Fair Queuing Aware Congestion Control

Maximilian Bachl

https://github.com/muxamilian/fair-queuing-aware-congestion-control

***Abstract*—Fair queuing is becoming increasingly prevalent in the internet and has been shown to improve performance in many circumstances. Performance could be improved even more if endpoints could detect the presence of fair queuing on a certain path and adjust their congestion control accordingly. If fair queuing is detected, the congestion control would not have to take cross traffic into account, which allows for more flexibility. In this paper, we develop the first algorithm that continuously checks if fair queuing is present on a path. When fair queuing is detected, a different congestion control is chosen, which results in reduced latency. Unlike an algorithm proposed in a previous paper of us, the approach presented here does not only detect the presence of fair queuing once at flow startup but it does so continuously.**

I. Background

When different applications send packets on the internet, one application can send more than the other and thus take unfairly take a larger share of bandwidth. This can result in unfairness and bad user experience. Several different approaches have been proposed to address this [1], [2] : One is to make sure every network flow is "well-behaved" (also known as TCP friendly) Another one is to enforce fairness at switches and routers, called "fair queuing" or "flow queuing" [3].

While fair queuing was proposed decades ago, it only gained popularity in the last couple of years because of implementations in the Linux kernel [4], [5]. Applications can benefit from increasing deployment of fair queuing: It makes sure that not the most aggressive one wins. It would be even better if applications could know if the connection they're sending on is managed by fair queuing. Then they could be sure that the can use a common congestion control mechanism, while not bothering or being bothered by other network flows. We proposed the first such approach in previous work [6] but our previous approach had some shortcomings upon which we improve in this paper.

Our approach is a congestion control mechanism which also performs measurements to determine if there is fair queuing. This approach of performing measurements in congestion control became popular in the last couple of years and was already followed by [7]–[10].

II. Introduction

In our previous work we proposed a technique which determines the presence of fair queuing at flow startup [6], which worked as follows: If fair queuing is successfully detected at flow startup, a congestion control was used, which aimed to keep queuing delay low (delay-based congestion control). While this delay-based congestion control achieved high throughput and low delay, it was vulnerable to be outcompeted by other network flows sending more aggressively, such as [7], [8], [11], similar to the Vegas congestion control algorithm [12]. This means that our delay-based congestion control performed well but only when it wouldn't have to compete with other flows. Thus is only used if fair queuing is detected. If it is detected that there is no fair queuing, our approach uses a more aggressive congestion control (specifically PCC [8]), which can compete better with other network flows, but doesn't keep delay as low as our delay-based congestion control.

	10 Mbit/s	50 Mbit/s	100 Mbit/s
10ms	Median: 100% 1st quart.: 100% 3rd quart.: 100%	Median: 100% 1st quart.: 100% 3rd quart.: 100%	Median: 100% 1st quart.: 100% 3rd quart.: 100%
50ms	Median: 100% 1st quart.: 100% 3rd quart.: 100%	Median: 100% 1st quart.: 100% 3rd quart.: 100%	Median: 100% 1st quart.: 100% 3rd quart.: 100%
100ms	Median: 100% 1st quart.: 100% 3rd quart.: 100%	Median: 100% 1st quart.: 100% 3rd quart.: 100%	Median: 100% 1st quart.: 100% 3rd quart.: 100%

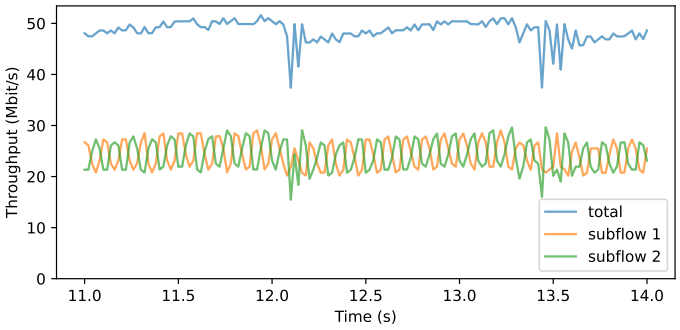
TABLE I: Detection accuracy in case there's **no fair queuing**. The overall median accuracy is 100%, the first quartile is 100% and the third quartile is 100%.

While our previous approach had high detection accuracy (98%), it also had some limitations:

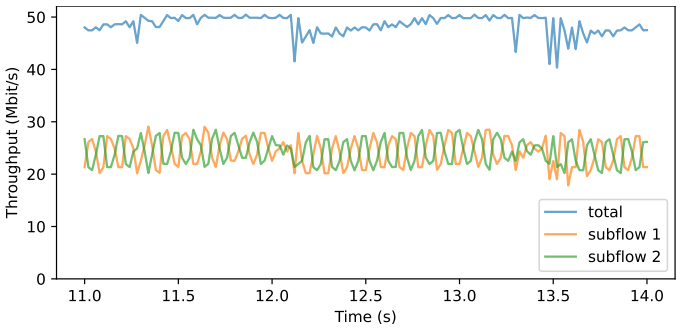
- It would **only** detect fair queuing at **flow startup**. But if the bottleneck link changes during a flow, it could be that the previous bottleneck had fair queuing while the new one doesn't. This wouldn't have been detected.
- It would detect fair queuing only after filling the queue at the bottleneck completely, **causing packet loss**.

We would thus like to have an approach which

- **continuously checks** for the presence or absence of fair queuing, not only at flow startup.
- **doesn't cause packet loss** while trying to determine if there is fair queuing or not.
- can be **transparently used** on top of any congestion control algorithm.



(a) Sending rate at the sender (fair queuing)



(b) Receiving rate at the receiver (fair queuing)

Fig. 1: Figure 1a shows the sending rate, 1b the receiving rate in case there's fair queuing. Around seconds 12 and 13.5, the sending rate reaches the maximum of the link – 50 Mbit/s – and fair queuing starts to limit the throughput of the flow that is sending more. Thus, in the lower figure, the dominant flow and the non-dominant flow achieve approx. the same receiving rate even though the dominant flow sends more (dominant flow). This is the effect of fair queuing.

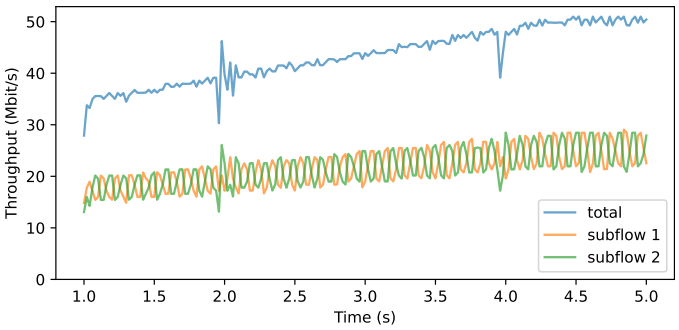
	10 Mbit/s	50 Mbit/s	100 Mbit/s
10ms	Median: 100% 1st quart.: 100% 3rd quart.: 100%	Median: 100% 1st quart.: 100% 3rd quart.: 100%	Median: 100% 1st quart.: 99% 3rd quart.: 100%
50ms	Median: 100% 1st quart.: 100% 3rd quart.: 100%	Median: 100% 1st quart.: 100% 3rd quart.: 100%	Median: 100% 1st quart.: 100% 3rd quart.: 100%
100ms	Median: 100% 1st quart.: 100% 3rd quart.: 100%	Median: 100% 1st quart.: 100% 3rd quart.: 100%	Median: 100% 1st quart.: 100% 3rd quart.: 100%

TABLE II: Detection accuracy in case there is **fair queuing**. The overall median accuracy is 100%, the first quartile is 100% and the third quartile is 100%.

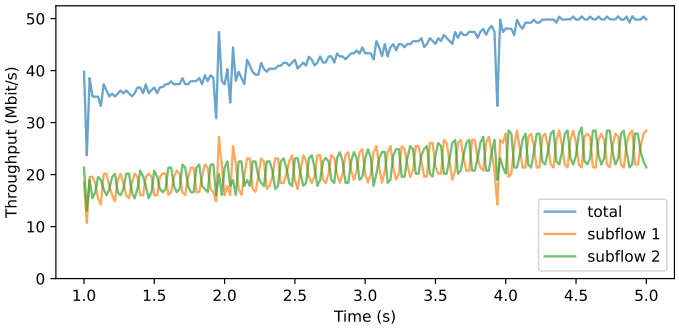
	Link utilization	Queuing delay
Newreno	95.6%	10.5 ms
Tonopah	94.3%	33.6 ms

TABLE III: Newreno's link utilization is 103% Tonopah's. But the **queuing delay** of Newreno is **321%** Tonopah's. This means that Tonopah can deliver the same throughput while cause a lot less queuing delay. The differences in **queuing delay** are **highly significant** (p-value[1] $< 10^{-5}$).

[1]Using Welch's t-test



(a) Sending rate at the sender (no fair queuing)



(b) Receiving rate at the receiver (no fair queuing)

Fig. 2: Figure 2a shows the sending rate, 2b the receiving rate in case there's no fair queuing. Even though the sender sends too much and a queue builds up, still the flow that sends more data also has a higher receiving rate. This is in contrast to Figure 1, where both flows have the same receiving rate once there is congestion at the bottleneck, thanks to fair queuing.

	10 Mbit/s	50 Mbit/s	100 Mbit/s
10ms	Median: 100% 1st quart.: 99% 3rd quart.: 100%	Median: 100% 1st quart.: 100% 3rd quart.: 100%	Median: 100% 1st quart.: 98% 3rd quart.: 99%
50ms	Median: 100% 1st quart.: 100% 3rd quart.: 100%	Median: 100% 1st quart.: 100% 3rd quart.: 100%	Median: 100% 1st quart.: 100% 3rd quart.: 100%
100ms	Median: 100% 1st quart.: 98% 3rd quart.: 100%	Median: 100% 1st quart.: 100% 3rd quart.: 100%	Median: 100% 1st quart.: 100% 3rd quart.: 100%

TABLE IV: Detection accuracy in case there's **no fair queuing** under the presence of **cross-traffic**. The overall median accuracy is 100%, the first quartile is 100% and the third quartile is 100%.

	10 Mbit/s	50 Mbit/s	100 Mbit/s
10ms	Median: 100% 1st quart.: 100% 3rd quart.: 100%	Median: 100% 1st quart.: 100% 3rd quart.: 100%	Median: 100% 1st quart.: 100% 3rd quart.: 100%
50ms	Median: 100% 1st quart.: 100% 3rd quart.: 100%	Median: 100% 1st quart.: 100% 3rd quart.: 100%	Median: 100% 1st quart.: 100% 3rd quart.: 100%
100ms	Median: 100% 1st quart.: 100% 3rd quart.: 100%	Median: 100% 1st quart.: 100% 3rd quart.: 100%	Median: 100% 1st quart.: 100% 3rd quart.: 100%

TABLE V: Detection accuracy in case there is **fair queuing** under the presence of **cross-traffic**. The overall median accuracy is 100%, the first quartile is 100% and the third quartile is 100%.

	10 Mbit/s	50 Mbit/s	100 Mbit/s
10ms	Median: 85% 1st quart.: 84% 3rd quart.: 85%	Median: 100% 1st quart.: 100% 3rd quart.: 100%	Median: 100% 1st quart.: 100% 3rd quart.: 100%
50ms	Median: 91% 1st quart.: 88% 3rd quart.: 92%	Median: 100% 1st quart.: 100% 3rd quart.: 100%	Median: 98% 1st quart.: 97% 3rd quart.: 99%
100ms	Median: 9% 1st quart.: 7% 3rd quart.: 13%	Median: 100% 1st quart.: 100% 3rd quart.: 100%	Median: 64% 1st quart.: 0% 3rd quart.: 75%

TABLE VI: Detection accuracy in case there is the ***fq_codel*** variant of **fair queuing**. The overall median accuracy is 95%, the first quartile is 83% and the third quartile is 100%.

III. Concept

IV. Evaluation

A. Throughput and delay of Tonopah

B. Cross-traffic

C. Other variants of fair queuing

V. Discussion

References

[1] L. Brown, G. Ananthanarayanan, E. Katz-Bassett, A. Krishnamurthy, S. Ratnasamy, M. Schapira, and S. Shenker, "On the Future of Congestion Control for the Public Internet," in *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*, (Virtual Event USA), pp. 30–37, ACM, Nov. 2020.

[2] R. Ware, M. K. Mukerjee, S. Seshan, and J. Sherry, "Beyond Jain's Fairness Index: Setting the Bar For The Deployment of Congestion Control Algorithms," in *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*, (Princeton NJ USA), pp. 17–24, ACM, Nov. 2019.

[3] J. Nagle, "On Packet Switches With Infinite Storage," Request for Comments RFC 970, Internet Engineering Task Force, Dec. 1985. Num Pages: 9.

[4] E. Dumazet, "pkt_sched: fq: Fair Queue packet scheduler [LWN. net]," 2013.

[5] T. Hoeiland-Joergensen, P. McKenney, D. Taht, J. Gettys, and E. Dumazet, "The flow queue codel packet scheduler and active queue management algorithm," tech. rep., 2018.

[6] M. Bachl, J. Fabini, and T. Zseby, "Detecting Fair Queuing for Better Congestion Control," Tech. Rep. arXiv:2010.08362, arXiv, Feb. 2021. arXiv:2010.08362 [cs] type: article.

[7] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-Based Congestion Control," *ACM Queue*, vol. 14, September-October, pp. 20 – 53, 2016.

[8] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira, "PCC: Re-architecting Congestion Control for Consistent High Performance," pp. 395–408, 2015.

[9] P. Goyal, A. Narayan, F. Cangialosi, S. Narayana, M. Alizadeh, and H. Balakrishnan, "Elasticity Detection: A Building Block for Internet Congestion Control," Tech. Rep. arXiv:1802.08730, arXiv, Feb. 2020. arXiv:1802.08730 [cs] type: article.

[10] D. A. Hayes, M. Welzl, S. Ferlin, D. Ros, and S. Islam, "Online Identification of Groups of Flows Sharing a Network Bottleneck," *IEEE/ACM Transactions on Networking*, vol. 28, no. 5, pp. 2229–2242, 2020. Publisher: IEEE.

[11] S. Ha, I. Rhee, and L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant," *ACM SIGOPS Operating Systems Review*, vol. 42, pp. 64–74, July 2008.

[12] L. Brakmo and L. Peterson, "TCP Vegas: end to end congestion avoidance on a global Internet," *IEEE Journal on Selected Areas in Communications*, vol. 13, pp. 1465–1480, Oct. 1995. Conference Name: IEEE Journal on Selected Areas in Communications.