Review #293A

Overall merit

2. Weak reject

Paper summary

PPT offers a transport protocol that can run a low-priority control loop to harvest the available bandwidth left by the high-priority control loop, as well as gradually distinguish small flows from long flows and use this distinction to optimize the FCT of small flows.

Reasons to accept

- The paper offers a strong implementation part

Reasons not to accept

- The goals of the paper are not clearly defined, and it is not clear how to measure success
- The paper's contributions vs a simple priority-queueing scheme could be better explained

Comments for authors

Thanks for submitting the paper to ACM SIGCOMM.

I like the implementation part of the paper. A PPT prototype is implemented as an extension to the Linux kernel. This makes the claim that PPT can be readily deployable more credible. In addition, the CloudLab testbed experiment is also a nice feature of the paper. Finally, the comment about the issues in the Homa-Linux implementation is a great one and strengthens a point of the paper.

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.

Assuming such a goal, the assumptions of the paper could also be better presented. To reach this goal, a natural scheme is to use strict priority queueing at each switch for the four classes of traffic, with a separate queue for each class. Then, maybe run DCTCP for each flow given the ECN congestion signals for its queue. PPT seems indeed to use such priority queueing. However:

- 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 \lambda). 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.

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.

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?

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?

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?

Review #293B

Overall merit

4. Accept

Paper summary

Reactive transports underutilize during ramp up, while proactive transports require wholesale changes making them difficult to deploy or spend initial RTT for setup which hurts small flows. This paper proposes a new transport mechanism (PPT) that augments reactive transports (DCTCP specifically) with an approach to gracefully use any spare bandwidth as it appears.

The key idea is to add a low-priority control loop (referred to as LCP in the paper) to continue sending opportunistic packets at low priority. PPT triggers LCP based on heuristics which detect when the high-priority control loop (referred as HCP and is the same as DCTCP) might be underutilizing the network. LCP's control loop is tuned to balance aggressiveness and underutilization. With just the HCP+LCP approach, it is possible that small flows might be stuck behind large flows. To deal with this, PPT adds a new heuristic to classify flows as small or large based on TCP send buffer usage; the heuristic falls back

to PIAS. A prototype implementation in the Linux kernel shows significant performance improvements compared to other schemes.

Reasons to accept

- Effective and practical design that tries to keep the simplicity of reactive transports but match the performance of proactive transports.
- Impressive implementation and evaluations show significant performance gains over the state-of-the-art.
- Valuable insights on detecting short flows and improving the FCT.

Reasons not to accept

- Need more details on how PPT avoids underutilization and unfairness in certain cases (details in comments).

Comments for authors

Thanks for the well-written submission addressing an important problem with a practical solution. My comments are mainly focused towards further clarifications to improve the paper.

A few clarifications on Intermittent Loop Initialization (Section 3.1):

- 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.
- I wonder if tracking $\alpha min \alpha_{min}$ 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 transient congestion causes $\alpha min \alpha_{min}$ 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.
- 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 $Wmax W_{max}$ (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.

Impressive implementation and evaluation to show the performance benefits of PPT.

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.

What are the overheads of PPT?

Review #293C

Overall merit

3. Weak accept

Paper summary

PPT is a pragmatic transport that combines the advantages of reactive and proactive transport. It utilizes spare bandwidth efficiently by running a low-priority control loop in parallel with the normal control loop. PPT also employs an intermittent loop initialization mechanism and an exponential window decrease strategy to ensure graceful utilization of spare bandwidth. To optimize the average flow completion time of small flows, it uses a buffer-aware flow scheduling scheme that identifies large flows and assigns priorities accordingly. Performance evaluation shows that PPT outperforms existing transports in reducing overall average flow completion time, including small flows, while maintaining deployability.

Reasons to accept

- The evaluation demonstrates notable enhancements in FCT reduction in relation to Homa and DCTCP.
- The paper presents a robust implementation and evaluation.

- Although two primary concepts were not novel, combining them effectively in a datacenter environment necessitated several innovative approaches.

Reasons not to accept

- A single TCP session using multiple QoS classes/queues in a data center isn't easy to deploy if a datacenter already uses QoS classes for its business priority.

Comments for authors

Thank you for submitting your paper, which addresses a complex problem by skillfully integrating a range of techniques, both novel and existing.

As I understand it, the paper introduces two fundamental concepts: separating the control loop and using multiple priorities to prioritize small flows without knowing their size, similar to RC3 and PIAS, respectively. While there are differences, it would be beneficial to acknowledge that prior work has addressed similar or identical ideas and to give intuition on why the prior work alone is insufficient. Notably, RC3 was designed with WANs in mind, utilizing loss-based congestion control, which results in an aggressive approach that is not well-suited for data centers.

I would like to see microbenchmarks utilizing a single bottleneck and synthetic workload to assess the exploitation of spare bandwidth. The research highlights two scenarios where LCP proves beneficial: during slow start and following intense congestion. The slow start scenario makes sense, explaining the efficacy of small flow RTTs. However, the severe congestion scenario is less clear. Theoretically, DCTCP should prevent most congestion unless the workload involves synchronized bursts, which doesn't seem likely in data mining or web search workloads.

One reason for spare bandwidth in a network is when a group of flows depart, allowing the remaining flows to increase. LCP doesn't appear to be helpful in this situation, as W_max from the last RTT would be equal to W and would increase as HCP receives Ack packets, leaving no space for LCP. In a steady state, we expect DCTCP to saturate the link, resulting in no spare bandwidth.

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.

Does it use 8 queues in the evaluation? It mostly mentions high-priority and low priority and I am curious how many queues were used, and if more queues help.

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?

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?

Review #293D

Overall merit

3. Weak accept

Paper summary

The paper posits that "proactive" congestion control algorithms (e.g., Homa) deliver good performance but are hard to deploy and reactive ones (e.g., DCTCP) are easy to deploy but have poor performance. It tries to strike a balance between performance and ease of deployment but proposes a system that works in conjunction with DCTCP and helps it close its performance gap with more performant algorithms such as Homa (and in fact, helps it outperform tham in some cases). It leverages two parallel control loops, one high-priority one (HCP) which sends normal DCTCP packets and a low-priority one (LCP) that opportunistically sends packets to fill the bandwidth.

Reasons to accept

- As an intellectual exercise, it is intriguing to see if/how DCTCP's performance can be improved.
- Good performance results

Reasons not to accept

- The proposed technique is tied to DCTCP. There are newer algorithms in the same "reactive" category that are demonstrated to have better performance and be relatively easy to deploy.

Comments for authors

Thanks for submitting this paper to Sigcomm! It was an interesting read. It was unexpected to me that a modified version of DCTCP can achieve around 30% lower mean flow completion times than Homa.

My main issue with the paper is about the premise that DCTCP is a more practical congestion control than some of the other existing algorithms with better performance. Take Swift from Google [SIGCOMM'20] as an example. One might argue that since it does not rely on ECNs -- and subsequently avoids all the ECN-related issues such as sensitivity to setting it correctly -- is a simpler technique to deploy compared to DCTCP. It has both been deployed and shown to have a better performance than GCN (Google's version of DCTCP-style congestion control) -- sometimes dramatically so.

It is questionable if DCTCP + PPT (which, in turn, relies on relatively complex operations, e.g., to identify large flows and schedule them

appropriately) is easier to deploy than alternative congestion control algorithms.

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.

"[PPT's] design can be integrated with delay-based transport." -- It would be helpful to see how this can be done and, importantly, how much benefit one would see from combining recent delay-based protocols and PPT.

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 [21])." What is the actual algorithm?

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?

Review #293E

Overall merit

2. Weak reject

Paper summary

PPT is a congestion control mechanism (not really a full transport handling reliability and flow ctrl) which fills up the available BW unused by TCP saw-tooth window/rate control (AIMD and slow-start), by opportunistically injecting low-priority flows transmitted at the rate complementing the rate of the main high-priority TCP flows. This way, the low latency transfer of short flows and high-BW transfer of large flows are achieved while keeping the overall injection rate and queue depth stable.

Reasons to accept

- it is an interesting follow-up of a prior work RC3 which first introduced a dual-loop rate control with low-priority loop complementing the main loop. Compared to RC3, PTP better handles modern AI workloads by using ECN and more sensitive window control and scheduling of low-priority flows.

- testbed and simulation experiments are thorough in terms of covering most of modern CC algorithms

Reasons not to accept

- key parameters and assumptions are missing or misleading.
- simulation set up is also misleading.

Comments for authors

(will massage the writing later)

Missing key parameter and assumption

- How do you set the key parameter -- # of last RTTs to measure the Maximum Window?
- the threshold at the TCP send buffer is application specific. Are you assuming the type of application and flow size distribution is known a priori?

Evaluation

- 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?

- 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 buffers, especially when compared against INT-based HPCC.
- fairness and TCP friendliness discussion are missing: tail FCT of large flows can tell the impact on fairness but is completely missing

Thanks for submitting to SIGCOMM. The paper was discussed at the PC meeting and accepted. Congratulations! Summarizing the reviews and discussion, the PC appreciated:

- strong implementation in the Linux kernel which seems practical.
- performance improvements seem significant.

However, there were several concerns raised leading to a heavy debate during the discussions:

- 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.
- there's lacking discussion on other related work such as PCC Proteus.
- the paper needs to discuss more insights on how the system is able to deliver the performance benefits.
- other claims such as "integration with other transports is easy" need to be further justified.
- missing details on evaluation setup and parameters.

Given these top-level concerns and the other points raised in the reviews, the PC felt that this paper needs *heavy* shepherding to address these points. Please work closely with the shepherd on these, and we hope to see a stronger camera-ready version.