Toward Greater Scavenger Congestion Control Deployment: Implementations and Interfaces

Tong Meng[1] Christopher Cai[1] Brighten Godfrey[1] Michael Schapira[2]
[1]UIUC [2]Hebrew University of Jerusalem

ABSTRACT

Heterogeneous applications and restricted bandwidth on the Internet have motivated recent works on scavenger congestion control, which yields bandwidth to competing primary traffic for increased network-wide utility. Although potential use cases are quite common, deployments are as yet limited, in part due to protocol design immaturity, lack of open source code, and limited experimental evaluation. In this work, we extend recent scavenger advances by providing (1) open-source implementations of two recent scavenger proposals, PCC Proteus (QUIC-based) and LEDBAT++; (2) early benchmarks of the two in a realistic network setup; and (3) a discussion of APIs needed for applications to take advantage of scavenger congestion control. Ultimately, we hope this line of work will lead to further community discussion, open development, and deployment of scavengers yielding better quality of experience for users.

CCS CONCEPTS

• **Networks → Transport protocols**.

KEYWORDS

Congestion Control; Scavenger

1 INTRODUCTION

Internet congestion control traditionally strives for fairness among competing flows. However, because application needs are heterogeneous, non-equal bandwidth allocation can improve overall network-wide utility. For example, in a home with a limited-bandwidth Internet connection, a video streaming flow may deliver degraded video quality when a software update runs in the background. Since the user does not require the software update to finish immediately (and may not even realize it is happening), it would be preferable for the update to yield and allow the video stream to utilize more bandwidth at that moment. This is the goal of a congestion control scavenger [12, 19, 19] – enabling a flow with elastic timing requirement to yield to more time-sensitive competing traffic. Moreover, as demonstrated in [12, 14], even for time-sensitive applications, network-wide Quality of Experience (QoE) can be enhanced if the application adaptively lowers its priority when it can be done safely (*e.g.*, when a video streaming flow's throughput is higher than needed to serve the highest video bitrate smoothly). While having many natural use cases, compared with primary protocols, the deployment of scavengers thus far remains sparse. To the best of our knowledge, the most significant examples are the use of LEDBAT [19] and LEDBAT++ [16] within the Windows Server operating system, and the use of LEDBAT by μTorrent [1] and Apple devices (for software updates) [2].

Several factors that limit the widespread adoption of scavengers. First, current scavengers exhibit suboptimal performance. An early scavenger proposal, LEDBAT [19], is designed to limit the induced queuing delay below a target. However, it suffers from a latecomer advantage with homogeneous LEDBAT flows [17], and is more aggressive than latency-aware primary protocols such as BBR [7] and PCC [9, 12]. LEDBAT++ [16], an improved version of LEDBAT, is yet to be evaluated comprehensively. Our own scavenger congestion control proposal, PCC Proteus [12], adopts the utility framework from past research on PCC [8, 9], and uses a dedicated scavenger utility function that leverages latency deviation as a sensitive early signal of flow competition. Although the Proteus scavenger (Proteus-S) addresses LEDBAT's main deficiencies, its performance under noisy wireless environments needs to be improved [12].

Furthermore, available implementations of scavenger congestion controllers are limited: LEDBAT and Proteus have open source implementations only in non-standard datapaths (in μTorrent [1] and a UDT variant [4], respectively). LEDBAT++ has no open-source codebase, being implemented only in Windows Server. Thus, researchers cannot conveniently evaluate the performance of scavenger protocols in real-world conditions. It is even harder to explore scenarios where the priority of a flow is *adaptively* chosen while the flow is in progress [12, 14], since existing congestion control interfaces (like the Linux kernel and CCP [13]) cannot convey the priority selection to transport datapath.

In this talk, we report on several advances in scavenger

ANRW '21, July 24–30, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8618-0/21/07.
https://doi.org/10.1145/3472305.3472323

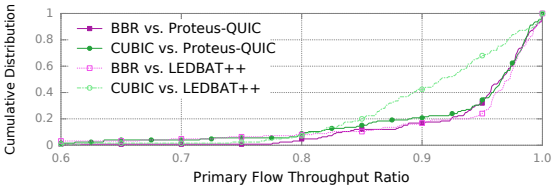
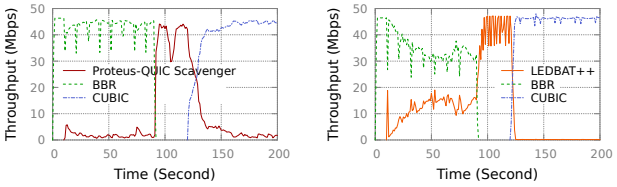


Figure 1: Primary flow throughput ratio



(a) Proteus-QUIC (b) LEDBAT++

Figure 2: Throughput across time

congestion control. We implemented Proteus within the widely-deployed QUIC datapath [11]. We also developed a LEDBAT++ implementation and are releasing it as open source code (this is the first such implementation of LEDBAT++, to the best of our knowledge). We present preliminary benchmarks of the two implementations. Finally, as scavengers can benefit from application-layer guidance, we discuss the congestion control interface necessary for scavenger usage by diverse Internet applications.

2 SCAVENGER IMPLEMENTATIONS

In [12], we implemented Proteus [4] in user-space on top of UDT [10]. Since UDT is not a popular transport datapath, we have now ported it to QUIC. The code [5] is open-sourced.

Next, we implemented LEDBAT++ [16]. LEDBAT++ introduces several enhancements to LEDBAT, such as better sampling of queuing delay to mitigate the latecomer advantage. Our open-source implementation [3] branches from the μTorrent Transport Library [1].

To test these implementations, we emulated network bottlenecks in Mahimahi [15]. First, we let our Proteus-QUIC scavenger and LEDBAT++ compete against CUBIC and BBR, respectively, on a wide range of bottlenecks: bandwidth in {12, 24, 48, 96, 192} Mbps, RTT in {10, 30, 60, 100, 200} ms, buffer size in {0.2, 0.5, 1, 2, 5} BDP. We tested all 125 combinations of those parameter settings. Fig. 1 presents the CDF of primary flow throughput ratio, computed as primary flow throughput when competing with scavenger divided by that when running alone. Compared with LEDBAT's performance in [12], LEDBAT++ does better: when competing with BBR, it can yield similarly well as our Proteus-QUIC scavenger, although with a worse tail (*e.g.*, it lowers BBR's throughput by at least 20% in 8% of cases, 2.5× as often as the Proteus-QUIC scavenger). With CUBIC as the primary flow, Proteus-QUIC yields better than LEDBAT++, giving CUBIC a median of 5.6% higher throughput. That is fundamentally due to LEDBAT++'s reliance on a target delay, which may inevitably be relatively more aggressively under specific bottlenecks.

To see performance effects in more detail, we pick a representative 48 Mbps Mahimahi bottleneck with 30 ms RTT and 4 BDP buffer, and let a scavenger flow compete with a BBR and a CUBIC flow successively across time. Fig. 2(a) shows that Proteus-S yields to both BBR and CUBIC. As in Fig. 2(b), LEDBAT++ fails to yield most of its bandwidth to BBR, but backs off faster than Proteus-S when the CUBIC flow starts. Thus, our QUIC implementation of Proteus maintains the scavenging gains claimed in [12], but further improvements (*e.g.*, convergence speed) are desirable.

3 CONGESTION CONTROL INTERFACES

CCP [13] and CONMax [6] demonstrate the importance of establishing unified interfaces for congestion control protocols. Existing congestion control interfaces on most transport datapaths are oblivious to application requirements. This layered network stack [18] is a design choice intended to accommodate plugging a congestion control protocol into any application once implemented, without application developers bothering with lower layers. However, this restricts each flow to a static congestion control mode, e.g., primary/scavenger, for the flow's entire lifetime, and cannot accommodate Proteus's ability to adaptively switch between primary and scavenger priorities for optimized network-wide utility. We are thus inspired to envision an interface such that the application can convey the selected priority to congestion controller.

Implementation Approaches. To support the above interface, the transport datapath needs to incorporate a callback that facilitates the switch of congestion control priorities (*e.g.*, `OnPrioritySwitch(.)`). For Proteus, it suffices to toggles a `boolean` flag (*e.g.*, `is_scavenger_prior`) to change the utility function in use. If primary and scavenger protocols are implemented separately (*e.g.*, as separate classes in QUIC), the callback can be used to initiate a connection migration (*e.g.*, `OnConnectionMigration(.)` method in QUIC), which effectively starts a new connection with a saved state including the congestion window. Otherwise, the application can open two parallel connections per flow, representing primary and scavenger priorities, respectively. Then, the callback can enforce a congestion window of 0 for a certain connection, and migrate the current congestion window to another connection with corresponding priority (which can be combined with multipath protocols like MPTCP [20]).

Acknowledgement. This work was supported by National Science Foundation Award 2008971.

REFERENCES

[1] 2018. μTorrent Transport Protocol library. http://github.com/bittorrent/libutp.

[2] 2019. LEDBAT Implementation by Apple. https://opensource.apple.com/source/xnu/xnu-1699.22.81/bsd/netinet/tcp_ledbat.c.

[3] 2020. LEDBAT++ Implementation based on libutp. https://github.com/meng-tong/libutp.

[4] 2020. PCC Proteus Implementation based on UDT. https://github.com/PCCproject/PCC-Uspace.

[5] 2021. PCC Proteus Implementation in QUIC. https://github.com/netarch/PCC_QUIC.

[6] H. Ballani and P. Francis. 2007. CONMan, a step towards network manageability. *Proc. of ACM SIGCOMM* (August 2007).

[7] N. Cardwell, Y. Cheng, C.S. Gunn, S.H. Yeganeh, and Van Jacobson. 2016. BBR: Congestion-Based Congestion Control. *Queue* 14, 5 (2016), 50.

[8] M. Dong, Qingxi Li, Doron Zarchy, Philip Brighten Godfrey, and Michael Schapira. 2015. PCC: Re-architecting Congestion Control for Consistent High Performance. *Proc. of NSDI* (March 2015).

[9] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. 2018. PCC Vivace: Online-Learning Congestion Control. *Proc. of NSDI* (April 2018).

[10] Yunhong Gu. 2005. *UDT: a high performance data transport protocol.* University of Illinois at Chicago.

[11] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, et al. 2017. The QUIC Transport Protocol: Design and Internet-Scale Deployment. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 183–196.

[12] Tong Meng, Neta Rozen Schiff, P Brighten Godfrey, and Michael Schapira. 2020. PCC proteus: Scavenger transport and beyond. *Proc. of ACM SIGCOMM* (August 2020).

[13] Akshay Narayan, Frank Cangialosi, Deepti Raghavan, Prateesh Goyal, Srinivas Narayana, Radhika Mittal, Mohammad Alizadeh, and Hari Balakrishnan. 2018. Restructuring endpoint congestion control. *Proc. of ACM SIGCOMM* (August 2018).

[14] Vikram Nathan, Vibhaalakshmi Sivaraman, Ravichandra Addanki, Mehrdad Khani, Prateesh Goyal, and Mohammad Alizadeh. 2019. End-to-end transport for video QoE fairness. *Proc. of ACM SIGCOMM* (August 2019).

[15] R. Netravali, A. Sivaraman, S. Das, A. Goyal, K. Winstein, J. Mickens, and H. Balakrishnan. 2015. Mahimahi: Accurate Record-and-Replay for HTTP. *Proc. USENIX ATC* (August 2015).

[16] D. Havey P. Balasubramanian, O. Ertugay. 2020. LEDBAT++: Congestion Control for Background Traffic. https://tools.ietf.org/html/draft-irtf-iccrg-ledbat-plus-plus-01.

[17] Dario Rossi, Claudio Testa, Silvio Valenti, and Luca Muscariello. 2010. LEDBAT: the new BitTorrent congestion control protocol. *ICCCN* (August 2010).

[18] J. Saltzer, D. Reed, and D. Clark. 1981. End-to-End Arguments in System Design. *Proc. International Conference on Distributed Computing Systems (ICDCS)* (April 1981).

[19] S. Shalunov. 2009. Low Extra Delay Background Transport (LEDBAT). Draft. https://tools.ietf.org/pdf/draft-ietf-ledbat-congestion-00.pdf

[20] Damon Wischik, Costin Raiciu, Adam Greenhalgh, and Mark Handley. 2011. Design, Implementation and Evaluation of Congestion Control for Multipath TCP. *NSDI* (March 2011).