged. (IW is set to BDP as suggested in sec 2.3.) This condition is for flows that are larger than BDP and finish the fast-start stage without packet loss. Since all packets sent during fast-start stage are confirmed, the stable stage algorithm now takes over and cw is set to BDP. Note that acknowledging a BDP size of data needs two RTTs (the ACK for the IW-th packet returns at the end of second RTT), but sending BDP-sized data only requires one RTT. After the end of the first RTT, the flow will not stop sending (because the ACK of the first packet will return to free the cw) but set the packets to ECN-capable ever since.

All these implementation details are transparent to user applications. LDCP supports all RDMA transport operations (READ, WRITE, SEND, with immediate data or not, ATOMIC), and thus has full support of IB verbs.

3.2. Implementation on commercial NIC

LDCP has been implemented on Mellanox CX6-DX NIC as well. This NIC has a programmable congestion control (PCC) platform that allows users to define their own algorithms, but the RoCE protocol are standard and are implemented by ASIC. In PCC users can issue a request to measure the round-trip time (RTT), and a standalone RTT request packet will be sent among data packets to the receiver NIC. Upon receiving an RTT requet, the receiver NIC returns a standalone RTT response packet to the sender, then the sender compares the timestamp difference to calculate the RTT.

When a data-sender NIC receives ACKs, NACKs, CNPs and RTT responses, or after transmitting a burst of data, it generates corresponding type of events and pushes the events to PCC. In PCC users can define event-handling functions to calculate the transmitting rate. Afterwards, the rate is fed to the transmitting hardware to control the speed at which the data packets are put onto the wire.

LDCP is implemented by the event-handling functions. As LDCP is an AIMD algorithm, the AI logic means the window size is increased by a fixed step per-RTT, and the MD logic reveals that window is decreased by beta upon *every* CNP. Therefore, MD can be easily implemented in the CNP handling function, while the difficulty is how to implement AI since standard RoCE does not have per-packet ACK for Send/Write requests, and Read Responses do not have ACKs at all. Eventually, the implementation of AI resorts to the RTT request and response. The RTT request is issued in this way: at the beginning of a flow, an RTT request is sent out, and the next RTT request is sent after the RTT response of the previous request is received (or a timeout is experienced). Upon the arrival of an RTT response, it is for sure that one RTT has elapsed and the window should increase by alpha. Therefore, the AI logic is implemented in the RTT response handling function where the window grows by alpha. Divided by RTT, the window is converted to rate, and the rate is provided to the TX pipeline via an interface in PCC.

In conclusion, the LDCP implementation on Mellanox CX6-DX is quite straightforward and does not require any customization. Evaluation results reveal that LDCP outperforms DCQCN and TIMELY remarkably in both throughput and latency.

4. IANA Considerations

This document makes no request of IANA.

5. Security Considerations

To be added.

6. References

6.1. 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>.

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001, <https://www.rfc-editor.org/info/rfc3168>.

6.2. Informative References

[Alizadeh2010data]
Alizadeh, M., Greenberg, A., Maltz, D., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., and M. Sridharan, "Data Center TCP (DCTCP)", ACM SIGCOMM 63-74, 2010.

[CiscoGuide2012]
"Cisco IOS Quality of Service Solutions Configuration Guide", , 2012.

[Floyd1993random]
Floyd, S. and V. Jacobson, "Random early detection gateways for congestion avoidance", IEEE/ACM Transactions on networking 1, 4 (1993), 397-413, 1993.

[Guo2016rdma]
Guo, C., Wu, H., Deng, Z., Soni, G., Ye, J., Padhye, J., and M. Lipshteyn, "RDMA over commodity Ethernet at scale", ACM SIGCOMM 202-215, 2016.

[Kuzmanovic2005power]
Kuzmanovic, A., "The power of explicit congestion notification", ACM SIGCOMM 61-72, 2005.

Authors' Addresses

Huichen Dai (editor)
Huawei
Huawei Mansion, No.3, Xinxi Road, Haidian District
Beijing
China

Email: daihuichen@huawei.com

Binzhang Fu
Huawei
Huawei Mansion, No.3, Xinxi Road, Haidian District
Beijing
China

Email: fubinzhang@huawei.com

Kun Tan
Huawei
Huawei Mansion, No.3, Xinxi Road, Haidian District
Beijing
China

Email: kun.tan@huawei.com