Workgroup: Network Working Group
Published: 29 February 2024
Intended Status: Standards Track
Expires: 1 September 2024
Authors: R. Miao S. Anubolu R. Pan J. Lee B. Gafni
Meta Broadcom Inc AMD Google NVIDIA
J. Tantsura A. Alemania Y. Shpigelman
NVIDIA Intel NVIDIA

HPCC++: Enhanced High Precision Congestion Control

Abstract

Congestion control (CC) is the key to achieving ultra-low latency, high bandwidth and network stability in high-speed networks. However, the existing high-speed CC schemes have inherent limitations for reaching these goals.

In this document, we describe HPCC++ (High Precision Congestion Control), a new high-speed CC mechanism which achieves the three goals simultaneously. HPCC++ leverages inband telemetry to obtain precise link load information and controls traffic precisely. By addressing challenges such as delayed signaling during congestion and overreaction to the congestion signaling using inband and granular telemetry, HPCC++ can quickly converge to utilize all the available bandwidth while avoiding congestion, and can maintain near-zero in-network queues for ultra-low latency. HPCC++ is also fair and easy to deploy in hardware, implementable with commodity NICs and switches.

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 1 September 2024.

Copyright Notice

Table of Contents

1. Introduction

The link speed in data center networks has grown from 1Gbps to 100Gbps in the past decade, and this growth is continuing. Ultralow latency and high bandwidth, which are demanded by more and more applications, are two critical requirements in today's and future high-speed networks.

Given that traditional software-based network stacks in hosts can no longer sustain the critical latency and bandwidth requirements as described in [Zhu-SIGCOMM2015], offloading network stacks into hardware is an inevitable direction in high-speed networks. As an example, large-scale networks with RDMA (remote direct memory access) often uses hardware-offloading solutions. In some cases, the RDMA networks still face fundamental challenges to reconcile low latency, high bandwidth utilization, and high stability.

This document describes a new congestion control mechanism, HPCC++ (Enhanced High Precision Congestion Control), for large-scale, high-speed networks. The key idea behind HPCC++ is to leverage the precise link load information from signaled through inband telemetry to compute accurate flow rate updates. Unlike existing approaches that often require a large number of iterations to find the proper flow rates, HPCC++ requires only one rate update step in most cases. Using precise information from inband telemetry enables HPCC++ to address the limitations in current congestion control schemes. First, HPCC++ senders can quickly ramp up flow rates for high utilization and ramp down flow rates for congestion avoidance. Second, HPCC++ senders can quickly adjust the flow rates to keep each link's output rate slightly lower than the link's capacity, preventing queues from being built-up as well as preserving high link utilization. Finally, since sending rates are computed precisely based on direct measurements at switches, HPCC++ requires merely three independent parameters that are used to tune fairness and efficiency.

HPCC++ is an enhanced version of [SIGCOMM-HPCC]. HPCC++ takes into account system constraints and aims to reduce the design overhead and further improves the performance. Section 6 describes these detailed proposed design enhancements and guidelines.

This document describes the architecture changes in switches and end-hosts to support the needed tranmission of inband telemetry and its consumption, that imporves the efficiency in handling network congestion.

2. Terminology

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. System Overview

Figure 1 shows the end-to-end system that HPCC++ operates in. During the traverse of the packet from the sender to the receiver, each switch along the path inserts inband telemetry that reports the current state of the packet's egress port, including timestamp (ts), queue length (qLen), transmitted bytes (txBytes), and the link bandwidth capacity (B), together with switch_ID and port_ID. When the receiver gets the packet, it may copy all the inband telemetry recorded from the network to the ACK message it sends back to the sender, and then the sender decides how to adjust its flow rate each time it receives an ACK with network load information. Alternatively, the receiver may calculate the flow rate based on the inband telemetry information and feedback the calculated rate back to the sender. The notification packets would include delayed ack information as well.

Note that there also exist network nodes along the reverse (potentially uncongested) path that the feedback reports traverse. Those network nodes are not shown in the figure for sake of brevity.

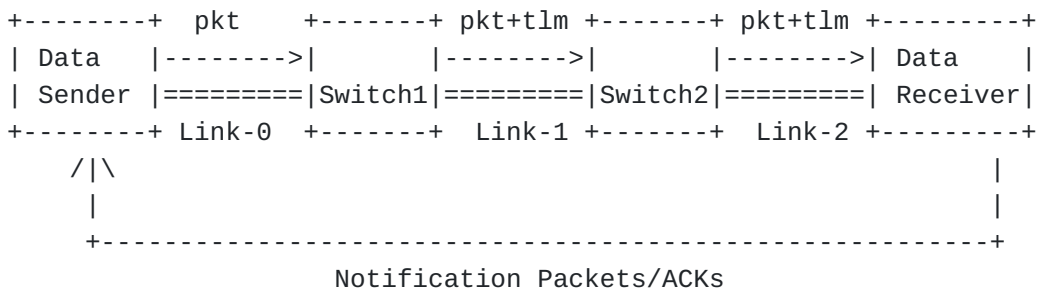

Figure 1: System Overview (tlm=inband telemtry)

*Data sender: responsible for controlling inflight bytes. HPCC++ is a window-based congestion control scheme that controls the number of inflight bytes. The inflight bytes mean the amount of data that have been sent, but not acknowledged by the sender yet. Controlling inflight bytes has an important advantage compared to controlling rates. In the absence of congestion, the inflight bytes and rate are interchangeable with equation inflight = rate

* T where T is the base propagation RTT. The rate can be calculated locally or obtained from the notification packet. The sender may further use the data pacing mechanism, potentially implemented in hardware, to limit the rate accordingly.

*Network nodes: responsible of inserting the inband telemetry information to the data packet. The inband telemetry information reports the current load of the packet's egress port, including timestamp (ts), queue length (qLen), transmitted bytes (txBytes), and link bandwidth capacity (B). Besides, the inband telemetry contains switch_ID and port_ID to identify a link.

*Data receiver: responsible for either reflecting back the inband telemetry information in the data packet or calculating the proper flow rate based on network congestion information in inband telemetry and sending notification packets back to the sender.

4. HPCC++ Algorithm

HPCC++ is a window-based congestion control algorithm. The key design choice of HPCC++ is to rely on network nodes to provide fine-grained load information, such as queue size and accumulated tx/rx traffic to compute precise flow rates. This has two major benefits: (i) HPCC++ can quickly converge to proper flow rates to highly utilize bandwidth while avoiding congestion; and (ii) HPCC++ can consistently maintain a close-to-zero queue for low latency.

This section introduces the list of notations and describes the core congestion control algorithm.

4.1. Notations

This section summarizes the list of variables and parameters used in the HPCC++ algorithm. Figure 3 also includes the default values for choosing the algorithm parameters either to represent a typical setting in practical applications or based on theoretical and simulation studies.

Notation	Variable Name
W_i	Window for flow i
Wc_i	Reference window for flow i
B_j	Bandwidth for Link j
I_j	Estimated inflight bytes for Link j
U_j	Normalized inflight bytes for Link j
qlen	Telemetry info: link j queue length
txRate	Telemetry info: link j output rate
ts	Telemetry info: timestamp
txBytes	Telemetry info: link j total transmitted bytes associated with timestamp ts

Figure 2: List of variables.

Notation	Parameter Name	Default Value
T	Known baseline RTT	5us
eta	Target link utilization	95%
maxStage	Maximum stages for additive increases	5
N	Maximum number of flows	...
W_ai	Additive increase amount	...

Figure 3: List of algorithm parameters and their default values.

4.2. Design Functions and Procedures

The HPCC++ algorithm can be outlined as below:

```
1: Function MeasureInflight(ack)
2:    u = 0;
3:    for each link i on the path do
4:                  ack.L[i].txBytes-L[i].txBytes
          txRate =  ---------------------------- ;
                        ack.L[i].ts-L[i].ts
5:               min(ack.L[i].qlen,L[i].qlen)      txRate
           u' = ---------------------------- +  ---------- ;
                     ack.L[i].B*T                ack.L[i].B
6:         if u' > u then
7:             u = u'; tau = ack.L[i].ts -  L[i].ts;
8:     tau = min(tau, T);
9:     U = (1 - tau/T)*U + tau/T*u;
10:    return U;
```

Figure 4

```
11: Function ComputeWind(U, updateWc)
12:    if U >= eta or incStage >= maxStagee then
13:             Wc
           W = ----- + W_ai;
               U/eta
14:        if updateWc then
15:            incStagee = 0; Wc = W ;
16:    else
17:        W = Wc + W_ai ;
18:        if updateWc then
19:            incStage++; Wc = W ;
20:    return W
```

Figure 5

```
21: Procedure NewAck(ack)
22:    if ack.seq > lastUpdateSeq then
23:        W = ComputeWind(MeasureInflight(ack), True);
24:        lastUpdateSeq = snd_nxt;
25:    else
26:        W = ComputeWind(MeasureInflight(ack), False);
27:    R = W/T; L = ack.L;
```

Figure 6

The above illustrates the overall process of CC at the sender side for a single flow. Each newly received ACK message triggers the procedure NewACK at Line 21. At Line 22, the variable lastUpdateSeq is used to remember the first packet sent with a new W c , and the sequence number in the incoming ACK should be larger than lastUpdateSeq to trigger a new sync betweenW c andW (Line 14-15 and 18-19). The sender also remembers the pacing rate and current inband telemetry information at Line 27. The sender computes a new window size W at Line 23 or Line 26, depending on whether to update W c , with function MeasureInflight and ComputeWind. Function MeasureInflight estimates normalized inflight bytes with Eqn (2) at Line 5. First, it computes txRate of each link from the current and last accumulated transferred bytes txBytes and timestamp ts (Line 4). It also uses the minimum of the current and last qlen to filter out noises in qlen (Line 5). The loop from Line 3 to 7 selects maxi(Ui) in Eqn. (3). Instead of directly using maxi(Ui), we use an EWMA (Exponentially Weighted Moving Average) to filter the noises from timer inaccuracy and transient queues. (Line 9). Function ComputeWind combines multiplicative increase/ decrease (MI/MD) and additive increase (AI) to balance the reaction speed and fairness. If a sender finds it should increase the window size, it first tries AI for maxStage times with the stepWAI (Line 17). If it still finds

room to increase after maxStage times of AI or the normalized inflight bytes is above, it calls Eqn (4) once to quickly ramp up or ramp down the window size (Line 12-13).

5. Configuration Parameters

HPCC++ has three easy-to-set parameters: eta, maxStagee, and W_ai. eta controls a simple tradeoff between utilization and transient queue length (due to the temporary collision of packets caused by their random arrivals, so we set it to 95% by default, which only loses 5% bandwidth but achieves almost zero queue. maxStage controls a simple tradeoff between steady state stability and the speed to reclaim free bandwidth. We find maxStage = 5 is conservatively large for stability, while the speed of reclaiming free bandwidth is still much faster than traditional additive increase, especially in high bandwidth networks. W_ai controls the tradeoff between the maximum number of concurrent flows on a link that can sustain near-zero queues and the speed of convergence to fairness. Note that none of the three parameters are reliability-critical.

HPCC++'s design brings advantages to short-lived flows, by allowing flows starting at line-rate and the separation of utilization convergence and fairness convergence. HPCC++ achieves fast utilization convergence to mitigate congestion in almost one round-trip time, while allows flows to gradually converge to fairness. This design feature of HPCC++ is especially helpful for the workload of datacenter applications, where flows are usually short and latency-sensitive. Normally we set a very small W_ai to support a large number of concurrent flows on a link, because slower fairness is not critical. A rule of thumb is to set W_ai = W_init*(1-eta) / N where N is the expected or receiver reported maximum number of concurrent flows on a link. The intuition is that the total additive increase every round (N*W_ai) should not exceed the bandwidth headroom, and thus no queue forms. Even if the actual number of concurrent flows on a link exceeds N, the CC is still stable and achieves full utilization, but just cannot maintain zero queues.

6. Design enhancement and implementation

There are three compoments HPCC++ needs to implement: telementry padding, congestion notification, and rate update.

6.1. Inband telemetry padding at the network switches

The specifications of switch padding for inband telemetry can be found in [draft-miao-tsv-hpcc-info].

6.2. Congestion Notification

HPCC++ uses congestion notification to fetch network congestion information from switches for proper rate updates at end-hosts. Although the basic algorithm described in Section 4 is to add inband telemetry information into every data packet for optimal performance, HPCC++ supports flexible implementation choices to work seamly with transport protocol stacks. We consider congestion nofication choices in both forward and reverse directions of the traffic.

6.2.1. Forward direction Congestion detection

Forward direction is the traffic direction of data packets that experience bandwidth contention and possible network congestion. The function of congestion notification in forward direction is to fetch inband telemetry from switches. HPCC++ defines two approaches of doing this.

1. Inband with data packet.

This is basic algorithm setting described in Section 4, where the end-host inserts inband telemetry header into data packets. Switches along the path detect the inband telemetry header and correspondingly add inband telemetry information into data packet to react to congestion as soon as the very first packet observing the network congestion. This is especially helpful to reduce the risk of severe congestion in incast scenarios at the first round-trip time. In addition, original HPCC's algorithm introduction of Wc is for the purpose of solving the over-reaction issue from using this per-packet response. Different with in Section 4, end-host can choice uses every data packet or only a subset of data packets to reduce the overhead. To insert telemetry header, differet telemetry protocols have specific settings for IFA, IETF IOAM, and P4.org INT as following.

2. Probe packet.

Switches touching every data packet for inband telemetry inserting may lead to security or performance concerns, HPCC++ supports the ``out-of-band'' approach that uses special-generated probe packets at end-hosts to fetch inband telemetry from switches. Thereby, the probe packets should take the same routing path and QoS queueing with the data packets. End-hosts can generate probe packets less frequently and we recommend once per round trip time. This is it sends a new probe packet once it receives the response. In addition, the end-host issues probe packets only when it has data packet in the flight.

6.2.2. Reverse direction

Reverse direction is the receiver conveying inband telemetry back to traffic sender for rate updates. Similar to forward direction, there are also inband and out-of-band approaches.

1. Inband with ACK packet.

HPCC++ supports to use the ACK packet in transport protocols to convey the inband telemetry. TCP generates ACK packet once per every data packet or per a few data packets. With ACK packet, the receive sends accumulated inband telemetry back to sender for rate updates.

2. Notification packet.

Using ACK packet for inband telemetry notification requires transport stack modification and sometimes leads to delay in notification when certain delayed acknowledged mechanism is used. Hence, HPCC++ allows the receiver to use special-generated notification packets to deliver inband telemetry. The nofication packet is generated per each probe packet or data packet with inband telemetry.

6.3. Congestion control at NICs

6.3.1. Sender-based HPCC

Figure 7 shows HPCC++ implementation on a NIC. The NIC provides an HPCC++ module that resides on the data path of the NIC, HPCC++ modules realize both sender and receiver roles.

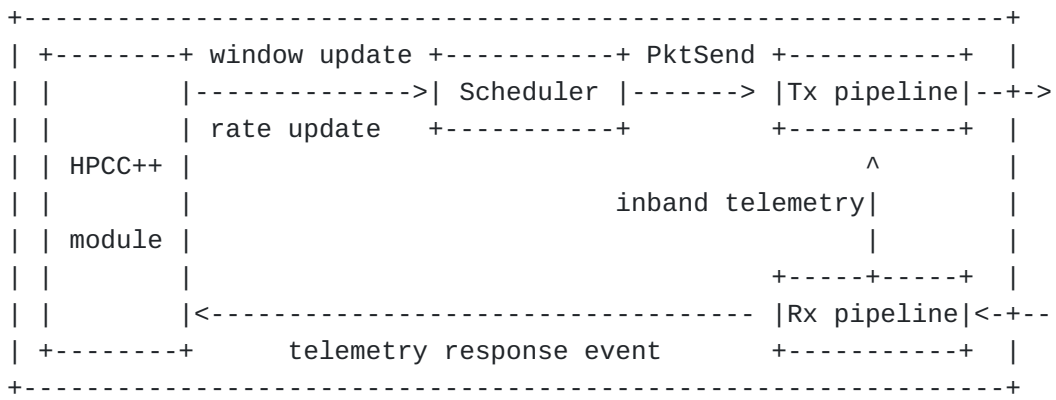

Figure 7: Overview of NIC Implementation

1. Sender side flow

The HPCC++ module running the HPCC CC algorithm in the sender side for every flow in the NIC. Flow can be defined by some transport parameters including 5-tuples, destination QP (queue pair), etc. It receives inband telemetry response events per flow which are generated from the RX pipeline, adjusts the sending window and rate, and update the scheduler on the rate and window of the flow.

The scheduler contains a pacing mechanism that determine the flow rate by the value it got from the algorithm. It also maintains the current sending window size for active flows. If the pacing mechanism and the flow's sending window permits, the scheduler invokes for the flow a PktSend command to TX pipeline.

The TX pipeline implements packet processing. Once it receives the PktSend event with flow ID from the scheduler, it generates the corresponding packet and delivers to the Network. If a sent packet should collect telemetry on its way the TX pipeline may add indications/headers that triggers the network elements to add telemetry data according to the inband telemetry protocol in use. The telemetry can be collected by the data packet or by dedicated prob packets generated in the TX pipeline.

The RX pipe parses the incoming packets from the network and identifies whether telemetry is embedded in the parsed packet. On receiving a telemetry response packet, the RX pipeline extracts the network status from the packet and passes it to the HPCC++ module for processing. A telemetry response packet can be an ACK containing inband telemetry, or a dedicated telemetry response prob packet.

2. Receiver side flow

On receiving a packet containing inband telemetry, the RX pipeline extracts the network status, and the flow parameters from the packet and passes it to the TX pipeline. The packet can be a data packet containing inband telemetry, or a dedicated telemetry request prob packet. The Tx pipeline may process and edit the telemetry data, and then sends back to the sender the data using either an ACK packet of the flow or a dedicated telemetry response packet.

6.3.2. Receiver-based HPCC

Note that the window/rate calculation can be implemented at either the data sender or the data receiver. If the ACK packets already exist for reliability purpose, the inband telemetry information can be echoed back to the sender via ACK self-clocking. Not all ACK packets need to carry the inband telemetry information. To reduce the Packet Per Second (PPS) overhead, the receiver may examine the inband telemetry information and adopt the technique of delayed ACKs that only sends out an ACK for a few of received packets. In order

to reduce PPS even further, one may implement the algorithm at the receiver and feedback the calculated window in the ACK packet once every RTT.

The receiver-based algorithm, Rx-HPCC, is based on int.L, which is the inband telemetry information in the packet header. The receiver performs the same functions except using int.L instead of ack.L. The new function NewINT(int.L) is to replace NewACK(int.L)

```
28:   Procedure NewINT(int.L)
29:   if now > (lastUpdateTime + T) then
30:      W = ComputeWind(MeasureInflight(int), True);
31:      send_ack(W)
32:      lastUpdateTime = now;
33:   else
34:      W = ComputeWind(MeasureInflight(int), False);
```

Figure 8

Here, since the receiver does not know the starting sequence number of a burst, it simply records the lastUpdateTime. If time T has passed since lastUpdateTime, the algorithm would recalcuate Wc as in Line 30 and send out the ACK packet which would include W information. Otherwise, it would just update W information locally. This would reduce the amount of traffic that needs to be feedback to the data sender.

Note that the receiver can also measure the number of outstanding flows, N, if the last hop is the congestion point and use this information to dynamically adjust W_ai to achieve better fairness. The improvement would allow flows to quickly converge to fairness without causing large swings under heavy load.

7. Reference Implementation

HPCC++ can be adopted as the CC algorithm by a wide range of transport protocols such as TCP and UDP, as well as others that may run on top of them, such as iWARP, RoCE etc. It requires to have the window limit and congestion feedback through ACK self-clocking, which naturally conforms to the paradigm of TCP design. With that, HPCC++ introduces a scheme to measure the total inflight bytes for more precise congestion control. To run in UDP, some modifications need to be done to enforce the window limit and collect congestion feedback via probing packets, which is incremental.

7.1. Implementation on RDMA RoCEv2

We describe reference implementation on RDMA RoCEv2. This is an implementation for ``Sender-based HPCC++'' (see section 6.3.1.) using dedicated probe packets to collect the telemetry. HPCC++

module in the sender triggers the sending of ``telemetry request packet'' for a given flow. The NIC then sends the probe packet. The packet will have the same IP and UDP headers as the data packets of the given flow. Such packet is expected to be sent every RTT, see section 6 for more details. On receiving of telemetry request packet, the NIC extracts the telemetry from all the links along the path from the sender. HPCC++ module chooses the link with the highest inflight bytes and sends its telemetry (queue length, timestamp and tx bytes) back to the receiver on top of dedicated ``telemetry response packet''. On receiving of telemetry response packet, the NIC extracts the telemetry and pass it to the HPCC++ module which using this info to implement the rate update scheme.

7.2. Implementation on TCP

Taking the benefit of precise congestion control for TCP is a natural next step. Since TCP segmentation at TX side (e.g., TSO) and coalescing at RX side (e.g., GRO) happen at the NIC HW or low-layer of TCP/IP stack, carrying per-pkt inband telemetry info between the TCP congestion control engine and network fabric has to work with the TSO and GRO. Instead, one way to adopt HPCC++ for TCP is using the special probe and notification packets to retrieve inband telemetry information. The sender generates a probe packet when it is actively sending data. The probe packet has the same 5-tuples (source and destination addresses, source and destination ports and protocol number) with the data packets and the inband telemetry header. The switches along the path identify the probe packet by its inband telemetry header and insert the inband telemetry. Once received the probe packet with inband telemetry, the receiver replies with a response packet piggybacking the inband telemetry to the sender. Note, both probe and response packets use a special DSCP number so that it can bypass the TSO and GRO in each side.

8. IANA Considerations

This document makes no request of IANA.

9. Discussion

9.1. Internet Deployment

Although the discussion above mainly focuses on the data center environment, HPCC++ can be adopted at Internet at large. There are several security considerations one should be aware of.

There may rise privacy concern when the telemetry information is conveyed across Autonomous Systems (ASes) and back to end-users. The link load information captured in telemetry can potentially reveal the provider's network capacity, route utilization, scheduling policy, etc. Those usually are considered to be sensitive data of

the network providers. Hence, certain action may take to anonymize the telemetry data and only convey the relative ratio in rate adaptation across ASes without revealing the actual network load.

Another consideration is the security of receiving telemetry information. The rate adaptation mechanism in HPCC++ relies on feedback from the network. As such, it is vulnerable to attacks where feedback messages are hijacked, replaced, or intentionally injected with misleading information resulting in denial of service, similar to those that can affect TCP. It is therefore RECOMMENDED that the notification feedback message is at least integrity checked. In addition, [I-D.ietf-avtcore-cc-feedback-message] discusses the potential risk of a receiver providing misleading congestion feedback information and the mechanisms for mitigating such risks.

9.2. Switch-assisted congestion control

HPCC++ falls in the general category of switch-assisted congestion control. However, HPCC++ includes a few unique design choices that are different from other switch-assisted approaches.

*First, HPCC++ implements a primal-mode algorithm that requires only the ``write-to-packet'' operation from switches, which has already been supported by telemetry protocols like INT [P4-INT] or IOAM [I-D.ietf-ippm-ioam-data]. Please note that this is very different from dual-mode algorithms such as XCP [Katabi-SIGCOMM2002] and RCP [Dukkipati-RCP], where switches take an actively role in determining flows' rates.

*Second, HPCC++ achieves a fast utilization convergence by decoupling it from fairness convergence, which is inspired by XCP.

*Third, HPCC++ enables the switch-guided multiplicative increase (MI) by defining the ``inflight byte'' to quantify the link load. The inflight byte tells both the underload and overload of the link precisely and thus it allows the flow to increase/decrease the rate multiplicatively and safely. By contrast, traditional approaches of using the queue length or RTT as the feedback cannot guide the rate increase and instead have to rely on additive increase (AI) with heuristics. As the link speed contines to grow, this becomes increasingly slow in reclaiming the unused bandwidth. Besides, queue-based feedback mechanisms subject to latency inflation.

*Last, HPCC++ uses TX rate instead of RX rate used by XCP and RCP. As detailed in [SIGCOMM-HPCC], we view the TX rate is more

precise because RX rate and queue length are overlapped and thus it causes oscillation.

9.3. Work with multiple queues

When using QoS (Quality of Service) priority queueing in switches, a flow cannot determine the actual queueing time and the extent of congestion, if any, based purely on the length of the queue assigned to it. Although general approaches for running congestion control with QoS queueing are out of the scope of this document, we provide several guidelines. In cases where telemetry for other queues is unavailable (e.g., due to lack of hardware support, security reasons), a flow's queue length is insufficient. Instead, HPCC++ can leverage a packet's sojourn time (i.e., the egress timestamp minus the ingress timestamp) to determine the actual queueing time the packet experiences. Furthermore, if the switch employs Deficit Weighted Round Robin (DWRR) QoS scheduling, which is typical of many operators, HPCC++ can leverage the DWRR minimum bandwidth guarantee for each queue to derive precise rate updates to avoid congestion. In the worst case where packet sojourn time and other telemetry is unavailable, an operator can instead use the bandwidth allocated to the flow's QoS class instead of a link's full bandwidth in the HPCC++ algorithm. In this case, HPCC++ can operate normally when the flow's calculated transmit rate is less than or equal to the bandwidth allocated to its QoS class. If the transmit rate exceeds the allocated bandwidth (due to additive increase) and the packet's queue is empty, a flow can make the reasonable inference that one or more other QoS classes are under-utilized and, thus, some portion of their allocated bandwidth can be used. Since precise telemetry on the transmit rate of other QoS classes is unavailable, a flow can adopt several strategies for increasing bandwidth via manipulation of U. For example, a flow can adopt a binary search approach by jumping halfway between its current rate and link rate. While operating in this mode, a flow may eventually encounter queueing and reasonably infer that it is using more additional bandwidth than is available (recall that a flow cannot precisely determine how much additional bandwidth is available because it does not have telemetry on the utilization of other QoS queues). Upon detecting this scenario, a flow can reduce its bandwidth by adopting strategies like those used to increase its bandwidth when it detects that additional bandwidth is available. In cases where telemetry for other queues is available, an operator can use the bandwidth allocated to a flow's QoS class as described in the previous paragraph. However, upon detecting that additional bandwidth (relative to a flow's QoS-allocated bandwidth) is available, an operator can leverage the telemetry of other QoS queues to intelligently determine the appropriate amount of additional bandwidth to consume. An example strategy is where a flow takes a proportional share of the additional bandwidth based on the ratio of

the bandwidth allocated to its QoS class to the sum of the bandwidth of all active QoS classes. This ensures each active QoS class gets its fair share. Using QoS telemetry in this way enables convergence to ideal rates in one rate update. In contrast, the binary search scheme described for the no-telemetry case would likely require multiple rate updates, and it may cause a flow to overshoot how much bandwidth it should consume.

9.4. Path migration

HPCC++ allows switches and end-hosts to share precise information of network utilization, which suggests a framework for path selection and rate control at end-hosts. The framework HPCC++ enabled is to leverage each switch to report its link load information via inband telemetry. The end-host fetches inband telemetry along the traffic routes and makes a timely and accurate decision on path selection and traffic admission.

10. Security Considerations

TBD

11. Acknowledgments

The authors would like to thank RTGWG members for their valuable review comments and helpful input to this specification.

12. Contributors

The following individuals have contributed to the implementation and evaluation of the proposed scheme, and therefore have helped to validate and substantially improve this specification: Pedro Y. Segura, Roberto P. Cebrian, Robert Southworth and Md Ashiqur Rahman.

13. 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>.

14. Informative References

[I-D.ietf-avtcore-cc-feedback-message] Sarker, Z., Perkins, C., Singh, V., and M. A. Ramalho, "RTP Control Protocol (RTCP) Feedback for Congestion Control", Work in

Progress, Internet-Draft, draft-ietf-avtcore-cc-feedback-message-09, 2 November 2020, <https://datatracker.ietf.org/doc/html/draft-ietf-avtcore-cc-feedback-message-09>.

[Katabi-SIGCOMM2002] Katabi, D., Handley, M., and C. Rohrs, "Congestion Control for High Bandwidth-Delay Product Networks", ACM SIGCOMM Pittsburgh, Pennsylvania, USA, October 2002.

[draft-miao-tsv-hpcc-info] Miao, R., "HPCC++: Enhanced High Precision Congestion Control (Informational)", June 2022.

[Zhu-SIGCOMM2015]
Zhu, Y., Eran, H., Firestone, D., Guo, C., Lipshteyn, M., Liron, Y., Padhye, J., Raindel, S., Yahia, M. H., and M. Zhang, "Congestion Control for Large-Scale RDMA Deployments", ACM SIGCOMM London, United Kingdom, August 2015.

[P4-INT] "In-band Network Telemetry (INT) Dataplane Specification, v2.0", February 2020, <https://github.com/p4lang/p4-applications/blob/master/docs/INT_v2_0.pdf>.

[I-D.ietf-ippm-ioam-data] "Data Fields for In-situ OAM", March 2020, <https://tools.ietf.org/html/draft-ietf-ippm-ioam-data-09>.

[SIGCOMM-HPCC]
Li, Y., Miao, R., Liu, H., Zhuang, Y., Fei Feng, F., Tang, L., Cao, Z., Zhang, M., Kelly, F., Alizadeh, M., and M. Yu, "HPCC: High Precision Congestion Control", ACM SIGCOMM Beijing, China, August 2019.

[Dukkipati-RCP] Dukkipati, N., "Rate Control Protocol (RCP): Congestion control to make flows complete quickly.", Stanford University , 2008.

Authors' Addresses

Rui Miao
Meta
1 Hacker Way
Menlo Park, CA 94025
United States of America

Email: rmiao@meta.com

Surendra Anubolu
Broadcom, Inc.

1320 Ridder Park
San Jose, CA 95131
United States of America

Email: surendra.anubolu@broadcom.com

Rong Pan
AMD
2485 Augustine Dr.
Santa Clara, CA 95054
United States of America

Email: Rong.Pan@amd.com

Jeongkeun Lee
Google
Headquarters 1600 Amphitheatre Parkway
Mountain View, CA 95043
United States of America

Email: leejk@google.com

Barak Gafni
NVIDIA
350 Oakmead Parkway, Suite 100
Sunnyvale, CA 94085
United States of America

Email: gbarak@NVIDIA.com

Jeff Tantsura
NVIDIA
United States of America

Email: jefftant.ietf@gmail.com

Allister Alemania
Intel
2200 Mission College Blvd
Santa Clara, 95952
United States of America

Email: allister.alemania@intel.com

Yuval Shpigelman
NVIDIA
Haim Hazaz 3A
Netanya 4247417
Israel

Email: yuvals@nvidia.com