Reproducing and Validating ABR Results from Puffer using NetReplica

Parker Carlson, Rithwik Kerur, Wentai Xie

June 6, 2025

Abstract

Machine learning for networking is in the midst of a reproducability crisis stemming from years of underspecified models. In this project, we turn our attention to reproducing and validating the results of machine learning-based adaptive bitrate (ABR) algorithms. We compare Fugu, which uses a neural network to predict chunk transmission time to the heuristic-based LinearBBA. We find that Fugu slightly outperforms LinearBBA in terms of time spent rebuffering, but both methods perform strongly. Our results confirm that NetReplica is useful for comparing ABR methods in challenging network conditions without the costly data collection of an in-situ experiment.

1 Introduction

Machine learning is transforming all scientific domains with the promise of higher quality models that learn subtle patterns from vast quantities of data. However, the naive application of machine learning often leads to underspecified and unreliable models. Indeed, recent findings from Trustee [5] confirm that many machine learning models designed for networking are underspecified and fail to generalize, revealing an ongoing reproducibility crisis in networking. Previous machine learning-based approaches need to be revisited and reproduced to guide future development of robust data-driven methods. In this report, we examine a recent machine learning approach to the adaptive bitrate problem.

Adapative bitrate (ABR) algorithms are used to dynamically adjust the bitrate of online video streaming to maximize the end-user's quality of experience (QoE) over dynamic and potentially unstable network conditions. Major components of QoE include the video quality (typically measured through bitrate or resolution), changes to the resolution, initial buffering time, and rebuffering time. Though different users may place greater value on different aspects of QoE, a robust ABR algorithm should balance different QoE components while delivering strong performance across many unique network conditions.

Evaluating ABR algorithms in similar conditions is challenging. Puffer is a large-scale experiment to compare the performance of ABR algorithms in a real-world setting [9]. Puffer broadcasts basic network television stations over the internet and randomly selects an ABR algorithm for each user session. Through long-term and wide-scale data collection, these methods can be compared in a diverse set of network conditions and with different QoE metrics. However, because of the randomized nature of Puffer, different ABR schemes cannot be tested under identical network conditions.

In this paper we provide a direct comparison between two ABR techniques using the NetReplica data collection framework [3]. We compare a linear buffer-based approach (Linear-BBA) to Puffer's Fugu model, a machine learning model trained on Puffer's data. We identify the relative strengths and weaknesses of these algorithms under nearly-identical network conditions. We find that Fugu slightly outperforms LinearBBA in terms of time spent rebuffering, and our findings highlight the utility of NetReplica in replicating the findings of a costly large-scale randomized control trial.

2 Related Work

Adaptive Bitrate Algorithms

There are many traditional (i.e. non-ML) and machine learning based approaches to the ABR problem. Notable recent work on ABR includes MPC [10], which creates a network model based on optimal control theory, solves for the optimal bitrate for each network condition offline, and then uses a lookup table during runtime to adjust bitrate. Oboe builds upon MPC, in addition to computing theoretically optimal parameters offline, Oboe varies the parameters at runtime to better adapt to observed network conditions [1].

Many ABR methods build atop MPC by incorporating a learned component to model network conditions. For instance, Fugu [9] uses a learned chunk transmission time predictor, with the time to transmit each chunk used as input to MPC to select the optimal chunk bitrate. Similarly, BayesMPC introduces a Bayesian neural network to model the uncertainty in future network conditions when predicting throughput, which is used in tandem with MPC to select the optimal bitrate [6].

Other approahes to ABR include BOLA, which models ABR as a utility maximization problem, attempting to maximize bandwidth while minimizing rebuffering using the buffer time as its input feature [8]. LinearBBA [4] bases its selection of bitrate on the remaining duration in the buffer. In steady-state, it does not need to estimate network throughput, though it is necessary when the buffer is first filling or nearly depleted. Penseive [7] models ABR selection as a reinforcement learning problem, directly learning the optimal bitrate based on previous network features (e.g. prior throughput, transmission time, bitrate, ect.).

Direct Comparison of Network Algorithms As there are many approaches to the ABR problem, it is challenging to directly compare each method in a fair setting. Puffer attempts to address this by running a large-scale randomized control trial of ABR algorithms [9], however any new ABR algorithm must be added to Puffer and be evaluated over a long period to validate its performance. This increases the research and development timeline Further, Puffer has no control over the observed network conditions nor the user's selection of channel, which necessitates its long-term data collection to compare different algorithms in similar conditions, and precludes direct apples-to-apples comparisons.

Instead of using experimental results from Puffer, it is reasonable to attempt to directly compare ABR methods using a simulator and existing network trace data. Previous works have used a simple chunk-level simulator, sometimes referred to as ExpertSim, which replays throughput from a network trace [7, 10]. However, this assumes that the choice of bitrate is independent of network conditions, which is nearly always false. CausalSim accounts for this by modeling the effect of bitrate choice on the network conditions in the network simulator [2]. However, CausalSim is still limited by replaying existing network traces, and the majority of Puffer's data was collected in "good" network conditions, limiting comparison of ABR methods on real-world "challenging" network conditions (i.e. low throughput, high latency, with lots of cross traffic). NetReplica addresses this limitation, allowing testing with direct selection of a target network environment while maintaining the causality of ABR algorithms [3].

3 Methodology

Based on initial exploratory data analysis, we select two ABR algorithms to compare in-depth under similar network conditions. LinearBBA is a representative heuristic-based approach that achieves good performance in the Puffer experiment. LinearBBA maintains a lower and upper buffer reservoir to maintain a consistent bitrate and prevent buffering. If the upper reservoir is depleted, a lower bitrate is selected based on how quickly the buffer is depleting. The lower reservoir serves as a buffer to prevent rebuffering under network variations. We follow Puffer in configuring LinearBBA with a lower reservoir of 20% of the total buffer size, and an upper reservoir of 10% of the total buffer size. We compare LinearBBA to Puffer's Fugu, which uses MPC control built upon a neural network that predicts chunk transmission time. Fugu is regularly trained in-situ on Puffer data - thus, one expects that it will perform the best for Puffer, though it is also liable to underspecification or overfitting.

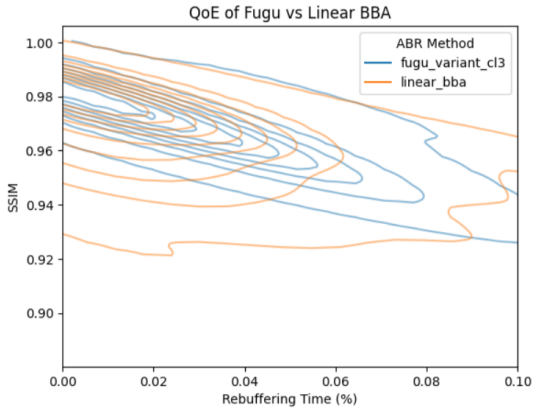


Figure 1: *QoE of Fugu vs Linear BBA from the Puffer dataset.*

To directly compare LinearBBA and Fugu in a near-identical setting we turn to NetReplica, running a version of Puffer's code. We test each algorithm for 60 seconds at a time under the same network conditions for each channels. We configure NetReplica to use a bandwidth of 6 Mbit/s, following Puffer's definition of "slow" networks, a latency of 80 ms, and a variety of realistic cross-traffic profiles. We arrived on these values with the help of Jaber Daneshamooz.

4 Results

Exploratory Data Analysis

To select which methods to compare in-depth, we performed exploratory data analysis to find a high performing non-ML algorithm. Highlights are presented here; full results can be found in Appendix A.

Figure 1 presents an overall comparison of the Quality of Experience (QoE) of LinearBBA and Fugu on the Puffer dataset, measured in terms of SSIM and rebuffering. Both LinearBBA and Fugu achieve high quality streams, with near-perfect SSIMs and fractions of a percent of the total stream time spent rebuffering. Both LinearBBA and Fugu have a similar distribution of rebuffering time; however, LinearBBA has more variation in terms of SSIM. Thus, Fugu may offer a slightly improved QoE without affecting rebuffering time.

Figure 2 compares the time spent rebuffering the the average buffer size maintained by each method. Here, two different versions of Fugu are presented. trained on

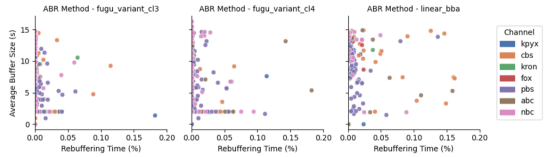


Figure 2: *Average buffer size versus time spent rebuffering by method for the Puffer dataset. There is no clear relationship between maintaining a full buffer and preventing rebuffering.*

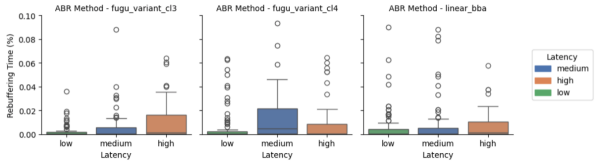


Figure 3: *Performance of ABR algorithms on Puffer under different amounts of latency. Fugu methods tend to rebuffer more under medium and high latency.*

different subsets of data. One may expect that sessions that maintained a lower average buffer size may experience more rebuffering; however, there is no clear trend between average buffer size and time spent rebuffering.

Figures 3 and 4 examines how network conditions affect Fugu and LinearBBA. As expected, both methods perform the worst in low throughput scenarios. However, LinearBBA outperforms both versions of Fugu for medium-throughput streams. This suggests that LinearBBA may be more robust in low-resource scenarios. Generally, high latency session were the most challenging for all methods, however Fugu-CL4 performed better for high latency sessions than medium latency sessions. This suggests that Fugu's TTP may be under-specified because two different training iterations generalize to significantly different behaviors on medium latency sessions.

LinearBBA versus Fugu

Now we will compare LinearBBA to Fugu using NetReplica in near-identical network conditions.

In Figure 5, we find that LinearBBA and Fugu perform similarly in challenging network environments. Both methods spend about 1% of total stream time rebuffering, with a heavy tail.

In Figures 6 and 7, we investigate if either method is sensitive to variations in network quality. Neither method seems sensitive to the changes in available throughput, as indicated by the flat trends observed in Figure 6. However, there is a slight positive trend for both LinearBBA and Fugu under extreme variations in network latency (i.e. a standard deviation in RTT above 600 ms).

In our initial experiments, we found that LinearBBA significantly outperformed Fugu in terms of rebuffering performance, both in general and under high latency variation. We initially found that Fugu attempted to select high bitrates too aggressively, contributing to network congestion. However, that appears to have been caused by a

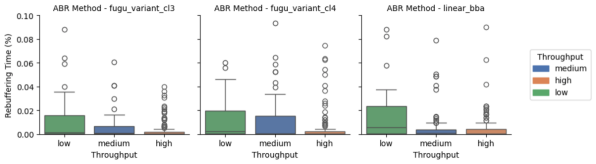


Figure 4: *Performance of ABR algorithms on Puffer under different amounts of throughput. All methods reubffer the most under low throughput, except the 2019 version of Fugu, denoted as puffer_ttp.*

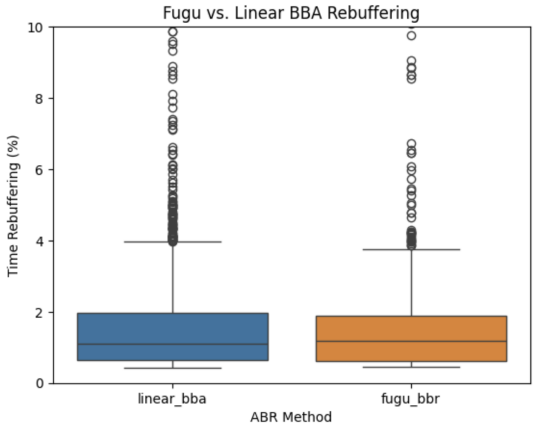


Figure 5: *Fugu vs Linear BBA on Rebuffering Performance*

data analysis error and not actual ABR performance. This underscores the importance of careful data analysis by domain experts in developing machine learning models.

For both Fugu and LinearBBA we plot the rebuffering percentage compared to the average bitrate in Figure 8. Additionally, we create different graphs for each format since videos in higher quality will need to send more data. A video in a lower quality may be underutilizing throughput compared to a video in a higher quality. In the Fugu BBR plot, nearly all sessions lie within a tight band of 0–10 % rebuffering, where the rebuffering percentage remains low while the average bitrate rises as high as 2.2 bps. Since the average bitrate is calculated by averaging each chunk's bitrate over all acknowledged chunks, Fugu's BBR-based ABR logic evidently maintains steady buffer levels as most sessions avoid buffer underruns and sustain throughput well above 1 bps. Interestingly, the lower quality videos exhibit more variance compared to the higher quality ones. One explanation for this could be that lower quality is chosen when the network conditions are worse and thus they would experience greater variance. By contrast, the higher resolution videos are only chosen during ideal network conditions when there is enough throughput.

The Linear BBA scatter plots in Figure 9 are similar to the FuGu but have some slight differences While there is still a dense cluster at 0–10 % rebuffering with bitrates up to 2 bps, a significant number of sessions shoot out

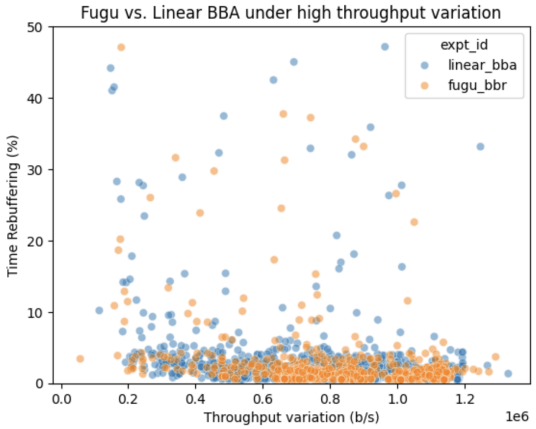


Figure 6: *Fugu vs Linear BBA under variable network throughput.*

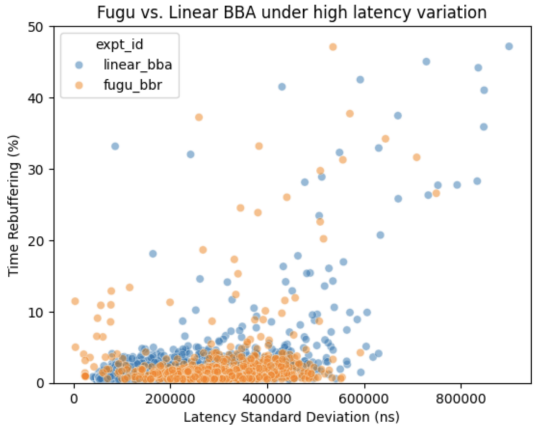
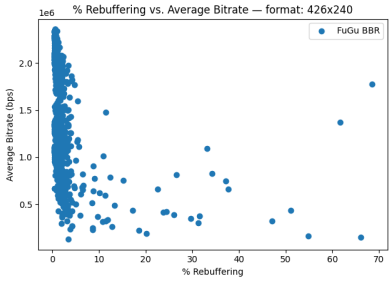


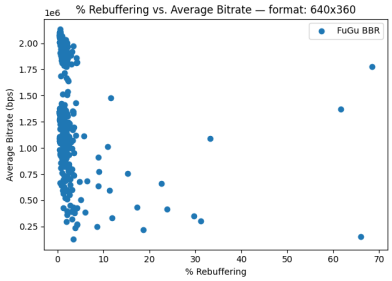
Figure 7: *Fugu vs Linear BBA under variable network latency.*

to 20–100% rebuffering. Linear BBA has a much higher variance compared to Fugu for sessions with the same video quality. One exception to this is the lowest quality 426x240 where Linear BBA has a significantly less variance than FuGu. One explanation for this is that LinearBBA has a much more conservative approach which would allow it to perform better in low resolution which is typically caused by bad network conditions. Fugu BBR's congestion-aware pacing keeps the rebuffering percentage low (mostly under 10 %) while maintaining an average bitrate near or above 1 bps, yielding a tight cluster. However, linear BBA stepwise increases often overshoot link capacity, resulting in many sessions with a higher rebuffering percentage and an outlier at 300%. Thus, Fugu BBR achieves more stable buffer occupancy and higher sustained throughput, whereas Linear BBA shows greater variance, frequent buffer-starvation events, and significantly reduced average bitrate in its high-rebuffer tail.

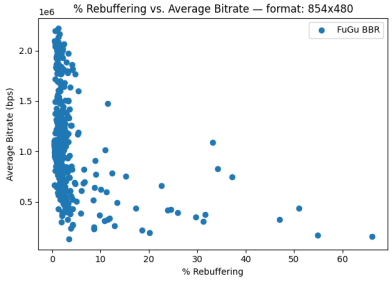
By testing the ABRs on data from NetReplica we are able to see how the ABR algorithms perform in more realistic scenarios. Getting real network data is difficult,



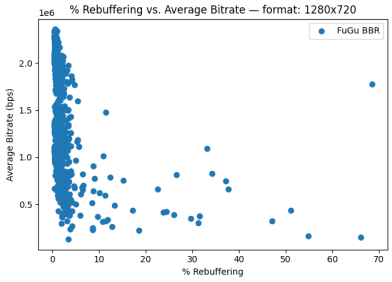
(a) *426×240*



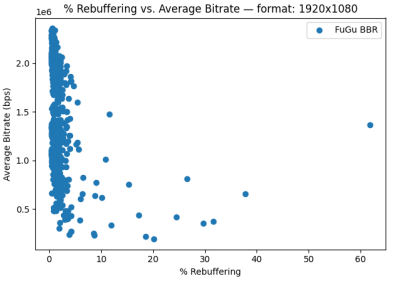
(b) *640×360*



(c) *854×480*

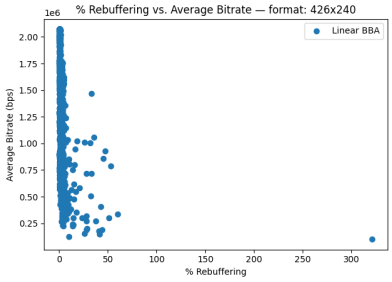


(d) *1280×720*

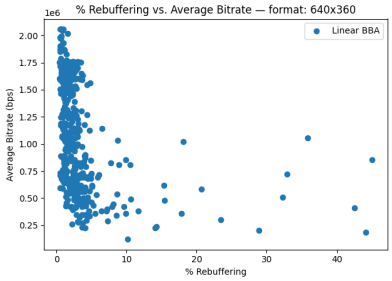


(e) *1920×1080*

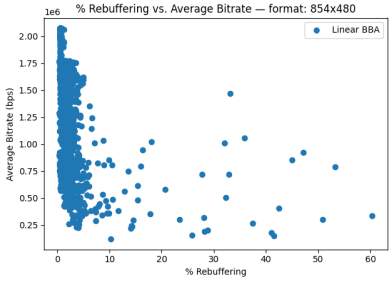
Figure 8: *% Rebuffering vs. Average Bitrate for FuGu BBR*



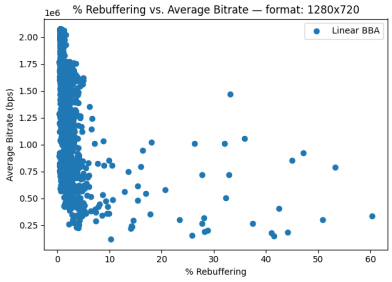
(a) *426×240*



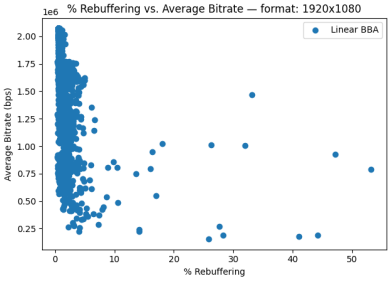
(b) *640×360*



(c) *854×480*



(d) *1280×720*



(e) *1920×1080*

Figure 9: *% Rebuffering vs. Average Bitrate for Linear BBA*

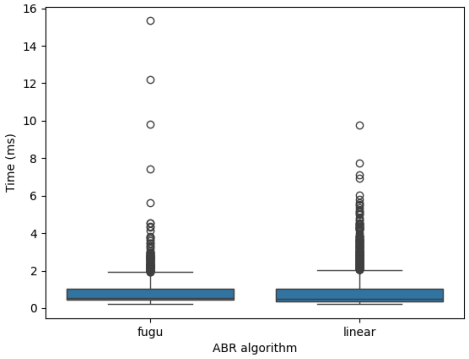


Figure 10: *Setup latency for different ABRs*

but by testing on data from NetReplica we can say with a higher confidence that Fugu performs slightly better than LinearBBA showing that a machine learning based approach can provide some benefits compared to a linear approach. Furthermore compared to the Puffer Data, the data from NetReplica exhibits much less ideal network conditions which allows us to properly test the ABR algorithms.

Setup Latency

Besides the performance of the ABR algorithm, the efficiency is a non-trival element in actual practice. We also collect the setup latency from the data. As shown in Figure 10, linear implies a smaller setup latency compared to Fugu; this is expected since Fugu envolves the ML process. However, from the figure the overall setup latency difference is marginal, suggesting that Fugu has a better overall performance.

5 Discussion

In summary, both Fugu and LinearBBA perform very similarly, but there are some slight differences between them. Fugu BBR's congestion-aware pacing keeps the rebuffering percentage low (mostly under 10%) while maintaining avg_bitrate_bps near or above 1 Mbps, yielding a tight, high-throughput cluster. Linear BBA's stepwise increases, however, often overshoot link capacity, resulting in many sessions with a higher rebuffering percentage and one outlier at 300%. Thus, Fugu BBR achieves more stable buffer occupancy and higher sustained throughput, whereas Linear BBA shows greater variance, frequent buffer-starvation events, and significantly reduced average bitrate in its high-rebuffer tail. By testing the ABRs on data from NetReplica we are able to see how the ABR algorithms perform in a realistic user-defined scenario. Getting real network data is difficult, but by testing on data from NetReplica we can say with a higher confidence that Fugu performs slightly better than LinearBBA showing that a machine learning based approach can provide some benefits compared to a linear approach.

6 Conclusion

We find that NetReplica's real environment with simulated cross-traffic matches the findings of a large-scale controlled random experiment for comparing ABR algorithms. Our findings show that LinearBBA and Fugu are both strong ABR algorithms, but Fugu may slightly outperform LinearBBA in challenging network conditions.

References

[1] AKHTAR, Z., NAM, Y. S., GOVINDAN, R., RAO, S., CHEN, J., KATZ-BASSETT, E., RIBEIRO, B., ZHAN, J., AND ZHANG, H. Oboe: auto-tuning video abr algorithms to network conditions. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication* (New York, NY, USA, 2018), SIGCOMM '18, Association for Computing Machinery, p. 44–58. (Cited on page 1.)

[2] ALOMAR, A., HAMADANIAN, P., NASR-ESFAHANY, A., AGARWAL, A., ALIZADEH, M., AND SHAH, D. {CausalSim}: A causal framework for unbiased {Trace-Driven} simulation. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)* (2023), pp. 1115–1147. (Cited on page 2.)

[3] DANESHAMOOZ, J., NGUYEN, J., CHEN, W., CHANDRASEKARAN, S., GUTHULA, S., GUPTA, A., GUPTA, A., AND WILLINGER, W. Addressing the ml domain adaptation problem for networking: Realistic and controllable training data generation with netreplica. (Cited on pages 1 and 2.)

[4] HUANG, T.-Y., JOHARI, R., MCKEOWN, N., TRUNNELL, M., AND WATSON, M. A buffer-based approach to rate adaptation: evidence from a large video streaming service. In *Proceedings of the 2014 ACM Conference on SIGCOMM* (New York, NY, USA, 2014), SIGCOMM '14, Association for Computing Machinery, p. 187–198. (Cited on page 2.)

[5] JACOBS, A. S., BELTIUKOV, R., WILLINGER, W., FERREIRA, R. A., GUPTA, A., AND GRANVILLE, L. Z. Ai/ml and network security: The emperor has no clothes. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2022), CCS '22, Association for Computing Machinery. (Cited on page 1.)

[6] KAN, N., LI, C., YANG, C., DAI, W., ZOU, J., AND XIONG, H. Uncertainty-aware robust adaptive video streaming with bayesian neural network and model predictive control. In *Proceedings of the 31st ACM Workshop on Network and Operating Systems Support for Digital Audio and Video* (New York, NY, USA, 2021), NOSSDAV '21, Association for Computing Machinery, p. 17–24. (Cited on page 1.)

[7] MAO, H., NETRAVALI, R., AND ALIZADEH, M. Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (New York, NY, USA, 2017), SIGCOMM '17, Association for Computing Machinery, p. 197–210. (Cited on page 2.)

[8] SPITERI, K., URGAONKAR, R., AND SITARAMAN, R. K. Bola: Near-optimal bitrate adaptation for online videos. *IEEE/ACM Transactions on Networking 28*, 4 (2020), 1698–1711. (Cited on page 2.)

[9] YAN, F. Y., AYERS, H., ZHU, C., FOULADI, S., HONG, J., ZHANG, K., LEVIS, P., AND WINSTEIN, K. Learning in situ: a randomized experiment in video streaming. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)* (2020), pp. 495–511. (Cited on pages 1 and 2.)

[10] YIN, X., JINDAL, A., SEKAR, V., AND SINOPOLI, B. A control-theoretic approach for dynamic adaptive video streaming over http. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication* (New York, NY, USA, 2015), vol. 45 of *SIGCOMM '15*, Association for Computing Machinery, p. 325–338. (Cited on pages 1 and 2.)

A Additional Results from Puffer's Data

Here, we present additional figures from our exploratory data analysis. These figures compare ABR methods under additional settings not included in our main report, as well as compares other ABR methods beyond Fugu and LinearBBA.

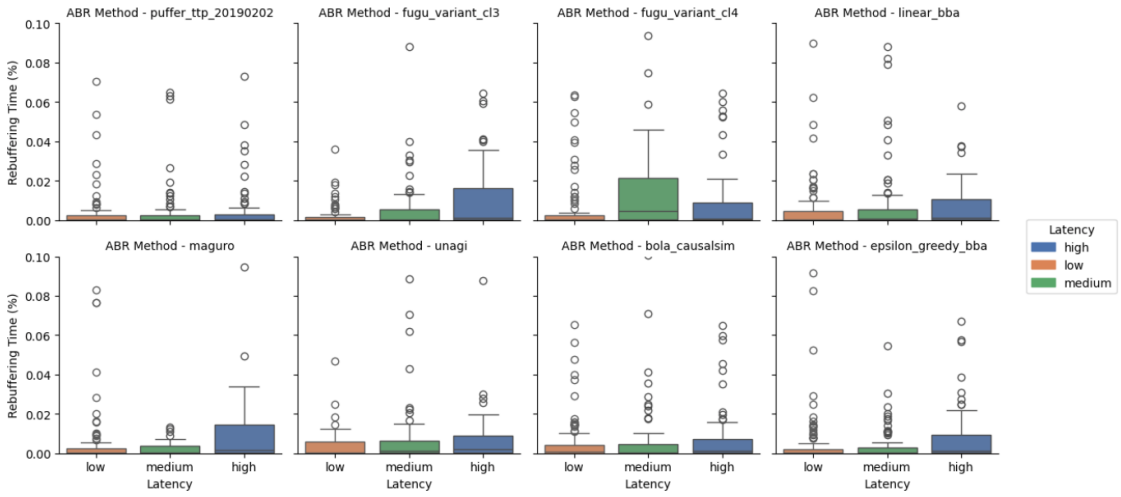


Figure 11: *Performance of ABR algorithms on Puffer under different amounts of latency. Fugu methods tend to rebuffer more under medium and high latency.*

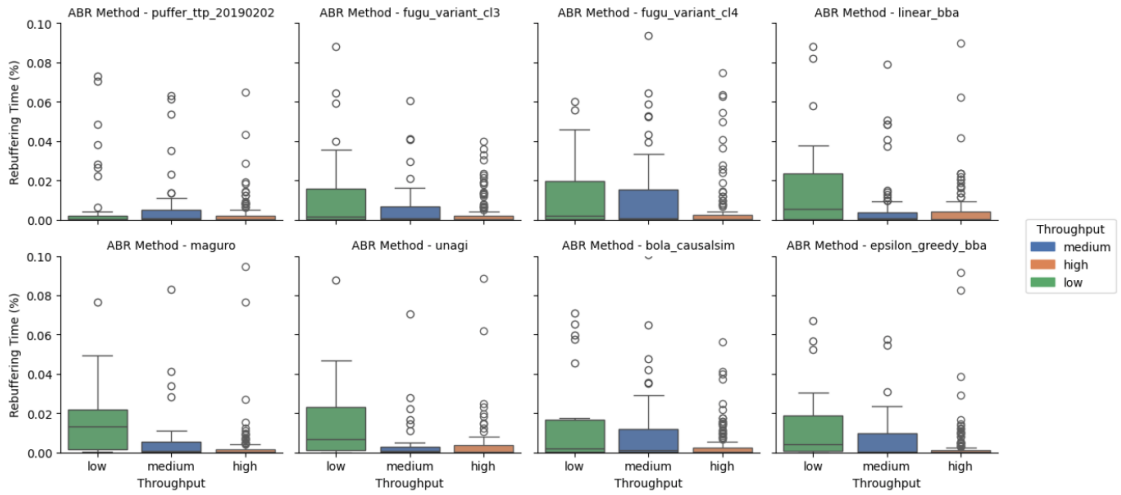


Figure 12: *Performance of ABR algorithms on Puffer under different amounts of throughput. All methods reubffer the most under low throughput, except the 2019 version of Fugu, denoted as puffer_ttp.*

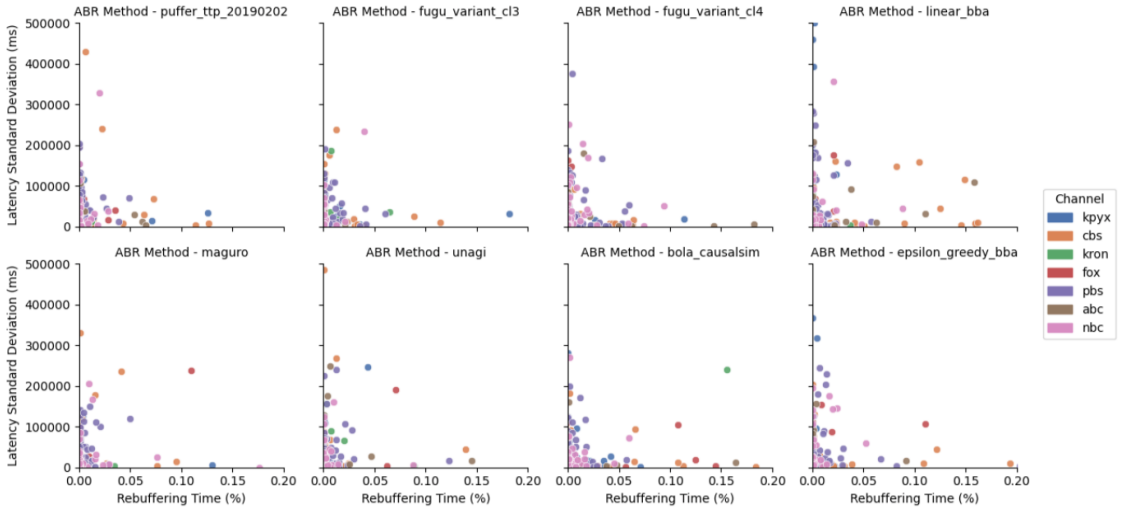


Figure 13: *Effect of latency variation on ABR rebuffering performance.*

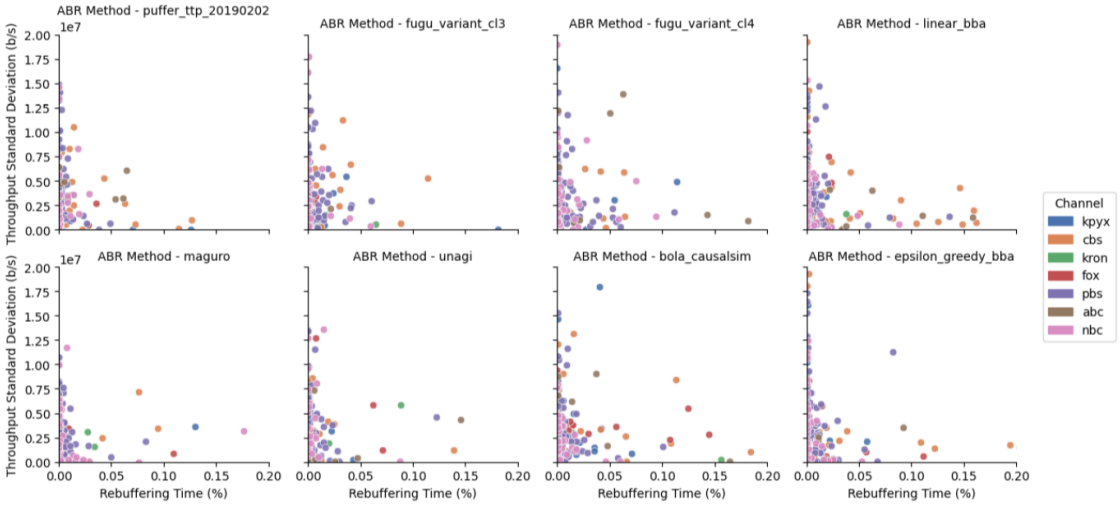


Figure 14: *Effect of throughput variation on ABR rebuffering performance.*

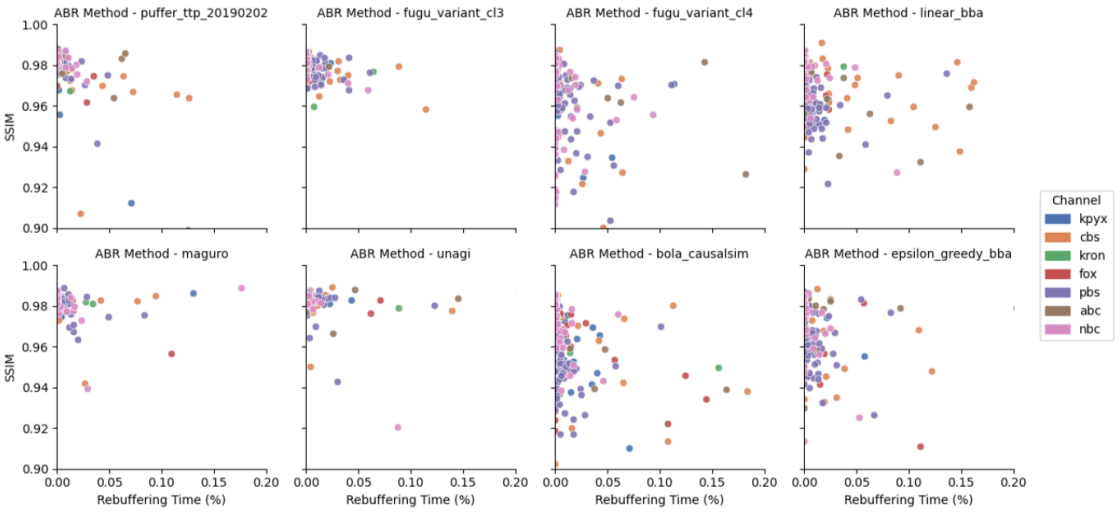


Figure 15: *ABR QoE by channel.*