Revision Plan for SIGCOMM'24 Paper #293

Dear Prof. Praveen Kumar,

Thank you for shepherding our paper (*#293 PPT: A Pragmatic Transport for Datacenters*). We carefully went through the reviews and outlined our revision plan (as detailed below).

Please let us know if you have any feedback. Thank you for your time and help to improve the quality of our paper!

Best Regards,
Authors of SIGCOMM'24 Paper #293

I Revision Guidance Response:

Concern 1: *Disconnect between the proposed goals vs the evaluation. The paper pitches itself to deliver comparable performance to proactive transports while being readily deployable. But the evaluations do not necessarily back this up.*

Response 1: We would like to clarify that one of PPT's goals is indeed optimizing the typical performance like overall average FCT of flows. To achieve this goal, it uses a parallel control loop design. Specifically, it runs a high-priority control loop using DCTCP, and a second low-priority control loop to utilize the spare bandwidth in the network, thus delivering low FCTs for the flows. From this point of view, our evaluation results can back this up. Please see the results achieved by our PPT and Homa (or Homa-Linux) in Fig. 8(a), Fig. 9 (a), Fig. 10(a), Fig. 11(a), Fig. 12(a), and Fig. 13(a).

Another goal of our PPT is to further optimize the performance of small flows, since the parallel control loop design treats the small and large flows equally and may lead small flows queued after large ones in switch buffer. To achieve this goal, our PPT uses a mirror-symmetric packet tagging method. This design makes PPT achieve comparable average FCT of small flows with Homa, as shown in Fig. 12(b)(c) and Fig. 13(b)(c). Note that this mirror-symmetric packet tagging is a complement component with the parallel control loop design. In case the reviewers may misunderstand this, we want to clarify the following points:

1): When a flow starts, the high-priority control loop (HCP) sends this flow's from the beginning of the send buffer. Meanwhile, the low-priority control loop (LCP) sends data from the very last byte of the send buffer. So, in general, for each flow, it contains both HCP and LCP packets during its lifetime.

2): Moreover, a flow's HCP packet sending rate is controlled by the DCTCP algorithm, and its LCP packet sending rate is controlled independently (using the intermittent loop initialization and exponential window decreasing techniques proposed by PPT).

3): For each flow, PPT lets its HCP packets use the first four high priorities while the LCP packets use the remaining four low priories. Initially, a flow's HCP and LCP packets use P0 (the highest priority of the HCP priorities) and P4 (the highest priority priority of the LCP priorities), respectively. As more bytes are sent for this flow, its HCP and LCP packet priorities will be demoted at the same pace.

So, in summary, we are not simply dividing the traffic into four class of flows and coupling flows with different priorities to utilize the spare bandwidth. Instead, we divide each individual flow into two parts: the first part runs with high priorities while the second part runs with low priorities to utilize the spare bandwidth in the network. Moreover, the priority queuing is a incremental design to further optimize the average FCT of small flows. We plan to add more text in Sec. 4 to clarify these points.

Finally, the PPT prototype implementation, the CloudLab testbed experiments and the comments about the issues in Homa-Linux implementation verify that PPT is a readily deployable solution.

Proposed modification: Our experiments indeed demonstrate that our approach delivers comparable performance to proactive transports while remaining readily deployable. Therefore, we will not add further explanations or analyses in the paper. To address the reviewers' possible misunderstandings regarding the flow scheduling scheme, we have added a summary at the end of section 2 on page 5.

Concern 2: *There's lacking discussion on other related work such as PCC Proteus.*

Response 2: PCC Proteus is radically different with PPT. First, PCC Proteus is designed for Internet while our PPT is used for datacenter network environment. Second, PCC Proteus depends on application requirements and use the traffic of low-priority applications to fill the bandwidth leftover by high-priority application. PPT uses different bandwidth padding scheme. That said, it does not rely on application's priority or requirement. It instead uses the opportunistic packets starting from the end of the same flow to fill the spare bandwidth in the network. Finally, PCC Proteus requires application to specify its priorities, while PPT gradually demotes a flow's priority as more bytes sent. We plan to discuss PCC Proteus and other related work in Appendix B.

Proposed modification: We have discussed PCC Proteus and other related work in the Appendix B on page 15.

Concern 3: *The paper needs to discuss more insights on how the system is able to deliver the performance benefits.*

Response 3: PPT's performance benefits are steamed from the following points. First, DCTCP contains spare bandwidth in slow start phase, which is obvious. Second, in queue build-up phase, DCTCP also contains some spare bandwidth. We have already shown this point using a preliminarily experiment in Fig.1. But the reviewer may consider this experiment only contains synchronized bursts. Actually, in this experiment, the flows are generated according to Poisson arrival pattern. So, there are both synchronized and nonsynchronized cases. We plan to clarify this point in our paper. Finally, PPT uses a mirror-symmetric packet tagging design. This makes PPT achieve comparable performance on the average FCT of small flows.

Proposed modification: We have added remark paragraph in section 2.3 on page 4 to show more insights.

Concern 4: *Other claims such as "integration with other transports is easy" need to be further justified.*

Response 4: We actually are not claiming that "integration with other transports is easy". What we claim is that the design of PPT can be integrated with delay-based transport. We have already justified this point in our submission. First, we have already implemented a delay-based transport conceptually equivalent to Swift in the ns3 simulator and showed that PPT's design can improve the performance of the original delay-based transport. Please see Fig. 26 for more details. On the other hand, we have also discussed about the compatibility of PPT with other transports. Please see Appendix A for more details.

Proposed modification: As relevant experiments and analyses have already been included in the paper, we will not add further discussion.

Concern 5: *Missing details on evaluation setup and parameters.*
Response 5: We will complete the missing details about the experimental setup and parameters in the form of tables.
Proposed modification: We have added a table to fill in the missing parameters on page 15.

II Reviewers Response

1 Reviewer #293A

A.1: Reviewer Comment: *The most significant shortcoming of the paper is that its goals are not defined sufficiently clearly. At first, the Introduction states: "In this work, we aim to explore a new transport that can achieve comparable performance to proactive transports while being readily deployable." So we would expect the paper to be about optimizing the typical performance, e.g. the median or 99th-percentile FCT. But reverse-engineering from the results, it looks like the paper is about entirely different goals: being able to implement a strict multi-priority scheme between four classes of traffic: (a) high-priority short flows, (b) high-priority long flows, (c) low-priority short flows, and (d) low-priority long flows. This can harvest leftover bandwidth, and enable short flows to complete more quickly. The paper should clarify if that is indeed the goal. In addition, it should specify metrics of success by which we can quantify that the goals are achieved.*
Response A.1 Thanks for the recommendations. We would like to clarify that one of PPT's goals is indeed optimizing the typical performance like overall average FCT of flows. PPT uses a dual-loop rate control design to achieve this goal. Specifically, it runs a high-priority control loop using DCTCP, and a second low-priority control loop to utilize the spare bandwidth in the network, thus delivering low FCTs for the flows. From this point of view, our evaluation results can back this up. Please see the results achieved by our PPT and Homa (or Homa-Linux) in Fig. 8(a), Fig. 9 (a), Fig. 10(a), Fig. 11(a), Fig. 12(a), and Fig. 13(a).

Another goal of our PPT is to further optimize the performance of small flows, since the parallel control loop design treats the small and large flows equally and may lead small flows queued after large ones in switch buffer. To achieve this goal, our PPT uses a mirror-symmetric packet tagging method. This design makes PPT achieve comparable average FCT of small flows with Homa, as shown in Fig. 12(b)(c) and Fig. 13(b)(c). Note that this mirror-symmetric packet tagging is a complement component with the parallel control loop design. In case the reviewers may misunderstand this, we want to clarify the following points:

1): When a flow starts, the high-priority control loop (HCP) sends this flow's from the beginning of the send buffer. Meanwhile, the low-priority control loop (LCP) sends data from the very last byte of the send buffer. So, in general, for each flow, it contains both HCP and LCP packets during its lifetime.

2): Moreover, a flow's HCP packet sending rate is controlled by the DCTCP algorithm, and its LCP packet sending rate is controlled independently (using the intermittent loop initialization and exponential window decreasing techniques proposed by PPT).

3): For each flow, PPT lets its HCP packets use the first four high priorities while the LCP packets use the remaining four low priories. Initially, a flow's HCP and LCP packets use P0 (the highest priority of the HCP priorities) and P4 (the highest priority priority of the LCP priorities), respectively. As more bytes are sent for this flow, its HCP and LCP packet priorities will be demoted at the same pace.

So, in summary, we are not simply dividing the traffic into four class of flows and coupling flows with different priorities to utilize the spare bandwidth. Instead, we divide each individual flow into two parts: the first part runs with high priorities while the second part runs with low priorities to utilize the spare

bandwidth in the network. Moreover, the priority queuing is a incremental design to further optimize the average FCT of small flows. We plan to add more text in Sec. 4 to clarify these points.

Proposed modification: To address the reviewers' possible misunderstandings regarding the flow scheduling scheme, we have added a summary at the end of section 2 on page 5.

A.2: Reviewer Comment: *The number of used queues could be clarified: p. 6 mentions two such queues (a high-priority and a low-priority queue with a different λ). But then it mentions that when queueing the opportunistic (presumably low-priority) packets, this may hurt normal (presumably high-priority) packets, suggesting they share the queue. Then, p. 8 mentions 8 priorities, such that "switches can use strict priority to dequeue packets", implicitly pointing to 8 different queues. Then, p.9 mentions 2 queues again.*

More significantly, if we indeed have such priority queueing, then what is the goal of the research beyond setting up these queues? What does the paper intend to show? It seems that with priority queueing, we already achieve the four priority types, that each priority provides protection from lower-priority flows, and that the low-priority flows can indeed harvest the spare bandwidth. Again, the paper should clarify its motivation.

Response A.2 Thanks for the reviewer's comments. PPT uses 8 priority queues (P0>P1> $\cdots$ >P7) in the switch. HCP packets use priorities P0$\sim$P3 while LCP packets use P4$\sim$P7. Therefore, the *high-priority* and *low-priority* mentioned in p.6 and p.9 refer to priorities P0$\sim$P3 and P4$\sim$P7 in the switch, respectively. To avoid ambiguity, we will include footnotes in Section 3.2 to explain this point.

On the other hand, as we have shown in the above response, the priority queuing design does not affect the primary goal of optimizing the overall average FCT of flows. It is an add-on design on the parallel control loop to further optimize the average FCT of small flows.

Proposed modification: Since we have already explained this in detail in the paper, we will not include further explanations.

A.3: Reviewer Comment: *A goal of the paper is to obtain a "scavenger" algorithm that can fully use spare bandwidth. This was the main goal of PCC Proteus (ACM SIGCOMM'20), maybe the best-known such scavenger algorithm. It is surprising that the paper does not quote it and compare against it.*

Response A.3 PPT is radically different from the "scavenger" algorithm like PCC Proteus. First, PCC Proteus is designed for Internet while our PPT is used for datacenter network environment. Second, PCC Proteus depends on application requirements and use the traffic of low-priority applications to fill the bandwidth leftover by high-priority application. PPT uses different bandwidth padding scheme. That said, it does not rely on application's priority or requirement. It instead uses the opportunistic packets starting from the end of the same flow to fill the spare bandwidth in the network. Finally, PCC Proteus requires application to specify its priorities, while PPT gradually demotes a flow's priority as more bytes sent. We will plan to discuss PCC Proteus and other related work in Appendix B.

Proposed modification: We have discussed PCC Proteus in other related work section on page 15.

A.4: Reviewer Comment: *Another major issue is that the paper somehow gets a free lunch, and we do not get much intuition about it. PPT handily beats state-of-the-art algorithms on small-flow FCTs, which was expected. We would expect a small hit for long-flow FCTs on the other hand. Instead, it also easily beats them for long-flow FCTs. The paper should devote much more space to explain the intuition behind this out-performance, and convey why this is not just due to the choice of the network and/or workload parameters. Is it due to the better bandwidth utilization? To the use of priorities? Would it still hold if the competing algorithm had 4 priorities with DCTCP or HPCC in each?*

Response A.4 Yes. PPT's large flow's performance benefit is mainly because PPT can dynamically identify and fill the spare bandwidth in the network. Besides, the use of priorities is mainly for flow scheduling to further optimize the performance of small flows by prioritizing the packets of small flows over those of large ones in switch buffer. We have already verified these points in design space section (Sec. 2.3).

Regarding the 4 classes (or priorities) of traffic, we have already claimed in the above response that we are not simply dividing the traffic into four class of flows and coupling flows with different priorities to utilize the spare bandwidth. Instead, we divide each individual flow into two parts: the first part runs with high priorities while the second part runs with low priorities to utilize the spare bandwidth in the network. So, it makes little sense to compare PPT with an algorithm that has 4 priorities with DCTCP or HPCC in each. But we do have already compared PPT with HPCC directly and the results demonstrated that PPT outperforms HPCC. Please see Fig. 16 for more details.

Proposed modification: As the paper already includes detailed analyses of the performance benefits of PPT, we will not add additional discussion. To address the reviewers' possible misunderstandings regarding the flow scheduling scheme, we have added a summary at the end of section 2 on page 5.

A.5: Reviewer Comment: *Filling the gap up to the DCTCP max congestion window sounds good when traffic is completely static. But if DCTCP uses the full link bandwidth when alone, then 1/11th of the bandwidth when 10 more flows arrive, it is hard to understand why a new opportunistic flow would suddenly take 1/2*10/11, i.e nearly 50% of the link rate. In fact, if W_{max} were the correct rate for DCTCP, then DCTCP could have used it as well. There is a reason why DCTCP has decreased its window. Likewise, while at the start the DCTCP flow starts cautiously, the opportunistic flow starts with a near-full BDP of traffic. Why?*

Response A.5 Thanks for the reviewer's very good comments. This is really a good question and a very extreme example. Let's rephrase the example the reviewer mentioned. Saying that there is a flow 1 that is currently using the full link bandwidth. Then, another 10 flows (flow 2,$\cdots$,11) arrive. In this case, the opportunistic packets for flow 1 will not be sent, because queue build-up forms and more ECN-marked packets will be exhibited by flow 1, thus leading the value of its α keep increasing. Whereas, PPT only initialize an LCP loop and send opportunistic packet for a flow when its α takes minimum value.

On the other hand, PPT will open an LCP loop for each of the flow in flows 2$\sim$11 (if they are identified as small flows) to let them send opportunistic packets. However, these opportunistic packets may be queued or dropped due to buffer overflow, thus their LCP loops will be closed in a short while.

Proposed modification: We will not analyse this extreme case further in the paper.

A.6: Reviewer Comment: *In addition, PPT seems to couple each LCP flow with an HCP flow, by defining the window size of the LCP flow using the window size of the associated HCP flow. But what if there are no HCP flows now? Are LCP flows stuck? Or more generally, what if the numbers of flows are not equal? Or what if they are equal but the flows are destined to different destinations? How does this coupling help?*

Response A.6: Thanks for the reviewer's comments. Actually, we are not simply dividing the traffic into four class of flows and coupling flows with different priorities to utilize the spare bandwidth. Instead, we divide each individual flow into two parts: the first part runs with high priorities while the second part runs with low priorities to utilize the spare bandwidth in the network.

Proposed modification: To address the reviewers' possible misunderstandings regarding the flow scheduling scheme, we will add an analysis under the title of section 4 on page 6 and include additional explanations in the footnotes on page 7.

2 Reviewer #293B

B.1: Reviewer Comment: *During LCP loop initialization for case 1, it is unclear how PPT computes BDP in the first RTT. As you mention, DCTCP is still probing for available bandwidth.*

Response B.1: Thanks for the reviewer's comments. BDP is computed as $BDP = rtt * rate_{nic}$, where $rate_{nic}$ is the rate of the network interface card, and rtt is the base RTT in the network.

Proposed modification: Based on the above analysis, we will not add further explanations in the paper.

B.2: Reviewer Comment: *I wonder if tracking α_{min} can lead to underutilization due to network dynamics. Suppose we have a long flow which experiences transient congestion at the beginning (say due to an incast); the trans ient congestion causes α_{min} to be $>$ 0.5 for the flow. After the transient congestion subsides, the large flow is unable to utilize any spare bandwidth arising from Case 2.*

Response B.2: Thanks for the reviewer's comments. It is very rarely observed in our experiments that α is greater than 0.5, as it requires the number of ECN marked packets to be more than half of the number of all packets over many sequential RTTs. On the other hand, even if α is greater than 0.5: it represents the network experienced severe congestion and there is no spare bandwidth that can be used for LCP.

Proposed modification: As the paper already contains a detailed analysis of the α_{min}, we will not add further explanations

B.3: Reviewer Comment: *It is unclear how PPT deals with fairness as flows arrive and leave. Suppose two large flows sharing a common bottleneck link arrive one after the other. The latter flow might see much lower W_{max} (50%?) than the earlier flow. In every LCP loop initialization (case 2), the latter flow would compute a lower value of initial congestion window and send less packets via LCP compared to the earlier flow.*

Response B.3: Thanks for the reviewer's comments. To be frank, we have not taken into account the fairness issue in this scenario. We will discuss this point in Appendix A.

Proposed modification: We have added an analysis in the footnote on page 6.

B.4: Reviewer Comment: *As the paper targets the issue of underutilizing high datacenter bandwidth, I wonder if some of the simulations could have been done at higher line rates (400+ Gbps) to demonstrate the problem more clearly.*

Response B.4: Thanks for the reviewer's comments. We will add a new simulation with 100/400G topology to show this point.

Proposed modification: We have added a new simulation with 100/400G topology to demonstrate this point. The results and analysis of this experiment have been presented in Appendix C on page 18 of the paper.

B.5: Reviewer Comment: *What are the overheads of PPT?*

Response B.5: Thanks for the reviewer's comments. We have already measured the kernel space CPU overhead of PPT and DCTCP, and the results shown that PPT only incurs a slightly higher CPU usage (less than 1%) than DCTCP. Please see Fig. 21 in Appendix C.1 for more details.

Proposed modification: We will not add further discussion as the appendix has already contained relevant experiments.

3 Reviewer #293C

C.1: Reviewer Comment: *It would be particularly useful to have an intuitive and straightforward example scenario where LCP can discover spare bandwidth other than during the slow start phase.*

Response C.1: Appreciate the reviewer's comments. In Fig. 1, we actually have already shown that other than during the slow start phase, DCTCP also contains spare bandwidth in queue build-up phase. The reviewer may consider Fig. 1 only contains synchronized bursts. Indeed, in the experiment of Fig. 1, the flows are generated according to Poisson arrival pattern. So, there are both synchronized and nonsynchronized cases. We plan to clarify this point in our paper.

Proposed modification: We will show more details and analysis in Section 2.3 on page 4.

C.2: Reviewer Comment: *Figure 13 doesn't add up by itself. RC3 has the highest average FCT for small and large flows. However, its overall average is not the highest. Given that large and small flows account for 13% and 87% respectively, the overall average should be around 0.13 * 39.63 + 0.87 * 0.77 = 5.8? Or am I missing something?*

Response C.2: Thanks for the reviewer's very good comments. This may be a typing error when drawing the figure. We will revise it accordingly.

Proposed modification: We will revise it on page 11.

C.3: Reviewer Comment: *Page 12, can you give an intuition why limiting RC3's low priority queue doesn't help? It kind of contradicts the argument that aggressive low priority packets hinder high-priority packets being a problem if the high-priority queue has enough buffers?*

Response C.3: Thanks for the reviewer's comments. We will add more intuitive reasons in Section 6.2 to show this point.

Proposed modification: We will add more intuitive reasons in Section 6.2 on page 12.

4 Reviewer #293D

D.1: Reviewer Comment: *Overall, given the recent research on protocols that are both (a) simpler than DCTCP to deploy and (b) deliver better performance than it, it is unclear to me why one would want to have an "add-on" system (with its own additional complexities) to help it perform better.*

Response D.1 Thanks for the reviewer's good comments. In fact, DCTCP is widely deployed in production and used as the default congestion control algorithm in linux kernel for many years. Therefore, DCTCP has fewer barriers to adoption and a broader reach compared to other transport protocols. Moreover, through our design and experiments, we have already demonstrated that we can achieve good performance by utilizing the spare bandwidth of DCTCP, while retaining the compatibility with legacy TCP/IP network stacks. These factors motivate us to design an add-on system on DCTCP.

Proposed modification: As we have already explained the motivation in section 2.3, we will not add further explanations.

D.2: Reviewer Comment: *Related work: The paper misses the recent delay-based congestion control algorithms such as Timely and Swift both in the qualitative discussions (e.g., Table 1) and in the quantitative comparisons in the evaluation section. Appendix C provides some data but needs some clarification. What exactly is "a delay-based transport (conceptually equivalent to Swift)." What is the actual algorithm?*

Response D.2 Thanks for the comments. We will add the discussions with Timely and Swift in Related Work section (Appendix B). By the way, Swift's variant actually doesn't include the component that handles endpoint congestion. We will clarify this in Appendix C.3.

Proposed modification: We have discussed Swift and Timely in other related work section on page 15 and show more details about the actual algorithm in the footnotes on page 17.

D.3: Reviewer Comment: *I'm not sure if I fully understand some of the experimental results. To give an example, why is Homa's performance so poor for the testbed incast experiment? You mention that PPT is effective because it can recover quickly but why can't Homa?*

Response D.3: Thanks for the reviewer's comments. Note that the original Homa SIGCOMM'18 paper does not provide implementation code, so we use its Homa-Linux implementation (which is published in ATC'21). However, Homa-Linux over-tunes the entire TCP/IP network stack and suffers from several implementation issues (as we have already discussed in Section 6.1.1).

Proposed modification: As it has been analyzed in Section 6.1.1, we will not add further explanations.

5 Reviewer #293E

E.1: Reviewer Comment: *In production systems, priority queues are scarce resource. Using only two HW queues for one application will be a luxury. How does PTP perform with 2 prio queues (one high, one low) compared to existing production solutions like Swift, HPCC, DCTCP?*

Response E.1: Thanks for the reviewer's comments. Actually, in our paper, we have already evaluated the performance of PPT using only two queues. Please see Fig. 19 for more details.

Proposed modification: Given our shown result, we will not add further explanations in the paper.

E.2: Reviewer Comment: *Even the 'large-scale' simulations cover only 144 servers w/ 40 100G as core link speeds. This resembles DC topology of more a decade ago. With the trend of AI clusters growing to have 10s of Ks of GPUs, I believe at least 2K or more endpoints must be simulated to see the real impact on shallow switch.*

Response E.2 Thanks for the reviewer's comments. Our current topology is commonly used in datacenter networking community. Simulation with 2K-node scale takes significant time, which may miss the camera-ready deadline. So, we plan to do it in the future.

Proposed modification: Due to time constraints, we will not add further experiment in the paper.

E.3: Reviewer Comment: *Fairness and TCP friendliness discussion are missing: tail FCT of large flows can tell the impact on fairness but is completely missing.*

Response E.2 PPT's HCP is exactly DCTCP. So, PPT is essentially a TCP-friendly design. Regarding the fairness issue, frankly speaking, we have not taken it into account. We plan to discuss it in Appendix A.

Proposed modification: We have added an analysis in the footnote on page 6.