PolyCC: Poly-Algorithmic Congestion Control

Shuaipeng Zhu, Tong Li*
Renmin University of China

Xinyu Ma
Renmin University of China

Yinfeng Zhu, Taotao Zhang
ByteDance

Senzhen Liu
ByteDance

Haiyang Wang
University of Minnesota Duluth

Ke Xu
Tsinghua University

ABSTRACT

This paper demonstrates PolyCC, a general framework for simultaneous operation of poly-algorithmic congestion control. PolyCC gains benefits from taking advantage of the complementary among already existing congestion controllers.

CCS CONCEPTS

• **Networks → Transport protocols**.

KEYWORDS

QUIC, congestion control, complementary

ACM Reference Format:
Shuaipeng Zhu, Tong Li, Xinyu Ma, Yinfeng Zhu, Taotao Zhang, Senzhen Liu, Haiyang Wang, and Ke Xu. 2023. PolyCC: Poly-Algorithmic Congestion Control. In *ACM SIGCOMM 2023 Conference (ACM SIGCOMM '23), September 10, 2023, New York, NY, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3603269.3610846

1 INTRODUCTION

For over 40 years, many different congestion control algorithms (CCAs) have been developed for specific environments, including over 15 CCAs in the Linux kernel alone [1]. However, no single CCA can adequately prevail across all environments [2, 3]. To satisfy the increasingly diverse application requirements over highly complex network conditions, learning-based CCAs have gained much attraction recently [4–8]. However, their exploration-based models may make mistakes or dangerous actions, resulting in poor performance.

On the other hand, different CCAs complement each other in achieving high throughput, low delay, or fast convergence (see §3). Recent studies have shown that leveraging the complementarity among existing CCAs holds the potential to consistently achieve high performance across different environments [1, 9–13]. However, their implementations concentrate on resolving specific problems or rely on specific system capabilities, lacking a general framework for the simultaneous operation of multiple CCAs.

In this paper, we present PolyCC, a collaborative framework that combines multiple CCAs to leverage their complementary strengths and overcome their limitations. Essentially, PolyCC brings dynamic cooperation among multiple CCAs, which forms a "jack-of-all-trades" team, resulting in a robust solution that can handle different environments. PolyCC follows three principles: (i) Reusing the already existing CCAs in the protocol stack, (ii) allowing modular extensions for newly proposed CCAs, and (iii) supporting user-customized policy on how the filtered CCAs collaborate.

We implement PolyCC upon QUIC and showcase two representative scenarios: (i) Employing PolyCC to improve the goodput of BBR in the case of high degrees of aggregation and RTT variance (see §3.1), and (ii) employing PolyCC to accelerate the convergence of BBR in the case of bandwidth change (see §3.2). For both scenarios, PolyCC has significant improvements in network performance.

*Tong Li is the corresponding author (tong.li@ruc.edu.cn).

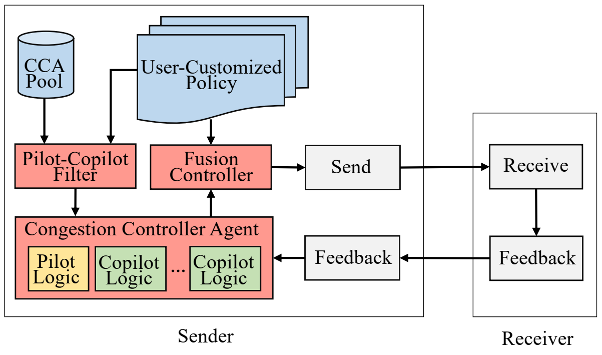


Figure 1: The PolyCC architecture.

2 DESIGN

Figure 1 illustrates the architecture of PolyCC, which contains sender-side components as we will describe below.

CCA Pool. PolyCC reuses the already existing implementations of CCAs in the protocol stack to build the CCA Pool. Besides, a newly proposed CCA, including the learning-based CCA, can also be added to the CCA Pool. In a way, the arts [1, 11–13], adopting a traditional CCA to cooperate with a learning-based CCA, have offered typical instances of the PolyCC framework implementation.

User-Customized Policy. Users can customize the policy on how CCAs collaborate. Accordingly, the Pilot-Copilot Filter answers the question of which CCA should be selected as the pilot or copilot. The policy also provides a fusion function for the Fusion Controller to compute the calibrated window/rate for the pilot CCA.

Pilot-Copilot Filter. This component selects only one CCA as the pilot and selects one or more CCAs as the copilots. The single-pilot-multiple-copilots design ensures a sender's deterministic action, regardless of the number of CCAs running simultaneously.

Congestion Controller Agent. By default, the protocol stack (e.g., QUIC) can only run one CCA at a time. Acting as the middleware, the Congestion Controller Agent enables the protocol stack to run multiple CCAs simultaneously, each of which reacts to the receiver's

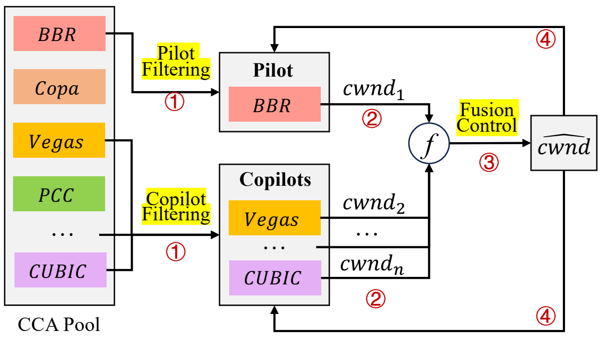


Figure 2: An example of the PolyCC workflow.

feedback (i.e., ACK) and computes the congestion window ($cwnd$) or pacing rate ($\propto \frac{cwnd}{RTT}$) independently according to its own logic. **Fusion Controller.** Considering the computed windows/rates of all copilots, the Fusion Controller computes a calibrated window/rate for the pilot CCA according to a fusion function defined by the User-Customized Policy. Based on this calibrated window/rate, the sender updates the windows/rates of all CCAs and then conducts congestion control based on the logic of the pilot CCA.

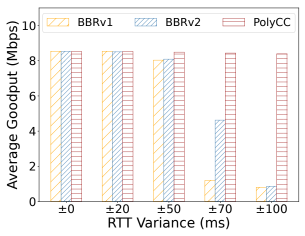
We further give an example of the PolyCC workflow in Figure 2. **Step 1**: Given the CCA Pool containing CCAs such as BBR, Copa, Vegas, PCC, and CUBIC, assume that according to the user-customized policy, the Pilot-Copilot Filter selects BBR as the pilot, and selects Vegas and CUBIC as the copilots. **Step 2**: The Congestion Controller Agent runs BBR, Vegas, and CUBIC simultaneously and computes their congestion windows (i.e., $cwnd_1, cwnd_2, ..., cwnd_n$) independently. **Step 3**: The Fusion Controller computes the $\widehat{cwnd}$ according to the fusion function ($f()$), i.e., $\widehat{cwnd} = f(cwnd_1, cwnd_2, ..., cwnd_n)$. Note that $f()$ is customizable, it can set as $\max()$, $\min()$, or $\widehat{cwnd} = \arg\min |cwnd_i - cwnd^*|, i \in [1, n]$, where $cwnd^*$ is the target cwnd. **Step 4**: The sender updates the windows of all CCAs with $\widehat{cwnd}$, and then conducts congestion control according to BBR.

3 DEMONSTRATION

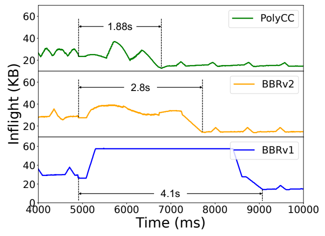
A demo video is at https://youtu.be/eQ4llFq5xZs. We implement the PolyCC framework based on the Chromium QUIC (gQUIC). To verify the feasibility of PolyCC, we set up a testbed running QUIC client/server on two hosts, with a network emulator forwarding. The CCA Pool currently contains some popular CCAs such as Reno, CUBIC, BBRv1, BBRv2, PCC, and Copa.

3.1 Scenario 1: Goodput Improvement

Issues and challenges. Extremely common in shared media (e.g., WiFi, cellular, and cable modems), or in interrupt processing (e.g., TSO, GSO, LRO, GRO), *aggregation effects* result in high degrees of RTT variance (jitter) in the Internet [14, 15]. For example, the RTT samples can range from 4 ms to 80 ms over the Wi-Fi links [16]. However, if ACKs are excessively delayed (i.e., $> 2RTT$) then BBRv1 [17] stops sending after sending $2BDP$, leaving the bottleneck idle for potentially long periods. Note that window-based CCAs (e.g., CUBIC) do not have this issue because they continue increasing the window after bursts of ACKs, growing the window until the buffer is full. This greatly motivates the use of PolyCC.



(a) Scenario 1



(b) Scenario 2

Figure 3: Testbed Results.

Results. In this demonstration, BBR acts as the pilot and CUBIC acts as the copilot. The fusion function is set $\widehat{cwnd} = \max\{cwnd_{bbr}, cwnd_{cubic}\}$. We vary the RTT deviation with a mean value of 100ms. For example, an RTT deviation of ±70ms means the RTT variance ranges from 30ms to 170ms. We compare PolyCC with BBRv1 and BBRv2 (which has applied a heuristic way to deal with aggregation effects [16]). Figure 3 (a) shows the results. Simple but useful, PolyCC outperforms the BBR variants by keeping consistent high goodput even under fierce RTT variance. Specifically, PolyCC performs similarly to the BBR variants under slight RTT variance (e.g., ±20ms ($RTT \in [80, 120]$ms): ACK delays $< 2RTT_{min}$). However, when the RTT variance becomes fierce (e.g., ±70ms, ±100ms), PolyCC achieves 1.82 to 10.35 times of goodput than BBR variants.

3.2 Scenario 2: Convergence Improvement

Issues and challenges. It has been disclosed that BBR spends 2 to 4 seconds to converge when the available bandwidth halves. This is because BBR estimates the maximum bandwidth using a windowed max-filtered value of the delivery rates [17]. In contrast, Copa's rate control reacts to each arriving ACK and is able to converge rapidly in less than 1 second. In this demonstration, we explore the feasibility of the fusion between BBR and Copa via PolyCC.

Results. BBR acts as the pilot, and Copa acts as the copilot. The fusion function is set $\widehat{r} = \min\{r_{bbr}, r_{copa}\}$, where r is the pacing rate. At $t = 0$s, we start a flow with $bandwidth = 20$Mbps and $RTT = 100$ms. At $t = 5$s, we halve the bandwidth to 10Mbps. Figure 3 (b) shows the convergence procedure of inflight bytes. It is demonstrated that by applying PolyCC, the convergence time is reduced by 54.1% under bandwidth change than BBRv1, and reduced by 32.8% than BBRv2.

4 CONCLUSION AND FUTURE WORK

PolyCC provides a general framework for enabling poly-algorithmic congestion control. Future work will focus on (i) the reproduction of the state-of-the-art CCAs [1, 11–13] based on the PolyCC framework, (ii) providing open PolyCC APIs for augmenting the CCA Pool, (iii) building a community evaluation platform for in-depth research on fusion policies, and (iv) exploring the possibility of zero-configuration PolyCC.

ACKNOWLEDGMENTS

This work is supported by the fund from ByteDance, the fund for building world-class universities (disciplines) of Renmin University of China, the NSFC Projects (No. 62202473 and No. 61932016), the China National Funds for Distinguished Young Scientists (No. 61825204), and the Beijing Outstanding Young Scientist Program (No. BJJWZYJH01201910003011).

REFERENCES

[1] Soheil Abbasloo, Chenyu Yen, and H Jonathan Chao. Classic meets modern: A pragmatic learning-based congestion control for the internet. In *SIGCOMM*, 2020.

[2] Michael Schapira and Keith Winstein. Congestion-control throwdown. In *ACM Hotnets*, pages 122–128, 2017.

[3] Bo Wu, Tong Li, Cheng Luo, Changkui Ouyang, Xinle Du, and Fuyu Wang. Autoplex: inter-session multiplexing congestion control for large-scale live video services. In *SIGCOMM Workshop (NAI)*, pages 1–6, 2022.

[4] Keith Winstein and Hari Balakrishnan. Tcp ex machina: Computer-generated congestion control. *ACM SIGCOMM Computer Communication Review*, 43(4):123–134, 2013.

[5] Mo Dong, Qingxi Li, Doron Zarchy, P Brighten Godfrey, and Michael Schapira. PCC: Re-architecting congestion control for consistent high performance. In *USENIX NSDI*, pages 395–408, 2015.

[6] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. {PCC} vivace: Online-learning congestion control. In *USENIX NSDI*, pages 343–356, 2018.

[7] Francis Y Yan, Jestin Ma, Greg D Hill, Deepti Raghavan, Riad S Wahby, Philip Levis, and Keith Winstein. Pantheon: the training ground for internet congestion-control research. In *USENIX ATC*, pages 731–743, 2018.

[8] Zhiren Zhong, Wei Wang, Yiyang Shao, Zhenyu Li, Heng Pan, Hongtao Guan, Gareth Tyson, Gaogang Xie, and Kai Zheng. Muses: Enabling lightweight learning-based congestion control for mobile devices. In *INFOCOM*, page 2208–2217, 2022.

[9] Jianer Zhou, Xinyi Qiu, Zhenyu Li, Gareth Tyson, Qing Li, Jingpu Duan, and Yi Wang. Antelope: A framework for dynamic selection of congestion control algorithms. In *ICNP*, pages 1–11, 2021.

[10] Jia Zhang, Enhuan Dong, Zili Meng, Yuan Yang, Mingwei Xu, Sijie Yang, Miao Zhang, and Yang Yue. Wisetrans: Adaptive transport protocol selection for mobile web service. In *WWW*, pages 284–294, 2021.

[11] Zhuoxuan Du, Jiaqi Zheng, Hebin Yu, Lingtao Kong, and Guihai Chen. A unified congestion control framework for diverse application preferences and network conditions. In *CoNext*, pages 282–296, 2021.

[12] Zhiyuan Pan, Jianer Zhou, Xinyi Qie, Weichao Li, Heng Pan, and Wei Zhang. Marten: A built-in security drl-based congestion control framework by polishing the expert. In *INFOCOM*, pages 1–10, 2023.

[13] Wenzheng Yang, Yan Liu, Chen Tian, Junchen Jiang, and Lingfeng Guo. Gemini: Divide-and-conquer for practical learning-based internet congestion control. In *INFOCOM*, pages 1–10, 2023.

[14] Tong Li, Kai Zheng, Ke Xu, Rahul Arvind Jadhav, Tao Xiong, Keith Winstein, and Kun Tan. Tack: Improving wireless transport performance by taming acknowledgments. In *ACM SIGCOMM*, pages 15–30, 2020.

[15] Tong Li, Kai Zheng, and Ke Xu. Acknowledgment on demand for transport control. *IEEE Internet Computing*, 25(2):109–115, 2021.

[16] BBR IETF 101. https://datatracker.ietf.org/meeting/101/materials/slides-101-iccrg-an-update-on-bbr-work-at/-google-00, 2018.

[17] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. BBR: Congestion-based congestion control. *Queue*, 14(5):20–53, 2016.