Localizing Traffic Differentiation with TCP BBR

Martynas Rimkevičius

School of Computer and Communication Sciences

Semester Project

June 2024

Responsible
Prof. Katerina
Argyraki
EPFL / NAL

Supervisor
Zeinab Shmeis
EPFL / NAL

1 Introduction

Traffic differentiation (TD) is when ISPs apply different traffic management techniques to incoming traffic, based on its source and/or content. ISPs may prioritize certain traffic by sending them through faster-dedicated routes, or they may throttle them using shapers and policers. ISPs may deploy TD practices with good intentions, for example, to prevent video traffic from congesting their network [1]; or they may do so to serve a business need [2].

Transparency is important. Without knowledge of an ISP's TD policies, users lack the information necessary to make informed choices when selecting an internet service provider.

To guarantee transparency, Shmeis *et al.* [3] proposed a tool (WeHe$\mathbb{Y}$) to localize traffic differentiation to a particular ISP. In their approach, they test if the loss time-series observed by two flows sent simultaneously to the same destination is correlated. If it is, then the two flows must have traversed the same bottleneck queue inside the ISP's network. Hence, that ISP deplays throttling.

Up till now, WeHe$\mathbb{Y}$ was proven to work well with UDP traffic and TCP traffic that uses TCP CUBIC as the congestion control algorithm but not TCP BBR. WeHe$\mathbb{Y}$ was designed with TCP CUBIC because it is the most used by systems today; however, people are transitioning more to BBR because it makes better use of bandwidth and is less aggressive. Hence, we think it is valuable to evaluate WeHe$\mathbb{Y}$'s performance under BBR as well. While the two are designed to handle network congestion, they identify and react to congestion differently.

The goal of this project is to evaluate and improve the current localization method with TCP BBR. First, we will compare the performance metrics (e.g., loss and delay) of TCP flows when using BBR v.s. CUBIC. Then, we will show how loss trend correlation is affected when using TCP BBR; we study the correlation under two scenarios: (1) traffic traverses a shared policer, and (2) traffic traverses separate policers. Lastly, we will look if there is a better metric than loss to localize differentiation with TCP BBR.

2 Background

2.1 TCP CUBIC

TCP CUBIC is a loss-based TCP congestion control algorithm used as default in the Linux kernel since 2006 [4]. CUBIC uses packet loss to adjust its congestion window: first the window grows aggressively following a cubic function until it reaches saturation; then it slows down to probe the link of the bandwidth. CUBIC's window growth is independent of the RTT which guarantees fair bandwidth allocation among flows with different RTTs. CUBIC is designed to balance high data rates and maintain network stability.

2.2 TCP BBR

TCP BBR is a delay-based congestion control protocol designed by Google and was included in the Linux kernel in 2016 [5]. BBR uses packet's RTT estimate to build a model of the network's available bandwidth and latency. Based on this model, it dynamically adjusts the rate at which it sends data packets. Because BBR relies on RTT estimates, it can identify congestion events through changes in RTT. For example, when RTT suddenly increases, it could mean that there exists an overloaded deep buffer along the path (bufferbloat); therefore, TCP needs to slow down to avoid congestion events and excessive delay.

There are three versions of BBR released by Google. However, we focus on BBRv1 because it is included in the current Linux kernel (6.5.0-1020-gcp); hence, it can be easily integrated into practice.

2.3 Traffic throttling

Traffic throttling is when network operators use a rate limiter to limit the rate of traffic traversing their network to a pre-defined rate. It is implemented using a token bucket mechanism. The token bucket is defined by three parameters: the *rate*, which specifies how quickly the token bucket is refilled, the *burst size*, which sets the capacity of the bucket, and the *queue size*, which defines the limit before the queue checks for available tokens. When the token bucket is empty and a packet arrives, the rate limiter may drop it (produce loss) or put it in the queue (produce delay). Depending on the queue size, packets either experience more delay (if the queue is deep), or loss (if the queue is shallow). In this report, we focus more on the policing behavior of a rate-limiter which introduces more loss than delay to traffic.

2.4 Localization test through loss correlation

To localize traffic differentiation, WeHe𝕐 replays pre-recorded traffic traces from two servers it controls toward the same destination, such that the path from the two servers intersects only within the destination's network. In the meantime, WeHe𝕐 records all packets sent/received by the server through PCAP files. From the PCAP files, it computes the loss time-series as follows:

1. Using TCP retransmission, it detects packets that were lost.
2. Using packet's RTT it computes a list of interval sizes that varies from 10 to 50 minRTT where minRTT is the higher one of the minimum RTTs of two flows.

3. For each interval size, it computes the loss ratios and applies the Spearman correlation coefficient.
4. If for half of the intervals, the output is correlation, WeHe$\mathbb{Y}$ concludes that the two paths experience correlated loss and thus traversed the same bottleneck.

3 Setup

3.1 Network setup

Our network setup consists of one client and four servers: two measurement servers to replay our measurement traces, and two background servers that simply simulate some background traffic. The client runs on a Linux machine in the lab. We emulate policing at the client by configuring Linux's traffic control layer to identify and rate-limit our measurement and background traffic. For the measurement servers, we use the Google VMs that simulate the measured traffic using prerecorded traces of YouTube traffic of 25Mb/s concatenated together to make a Longtcp trace. The goal behind the Longtcp is to have traces long enough to have more than 30 intervals with a non-zero loss ratio which is important for later statistical tests. For the background servers, we use icnals servers in the IC cluster. Background servers simulate real-world network traffic by playing back pre-recorded data that closely resembles real traffic patterns. These preprocessed recordings, called CAIDA traces, are sent over sockets by the icnals servers.

To modify the TCP flavour of traffic sent from the server, we use the command `sysctl net.ipv4.tcp_congestion_control=[cubic or bbr]` since both tested TCP congestion control algorithms are already in the Linux kernel.

3.2 Parameters

In our experiment, we use 5 different parameters as shown in Table 1. The first three correspond to the policer configuration, the fourth represents the traffic setup (the TCP flavour of measurement traffic and background traffic), and the last is the policer type. In common policer, we sent both measurement traffic flows through the same queue. In non-common policer each measurement and background traffic along some background is sent to a separate queue.

We set the policer token bucket rates the same as in the Shmeis [3] paper. More specifically, we set the rate such that $\frac{\text{Input traffic}}{\text{rate}} = [2.5, 2, 1.5, 1.3]$. To compute the input traffic volume, we calculate the average rate of the CAIDA background traffic after sampling half of it (110Mb/s) and add the rate of the two LongTCP flows, this yields a total of 105Mb/s, which gives us the rates: $[40, 50, 70, 80]$Mb/s.

For the burst period, we take the minimum RTT measured from the client to the Google VMs rounded to the nearest 5ms. Using this burst period we calculate the token bucket burst which is rate $\times$ burst period $\times$ 125000Mb/B.

We set the queue size to 1. Using the queue size we get the queue limit which is calculated as follows: burst $\times$ queue size. In this report, we focus on policing and thus the queue size is fixed.

3.3 Setup verification

To verify whether our setup is correct we looked at the throughput of TCP BBR and TCP CUBIC flows using the same flavour as the background. Figure 1 shows the throughput achieved by one of the two measurement flows over an interval of 1 RTT, which means we would be looking at an approximate of the congestion window change. The key verifying feature of TCP BBR is its bandwidth bottleneck probing, where it injects more data into

Parameter	Values
Rate	[40, 50, 70, 80]Mb/s
Burst Period	0.02s
Queue Size	1
TCP scenario	BBR with CUBIC back, BBR with BBR back, CUBIC with CUBIC back, CUBIC with BBR back
Policer Type	Common, Non-common

Table 1: Network setup parameters

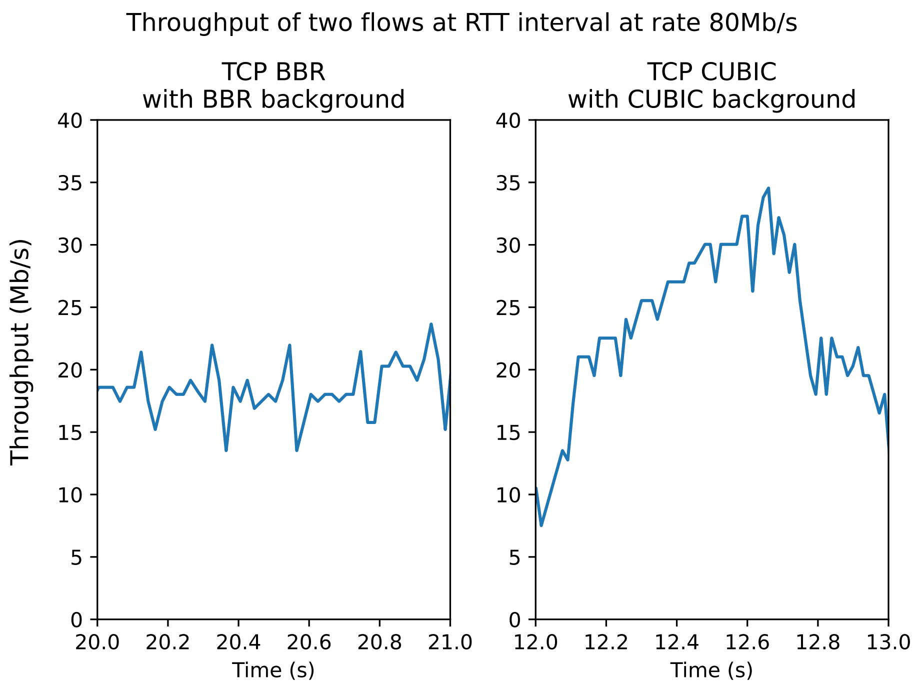

Figure 1: Throughput of TCP BBR and TCP CUBIC

the link for short periods and then lowers the sending rate to drain the queue. This is demonstrated in the throughput plot through the equally spaced spikes, which is shown in the original BBR showcase [5]. TCP CUBIC, on the other hand, shows a fast convergence and multiplicative decrease in its throughput. Since the plot is at the rate of 1 RTT the congestion window follows the pattern that is specific to the TCP CUBIC congestion window shown in the TCP CUBIC paper [4].

4 Effect on performance metrics

In this section, we evaluate how TCP's performance (loss and delay) is affected by the TCP flavour we use. We run 4 sets of experiments, each belonging to one of the TCP scenarios in Table 1. For each experiment, we vary the policer rate according to values of Table 1, and for each rate, we repeat the run 10 times which uses 5 different background traffic setups. We report the measurements collected from these results below.

4.1 Effect on loss

First, we compare the loss of the TCP BBR and CUBIC flows with both TCP BBR and CUBIC background traffic to determine how loss is affected by the congestion control algorithm. This comparison is shown in Figure 2.

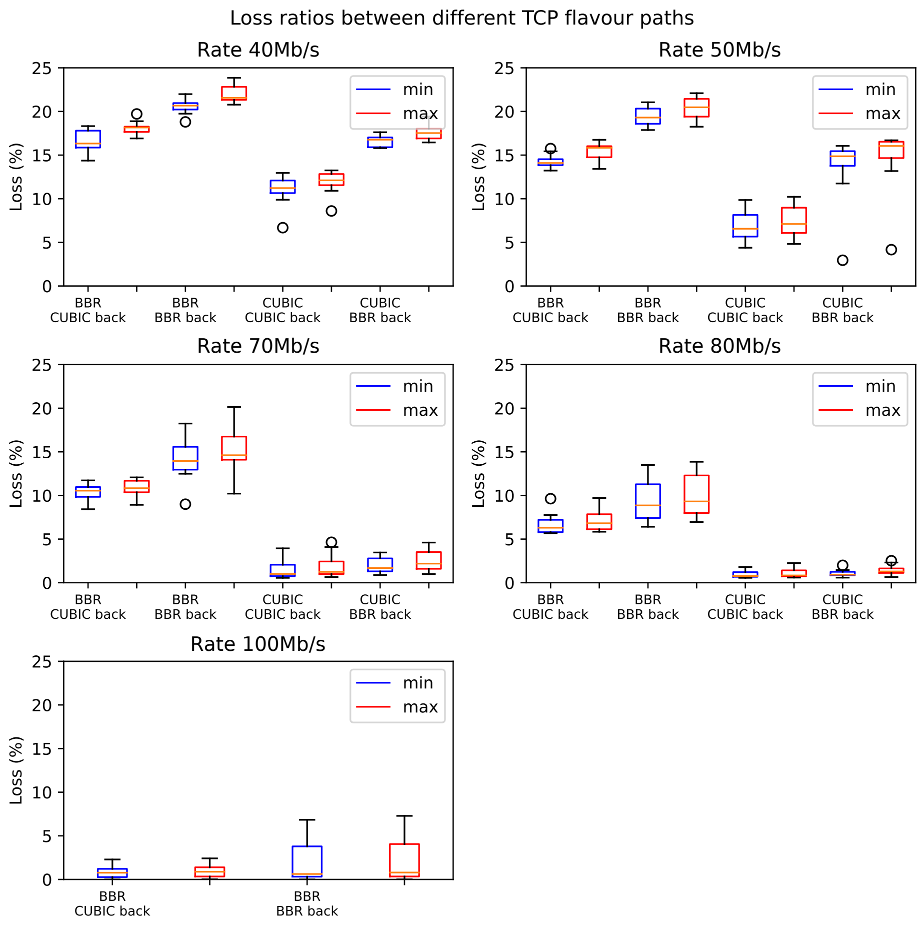

Figure 2: Average loss experienced by measurement TCP flows in the four TCP scenarios when traversing a common policer (blue boxplots refer to the min loss of the two flows and red refer to max).

Figure 2 shows that TCP BBR clearly experiences higher loss than CUBIC for every policer rate. We further confirm this by performing the Kolmogorov-Smirnov test on the two congestion control distributions for each rate for the same background. TCP BBR experiences more loss at all policer rates, the highest being at 40Mb/s rate with BBR background where it reaches a median of 22% loss, whereas CUBIC only reaches 18% median loss with BBR background at rate 40Mb/s.

Moreover, more loss is observed for both TCP BBR and TCP CUBIC when using TCP BBR as background traffic. This is because BBR is less fair to other traffic going through the same link when there is a lot of different traffic shown by Philip *et al.* [6]. However, it can also be noticed that TCP BBR is more affected by the background change, and its loss increases significantly for 40Mb/s, 50Mb/s, and 70Mb/s policer rates.

It is also important to note that between the measured flows there is little difference between the flow that experienced lower loss and the flow that experienced higher loss. This small difference means that the competition between flows is low, therefore the loss is the result of simulated traces competing with the background traffic. Most competition is observed at the policer rate of 40Mb/s for TCP BBR flows, but the difference between medians is minimal.

We run an extra experiment with a policing rate of 100Mb/s for TCP BBR only. The goal is to test BBR's performance under low congestion scenarios as we will show in the next section. BBR experiences 2% loss. We do not test this for CUBIC because CUBIC does not experience loss at this rate.

A technical challenge with inferring loss. One challenge we ran into during the experiment analysis was the high memory (RAM) consumption encountered while inferring the loss events from PCAP files which caused our program to halt. Currently, our analysis is implemented in Python, and to infer the loss events we use the pandas library with dataframe operations. The high memory use comes from the fact that to find the packet loss event we need to get a cross-product of the recorded dataframe with itself to get whether the packet was retransmitted. Our solution to reduce occupied memory was to split the data frame into smaller parts and then compute the cross-product.

4.2 Delay

Next, we compare the delay metric between the TCP BBR and CUBIC flows. To compute the delay experienced by a given flow, we take the packet's RTT (extracted from received ACKs) and subtract the minimum RTT experienced by all packets of this flow. The comparison is shown in Figure 3.

Unlike loss, where BBR experiences higher loss than CUBIC, BBR experiences lower delay than CUBIC according to the Kolgorov-Smirnov test for the same rate and background. This can be explained by the fact that BBR is a delay-based algorithm and adjusts its congestion window to minimize delay.

At rates 40 and 50Mb/s CUBIC with BBR background experiences a lot higher delay compared to all other TCP scenarios. This is because (1) in this scenario the two CUBIC flows are competing with many BBR flows, and (2) BBR flows manage their congestion window using the delay to avoid bufferbloat. This means many flows avoid filling the queue, and two flows keep trying to fill the queue before they experience loss. Therefore CUBIC flows with BBR background experience significantly more delay.

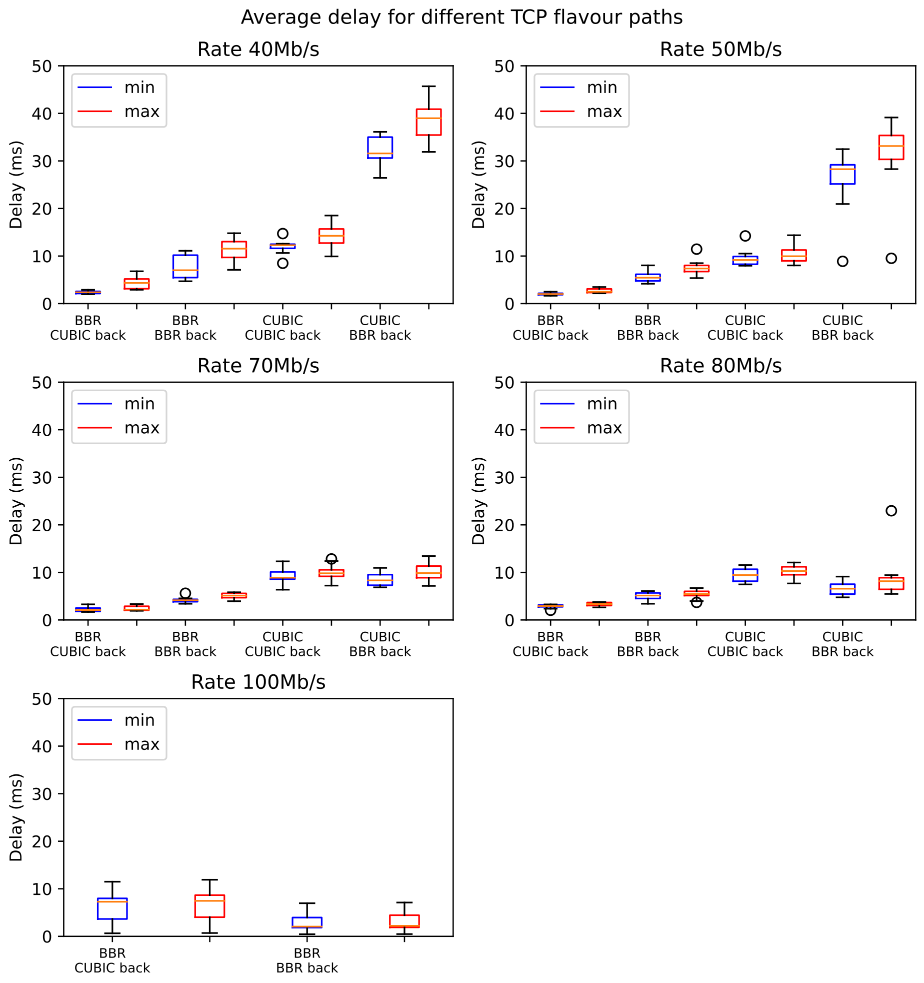

Figure 3: Average delay experienced by measurement TCP flows in the four TCP scenarios when traversing a common policer (blue boxplots refer to the min loss of the two flows and red refer to max).

5 Evaluating loss correlation

In this section, we test WeHe𝕐's loss trend correlation approach with TCP BBR. The approach was proven to work well for setups of TCP CUBIC with all backgrounds being TCP CUBIC but not the other three TCP scenarios.

Our evaluation is composed of two parts: first we compare the actual correlation values of the loss time-series experienced by the two flows under the different TCP scenarios, and second we compare the false negative/positive rates of the method for each scenario. The goal of the first is to determine whether we have more or less correlation with BBR compared to cubic. The goal of the second is to evaluate the method's overall performance.

For our first evaluation method, we use boxplots (as in Figure 4 and Figure 5) that show the best Spermans's rank correlation coefficient observed at a given experiment run. This is computed as follows: for a given run, we calculate the set of interval sizes (same as WeHe𝕐), and for each interval, we compute the loss ratios of both flows and apply Spearman's rank method to get the correlation coefficient. Then, we take the highest coefficient value where p-value < 0.05 or if there is no interval where this condition is satisfied, we take the highest correlation value.

For our next evaluation, we report the false negative and false positive rates. A false negative is when we fail to detect a common policer (section 5.1) and a false positive is when we falsely detect a common policer even though each TCP flow traverses a separate policer (section 5.2). Recall that we report flows to be correlated when Spearman's method outputs correlation for more than 50% of the intervals.

5.1 Common policer

First, we evaluate the loss-trend correlation performance when TCP flows traverse a common policer.

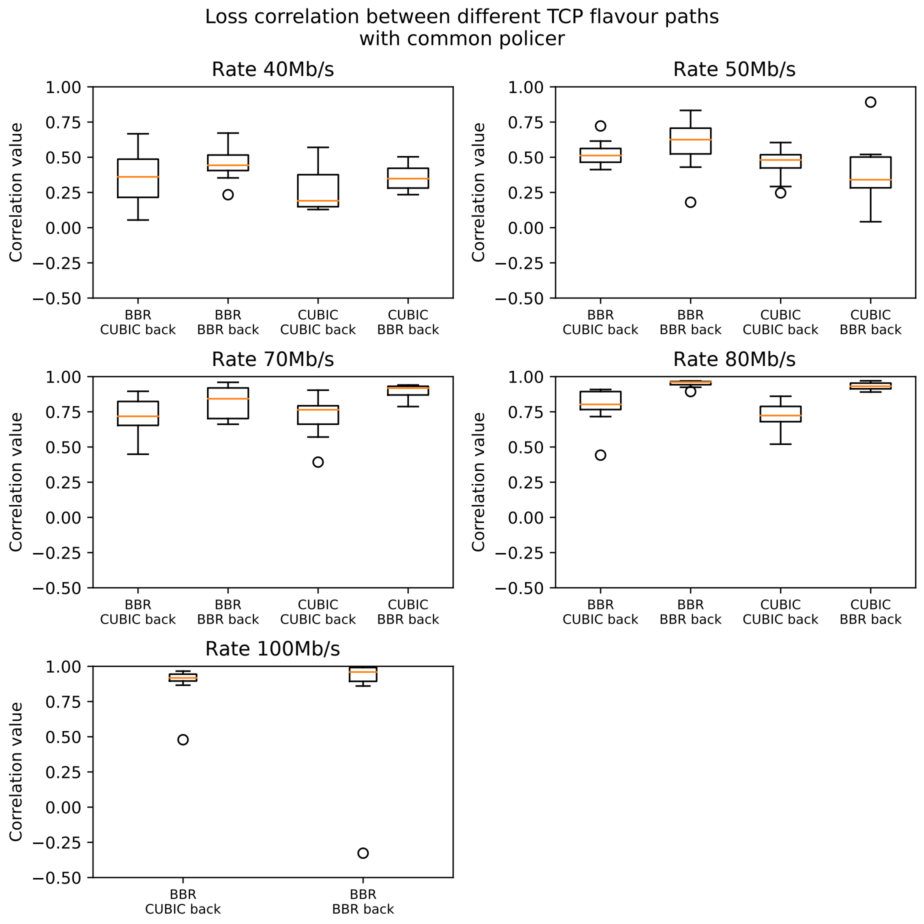

Figure 4: Distribution of the loss-trend correlation coefficient for TCP BBR and TCP CUBIC with common policer.

Figure 4 shows the best correlation coefficient for the loss experienced by the two flows when traversing a common policer and for the four TCP scenarios of Table 1. TCP flows experience less correlated loss under high congestion (40 and 50Mb/s) compared to medium congestion (70 and 80Mb/s).

Table 2 shows the fractions of false negative rates. It could be seen that BBR outperforms CUBIC by having fewer false negatives than CUBIC. This means that loss is still a good measure to show the correlation between two paths even when using TCP BBR congestion control.

	False Negatives (with 50% of intervals)	False Negatives (with 95% of intervals)
BBR with BBR	3/40 (1/10 for 100Mb/s)	11/40 (1/10 for 100Mb/s)
BBR with CUBIC	5/40 (0/10 for 100Mb/s)	15/40 (0/10 for 100Mb/s)
CUBIC with BBR	17/40	19/40
CUBIC with CUBIC	12/40	17/40

Table 2: False negative rate with common policer for each TCP scenario.

5.2 Non-common policer

Next, we test loss-trend correlation performance when TCP flows traverse separate policers. In this case, the correlation coefficient should be low and the p-value should be > 0.05. The correlation value is around 0.20 for both BBR and CUBIC in Figure 5, which is not low considering there should be no correlation. This can be explained by the fact that even though they do not share the same queue, they share the same client which means they can share part of the same path.

	False Positives (with 50% of intervals)	False positives (with 95% of intervals)
BBR with BBR	4/34 (2/10 for 100Mb/s)	3/34 (1/10 for 100Mb/s)
BBR with CUBIC	7/37 (0/10 for 100Mb/s)	4/37 (0/10 for 100Mb/s)
CUBIC with BBR	0/34	0/34
CUBIC with CUBIC	3/37	0/37

Table 3: False positive rate with non-common policer for each TCP scenario.

Since the p-values of Spearman's correlation coefficient must be > 0.05 for a non-common policer, any flows deemed correlated are a false positive. Contrary to the common policer loss correlation evaluation, BBR here performed worse than CUBIC as seen in Table 3 since it tends to generate a higher number of false positives when the policer is configured for non-common policies. Both algorithms had 9 flows with no loss events detected, so there are fewer flows to compare.

We think we have a high number of false positives because small parts of the flows could be correlated, for example, BBR probing spikes as seen in Figure 1. The WeHe𝕐 method could fail with BBR since it depends on the p-value of Spearman's test, which tests that correlation is consistently zero over the run duration. However, because of the probing spikes, there can be some correlation with BBR flows, and we see that as low correlation values in the boxplot.

To see whether our chosen threshold value is the one causing the high false positive count, we decided to raise the threshold from 50% to 95%. The result was that the BBR false positive

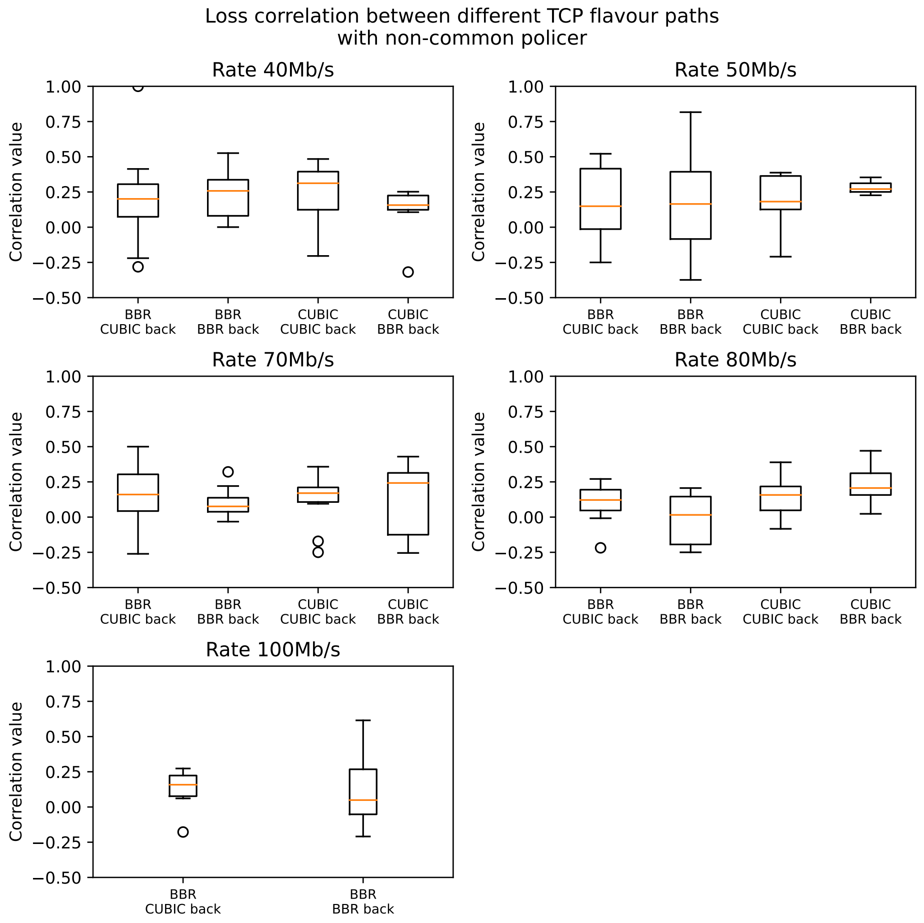

Figure 5: Loss correlation distribution of TCP BBR and TCP CUBIC with non-common policer.

count changed from 15% to 11%, and the CUBIC false positive count decreased from 4% to 0%. However, the BBR false negative count increased from 10% to 34%, and the CUBIC count rose from 36% to 45%. In general, the false negative rate of BBR was significantly lower than CUBIC's, but there were more false positives detected by BBR correlation.

6 Correlation for other metrics

To explore a different metric than loss we chose to test the Spearman correlation method on throughput. For this, the same experiment of TCP BBR and TCP CUBIC flows with different backgrounds and a common policer was used to compute throughput and its correlation. Since CUBIC is a loss-based congestion control algorithm and BBR is congestion-based, i.e., dependent on the bandwidth-delay product, it was interesting to see whether throughput would show the same correlation values as a loss.

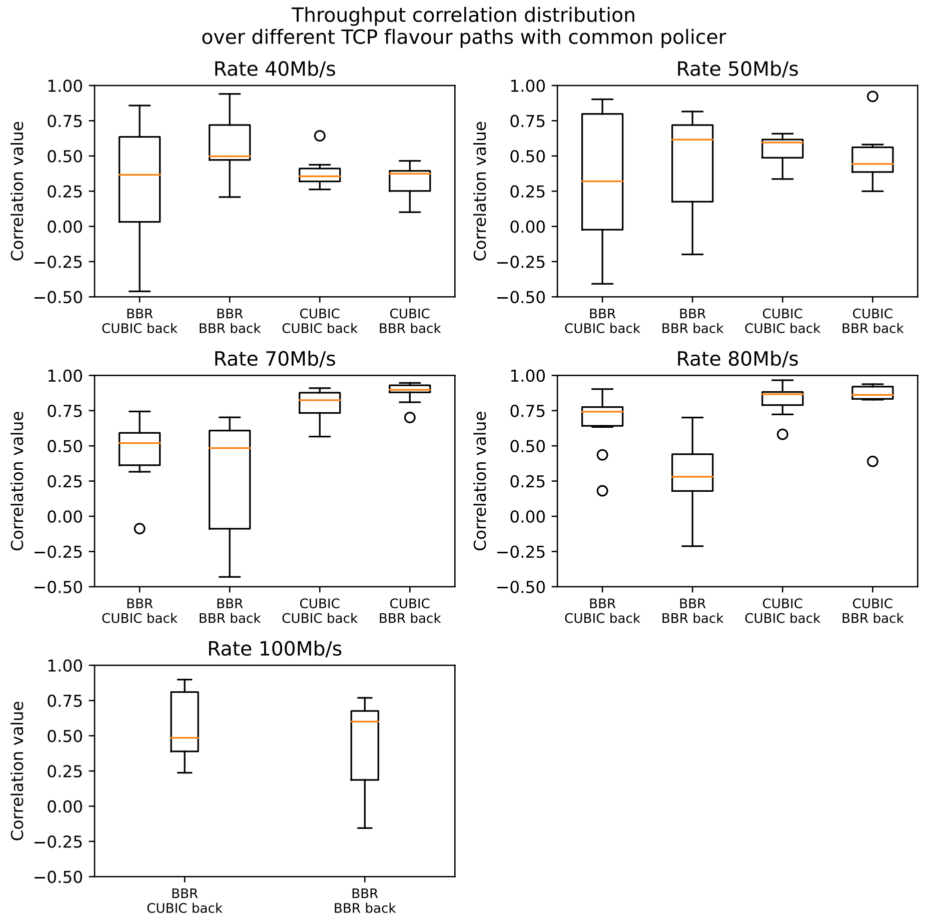

Figure 6: Throughput correlation distribution of TCP BBR and TCP CUBIC with common policer.

In Figure 6 we show the correlation of throughput under various rates of the experiment. Here we show the highest interval correlation value no matter the p-value to see the possible indication of correlation.

Comparing BBR correlation under different backgrounds, it can be seen that the median correlation for lower rates is higher with a BBR background, indicating that BBR has more consistent congestion management behaviour under the same background. One outlier, however, is BBR with a BBR background and a policer at a rate of 80Mb/s, where the correlation

	False Negatives	False Positives
BBR with BBR	13/40 (6/10 for 100Mb/s)	22/40 (3/10 for 100Mb/s)
BBR with CUBIC	11/40 (7/10 for 100Mb/s)	24/40 (2/10 for 100Mb/s)
CUBIC with BBR	11/40	14/40
CUBIC with CUBIC	8/40	13/40

Table 4: Throughput false positive and false negative rates with non-common policer for each TCP scenario.

is significantly lower than the one with the CUBIC background. BBR correlation does not seem to increase the same way CUBIC correlation does where the rate increase results in the rise correlation.

On the other hand, the spread of the CUBIC correlation is small compared to the BBR spread. For instance, BBR can show a high positive correlation as well as a negative correlation, whereas CUBIC only shows positive correlation behaviour. This means that throughput is not a reliable metric for detecting correlation using TCP BBR because it is not predictable.

To evaluate the correctness of throughput as a correlation metric, we computed the confusion matrix for both BBR and CUBIC flows, shown in Table 4. As already seen in correlation distribution, throughput does not reliably detect correlation when it should, which is confirmed by high false negative values for both BBR and CUBIC. In addition to that, BBR flows also show more correlation than not when they go through non-common policer, which is inaccurate. BBR 58% false positive rate in comparison to loss 15% rate shows how inaccurate throughput gets. CUBIC shows a similar tendency of having higher false values for both common and non-common policers compared to loss, however, the difference is not as high as BBR values.

7 Future Work

TCP BBR is a congestion control algorithm that is constantly under development, having its initial release in 2016, BBRv2 in 2021 [7], and the newest BBRv3 release in 2023 [8]. According to Google, BBRv3 is set to replace v1 in the Linux kernel as the official BBR version. The main reason for this is performance: its better coexistence with Reno and CUBIC, lower loss rates than v1, and lower latencies for short requests. After the new BBR version is introduced into the Linux kernel and wider use, there is potential to adapt our existing traffic localization algorithms to work better with BBR as well.

8 Conclusion

After analysing TCP BBR's loss and delay metrics we have found that the current BBR version experiences more loss than CUBIC. Despite that, BBR experiences less delay than CUBIC, especially in high-congestion scenarios, thus compensating for its higher loss with lower delay.

Comparing the loss correlation of BBR and CUBIC, we found that BBR has better accuracy than CUBIC when spotting that the two flows went through a common policer. Nevertheless, BBR has more false positives than CUBIC resulting in overall less accurate common policer detection.

Lastly, we have evaluated the throughput correlation of BBR and CUBIC. BBR did not show as high a correlation as CUBIC for less congested links and had a higher spread of correlation values. Because of this, we cannot come up with a technique to capture the correlation there.

References

[1] "At&t video management." (2024), [Online]. Available: https://www.att.com/support/article/wireless/KM1169198/.

[2] "At&t business fiber jumps in the fast lane with gigabit speeds." (May 2016), [Online]. Available: https://www.prnewswire.com/news-releases/att-business-fiber-jumps-in-the-fast-lane-with-gigabit-speeds-300266126.html.

[3] Z. Shmeis, M. Abdullah, P. Nikolopoulos, K. Argyraki, D. Choffnes, and P. Gill, "Localizing traffic differentiation," in *Proceedings of the 2023 ACM on Internet Measurement Conference*, ser. IMC '23, Montreal QC, Canada: Association for Computing Machinery, 2023, pp. 591–605, ISBN: 9798400703829. DOI: 10.1145/3618257.3624809. [Online]. Available: https://doi.org/10.1145/3618257.3624809.

[4] S. Ha, I. Rhee, and L. Xu, "Cubic: A new tcp-friendly high-speed tcp variant," *SIGOPS Oper. Syst. Rev.*, vol. 42, no. 5, pp. 64–74, Jul. 2008, ISSN: 0163-5980. DOI: 10.1145/1400097.1400105. [Online]. Available: https://doi.org/10.1145/1400097.1400105.

[5] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "Bbr: Congestion-based congestion control," *Communications of the ACM*, vol. 60, pp. 58–66, 2017. [Online]. Available: http://cacm.acm.org/magazines/2017/2/212428-bbr-congestion-based-congestion-control/fulltext.

[6] A. A. Philip, R. Ware, R. Athapathu, J. Sherry, and V. Sekar, "Revisiting tcp congestion control throughput models & fairness properties at scale," in *Proceedings of the 21st ACM Internet Measurement Conference*, ser. IMC '21, Virtual Event: Association for Computing Machinery, 2021, pp. 96–103, ISBN: 9781450391290. DOI: 10.1145/3487552.3487834. [Online]. Available: https://doi.org/10.1145/3487552.3487834.

[7] N. Cardwell and Gooble BBR team. "Bbrv2 update: Internet drafts & deployment inside google." (2021), [Online]. Available: https://datatracker.ietf.org/meeting/112/materials/slides-112-iccrg-bbrv2-update-00.

[8] N. Cardwell and Gooble BBR team. "Bbrv3: Algorithm overview and google's public internet deployment." (Apr. 2024), [Online]. Available: https://datatracker.ietf.org/meeting/119/materials/slides-119-ccwg-bbrv3-overview-and-google-deployment-00.